

# End-zu-End-Verschlüsselung

FÜR EINE MESSAGING-WEB-APP

MATTIA METZLER

Maturajahrgang 2022

Betreuung: Marco Schmalz

Fach: Informatik

Gymnasium Neufeld, Abteilung MN, Klasse 22Mc



## Abstract

Diese Arbeit beschäftigt sich mit der End-zu-End-Verschlüsselung, einem heutzutage immer wichtiger werdenden Gebiet der Kryptographie, der Lehre der Verschlüsselung. Dieses Teilgebiet der Informatik wird theoretisch auf einfache Weise erklärt und wurde in einer Web-Applikation umgesetzt.

Erklärt werden Beispiele der Klassischen Verschlüsselung, der End-zu-End-Verschlüsselung sowie einige wichtige mathematische Aspekte und der Diffie-Hellman-Schlüsselaustausch, ein grundlegendes Protokoll der modernen Kryptographie.

Der Fokus dieser Arbeit liegt jedoch auf der parallel entwickelten Web-Applikation, welche eine Art der End-zu-End-Verschlüsselung, die sogenannte Hybrid-Kryptographie, unterstützt durch einen Diffie-Hellman-Schlüsselaustausch, implementiert. Die Web-App sowie der Server, wurden selbst geschrieben, basieren jedoch teilweise auf aussenstehenden *Frameworks*. Auch der Verschlüsselungsalgorithmus wurde nicht selbst geschrieben, es handelt sich um den ChaCha20-Verschlüsselungsalgorithmus.

In der entstandenen Web-App können sich Benutzer:innen in dem Netzwerk einen Benutzernamen zuordnen und dann miteinander kommunizieren. Diese Kommunikation kann entweder unverschlüsselt mit allen Teilnehmer:innen geschehen oder verschlüsselt mit einzelnen Teilnehmer:innen. Bei verschlüsselter Kommunikation werden die Nachrichten zwar trotzdem an Alle geschickt, jedoch liegen sie nur bei Sender:in und Empfänger:in unverschlüsselt vor.

Jede:r Benutzer:in kann selbst entscheiden, mit welchen Benutzer:innen verschlüsselt kommuniziert werden soll, da der Schlüsselaustausch manuell initialisiert und bestätigt wird.

Ebenfalls dargelegt in dieser Arbeit ist der Weg der Entwicklung der Web-App, welcher mehrere Prototypen beinhaltet.

Um die erklärten Konzepte zu verstehen ist kein Vorwissen im Gebiet der Kryptographie nötig, jedoch ist ein grundlegendes Verständnis von Algebra und ein gewisses Interesse für Informatik von Nutzen.

Die Web-App ist open-source und kann auf GitHub<sup>1</sup> heruntergeladen werden. Dort kann ebenfalls der Verlauf der Entwicklung eingesehen werden.

Für eine Anleitung zum Starten der Web-App siehe Kapitel B.1.

Alle in der folgenden Arbeit *schräggeschriebenen* Wörter werden im Glossar auf Seite 28 erklärt.

## Information

Die Wörter *Benutzer:in*, *Sender:in* sowie *Empfänger:in* und Ähnliche werden in dieser Arbeit oft verwendet. Sie wurden in dieser Arbeit bewusst teils männlich, teils weiblich verwendet, um somit alle Geschlechter einzuschliessen. Das Gendern mit Doppelpunkt würde in manchen Abschnitten die Lesbarkeit hindern.

---

<sup>1</sup> Erhältlich unter: <https://github.com/MaGaMe19/Maturaarbeit-App>



# Inhaltsverzeichnis

<b>Abstract</b>	<b>3</b>
<b>1. Einleitung</b>	<b>6</b>
<b>2. Zielsetzung</b>	<b>7</b>
<b>3. Theorie</b>	<b>8</b>
3.1. Klassische Verschlüsselung	8
3.1.1. Caesar-Verschlüsselung	8
3.1.2. Vigenère-Verschlüsselung	9
3.2. End-zu-End-Verschlüsselung	10
3.2.1. Symmetrische Kryptographie	10
3.2.2. Asymmetrische Kryptographie	11
3.2.3. Hybrid-Kryptographie	11
3.3. Mathematik, Algorithmen und Protokolle	12
3.3.1. Modulo-Arithmetik	12
3.3.2. Square-and-Multiply-Algorithmus	13
3.3.3. Diffie-Hellman-Schlüsselaustausch (DHM-Protokoll)	14
<b>Implementierung</b>	<b>16</b>
<b>4. Material und Methoden</b>	<b>16</b>
4.1. (Web-) Server	16
4.1.1. Funktionsweise	16
4.1.2. <code>api_utils.py</code>	16
4.1.3. <code>server.py</code>	16
4.1.4. Lokale Dateien	17
4.2. Client	17
4.2.1. <code>Vue.js</code>	17
4.2.2. Local Storage	18
4.3. Kommunikation und Verschlüsselung	18
4.3.1. Kommunikation - <code>Axios</code>	18
4.3.2. Verschlüsselung - <code>jschacha20.js</code>	18
<b>5. Der Weg zur Web-Applikation</b>	<b>19</b>
5.1. Prototypen	19
5.1.1. Verschlüsselung mit <code>Fernet</code> in Python	19
5.1.2. User Interface	20
5.1.3. DHM-Schlüsselaustausch und Verschlüsselung mit <code>jschacha20.js</code>	20
5.2. Zeitleiste	21
5.3. Back-End - <code>server.py</code>	21
5.4. Front-End - <code>app.html</code> & <code>app-style.css</code>	21
<b>6. Endprodukt und Testing</b>	<b>23</b>
6.1. Testing	24
<b>7. Evaluation und Bemerkungen</b>	<b>25</b>
7.1. Erreichte Ziele	25
7.2. Zukunft der Web-App	25
7.3. Bemerkungen	25
<b>8. Schlusswort</b>	<b>26</b>
<b>9. Links, Quellen und Literatur</b>	<b>27</b>
<b>A. Glossar</b>	<b>28</b>
<b>B. Anhang</b>	<b>31</b>
1. Anleitung	31
2. Quellcode	36

## 1. Einleitung

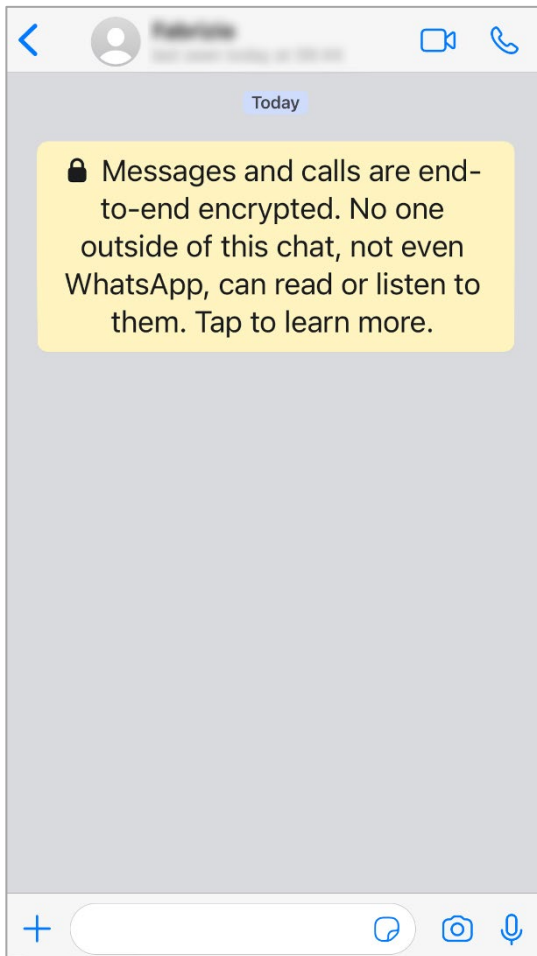


Abbildung 1: Screenshot aus der App WhatsApp

Beginnt man eine neue Unterhaltung auf *WhatsApp*, einer der meistgenutzten Messaging-Apps, erhält man die linksstehende Mitteilung. Das Adjektiv „end-to-end-encrypted“ bedeutet auf Deutsch „end-zu-end-verschlüsselt“. Dies sagt jedoch nicht sehr viel aus. Aus diesem Grund liefert WhatsApp eine kurze Erklärung, welche schon besser wirkt. Durch Klicken auf diese Nachricht kann eine ausführlichere Erklärung im Web angesehen werden, welche das Ganze aber leider sehr vereinfacht erklärt.

Es werden zwar die Grundbausteine der End-zu-End-Verschlüsselung erwähnt, jedoch werden deren genauen Zusammenhänge nicht erklärt.

Dies brachte mich dazu, eigene Nachforschungen zu der Kryptographie, der Lehre der Verschlüsselung, zu betreiben. Schlussendlich führte dies dazu, dass ich dieses Gebiet der Informatik als Thema meiner Maturaarbeit bestimmte.

Mein Ziel dieser Arbeit ist nicht, die sicherste Messaging-App der Welt zu erstellen, sondern mithilfe einer guten Implementation das Konzept der End-zu-End-Verschlüsselung auf relativ einfache Weise zu erklären. Es ist meiner Meinung nach sehr wichtig, dass Personen, welche solche Messaging-Apps nutzen, wissen, wie diese in etwa funktionieren. Dies lässt sich aber auch auf unzählige andere Apps oder Geräte anwenden, da Kryptographie mittlerweile allgegenwärtig ist.

### Danksagung

Mein herzlicher Dank gilt Marco Schmalz, meinem ehemaligen Mathematik- und Informatiklehrer sowie Betreuer dieser Arbeit. Dank wöchentlichen Besprechungen war ich immer auf dem richtigen Weg und erhielt viel Unterstützung sowie viele gute Ideen.

Ohne seine Hilfe hätte ich auch keinen Zugriff auf das *Python-Framework api\_utils.py* gehabt, welches das Erstellen des Servers um vieles einfacher und schneller machte.

Ebenfalls möchte ich Allen danken, die mir beim Testen der Web-App geholfen haben, da dies allein sehr schwierig gewesen wäre.

Auch meinen Eltern gilt ein grosser Dank, da Sie mich bei dieser Arbeit oftmals unterstützt haben.

Zuletzt danke ich Ihnen, dem Leser bzw. der Leserin für Ihr Interesse.

## 2. Zielsetzung

Meine Ziele für diese Maturaarbeit und für die Web-App waren dank vorsichtiger Planung bereits zu Beginn der Entwicklung klar. Ich habe diese Ziele in mehrere Kategorien eingeteilt.

- **Funktionalität:**  
Mir war klar, um eine gute Demonstration einer End-zu-End-Verschlüsselung zu erreichen, wird eine gute Verschlüsselung der Nachrichten nötig sein. Deshalb wollte ich von Anfang an mit Hybrid-Kryptographie arbeiten, da diese starke Sicherheit und hohe Geschwindigkeit liefert. Somit bin ich auf den Diffie-Hellman-Schlüsselaustausch gestossen, welcher meine Forderungen erfüllte.  
Ein weiteres Ziel war, dass mehrere Geräte in einem grossen Chat miteinander kommunizieren können, man aber auch verschlüsselte Nachrichten anderer sehen kann.  
Von Anfang an klar war jedoch auch, dass das Versenden von verschlüsselten Grafiken oder Audiodateien nicht möglich sein wird, da dies den Rahmen der Maturaarbeit gesprengt hätte.
- **Web-Applikation:**  
Damit ich eine Website oder eine App gerne benutze, muss die Grafische Oberfläche schön und benutzerfreundlich gestaltet sein. Da jede Person eine andere Wahrnehmung von Schönheit hat, wollte ich die fertige Web-App minimalistisch halten, damit sie einem grossen Teil der Benutzer:innen gefällt. Benutzerfreundlichkeit wollte ich durch selbsterklärende Zeichen erreichen, jedoch sollte es trotzdem eine zentrale Erklärung in einem Menu oder sondergleichen geben.
- **Server:**  
Wichtig ist, dass der Server zuverlässig und ohne grosse Vorbereitung jederzeit funktioniert. Das Ziel für diese Maturaarbeit ist ein lokaler Server, welcher nur die Kommunikation zwischen den Geräten übernimmt. In der Zukunft könnte dieser Server jedoch öffentlich gemacht werden, was Kommunikation zwischen verschiedenen Netzwerken ermöglichen würde.

Die oben genannten Ziele beschreiben das sogenannte *minimum viable product (MVP)*, zu Deutsch das *minimal brauchbare Produkt*, welches eine erste funktionsfähige Version eines Produkts darstellt. Diese Version wird als Basis für weitere Versionen verwendet, welche zum Beispiel zusätzliche Funktionalitäten einführen.

Auch für die Web-App dieser Maturaarbeit wurde ein weiterführendes Konzept entwickelt, mit Funktionalitäten, welche zukünftig noch hinzugefügt werden könnten. Für weitere zukünftige Pläne siehe Kapitel 7.2.

### 3. Theorie

Verschlüsselung bedeutet, Daten, zum Beispiel Text, mithilfe eines Schlüssels so zu verändern, dass der ursprüngliche Text ohne Kenntnis des Schlüssels nur sehr schwierig rekonstruiert werden kann.

Dazu gibt es verschiedene Methoden, manche nutzen Zahlen als Schlüssel, andere Buchstaben. Bei einigen Verschlüsselungsmethoden werden sogar mehrere Schlüssel gebraucht.

Die Verschlüsselung ist bei modernen Methoden oft ein sehr komplizierter, mathematischer Prozess.

Im folgenden Kapitel wird die Theorie zu klassischer Verschlüsselung, End-zu-End-Verschlüsselung und wichtigen mathematischen Grundlagen, Algorithmen sowie Protokolle erklärt.

#### 3.1. Klassische Verschlüsselung

Schon vor langem verwendete die Menschheit Methoden, um Nachrichten oder sogar ganze Schriften zu verschlüsseln, um sie vor Anderen - zum Beispiel Gegnern - geheim zu halten.

Dazu gab es verschiedene Herangehensweisen, von welchen im Folgenden zwei näher beschrieben werden.

##### 3.1.1. Caesar-Verschlüsselung

###### Allgemeines

Die Caesar-Verschlüsselung gehört zu den einfachsten klassischen Verschlüsselungsmethoden. Dabei werden alle Buchstaben in der Nachricht um einen gewissen Wert - den *key* - im Alphabet verschoben. Bei einem *key* von 4 würde aus dem Buchstaben 'A' ein 'E' werden, aus 'B' ein 'F' und so weiter. 'A' befindet sich an der ersten (1) Stelle im Alphabet. Rechnet man den Wert des *key*, welcher 4 ist dazu, erhält man die Position 5, welche für das 'E' steht.

Um die so verschlüsselten Nachrichten wieder zu entschlüsseln wird ebenfalls der *key* benötigt, dieser muss deshalb dem Empfänger bekannt sein.

###### Geschichte

Die Caesar-Verschlüsselung ist nach Julius Caesar benannt, welcher laut des römischen Schriftstellers *Sueton* seine militärischen Nachrichten mit Hilfe der oben beschriebenen Verschlüsselung verschickte.

Das Verfahren wurde im 15. Jahrhundert von *Leon Battista Alberti* durch die Erfindung der Chiffrierscheibe verbessert. Mit ihr konnte man die verschlüsselten Buchstaben einfach ablesen, indem man eine innere Scheibe gegenüber einer äusseren Scheibe um die Anzahl der verschobenen Buchstaben drehte.

*Trithemius* beschrieb 1508 mit Hilfe der von ihm erfundenen *Tabula recta* ein Verfahren, basierend auf der Caesar-Verschlüsselung, welche später als *Vigenère-Verschlüsselung* bekannt wurde.

###### Mathematik

Das Ganze lässt sich mathematisch mit Hilfe der Modulo-Addition (siehe Kapitel 3.3.1) beschreiben, wobei jedem der 26 Buchstaben im Alphabet zuerst ein Zahlenwert im Rahmen einer Gruppe zugeordnet werden muss, z.B. {a = 0; b = 1; c = 2; ...; z = 25}:

P = Klartext; C = Geheimtext; K = key;

Verschlüsselung:

$$C = (P + K) \pmod{26}$$

Entschlüsselung:

$$P = (C - K) \pmod{26}$$

(Caesar-Verschlüsselung, 2021)





3. Nun wird für jedes übereinanderstehende Buchstabenpaar in der Nachricht und im Schlüssel die "Kreuzung" im Vigenère-Quadrat gesucht und der daraus folgende Buchstabe notiert, was zur verschlüsselten Nachricht führt:

"HALLOMEINNA MEISTMATTIA"	Nachricht
"NJSDFWLKHGA NJSDFWLKHGA"	Schlüssel
"UJDOTIPSUTA ZNAVYILDAOA"	„Kreuzung“ im Vigenère-Quadrat

Verschlüsselte Nachricht: "UJDOTIPSUTAZNAVYILDAOA"

Oftmals wurde die verschlüsselte Nachricht noch in zufällig lange "Wörter" gegliedert:

"UJDOT IPSUTA ZNAVY ILDAOA"

Dies machte es schwieriger den Schlüssel herauszufinden.

4. Um die Nachricht als Empfänger zu entschlüsseln, muss man die verschlüsselte Nachricht und den Schlüssel übereinanderlegen und kann dann im Vigenère-Quadrat den dazugehörigen Klartext-Buchstaben suchen.

Dafür sucht man zuerst den Buchstaben im Schlüssel, sucht auf dessen Zeile den verschlüsselten Buchstaben, und liest dann zuoberst in dessen Spalte den Klartextbuchstaben ab.

Die Vigenère-Verschlüsselung ist sicherer als die Caesar-Verschlüsselung, da ein Buchstabe nicht immer gleich verschlüsselt wird, weil mehrere Buchstaben im Schlüssel zur Verfügung stehen.

(Vigenère-Chiffre, 2021)

### 3.2. End-zu-End-Verschlüsselung

In der modernen Kryptographie ist End-zu-End-Verschlüsselung der Überbegriff für sichere, verschlüsselte Kommunikation zwischen zwei oder mehreren Geräten, bei welcher der Inhalt der versendeten Nachrichten nur beim Sender und Empfänger in Klartext-Form vorliegt - überall sonst ist er komplett verschlüsselt und unlesbar.

Das bedeutet auch, dass die Nachricht nicht einmal vom Server, welcher die Kommunikation ermöglicht, entschlüsselt werden, da dies eine grosse Schwachstelle wäre.

(Computerphile, End to End Encryption (E2EE) - Computerphile [Video], 2017)

Die folgenden Unterkapitel zeigen moderne Methoden, wie eine solche End-zu-End-Verschlüsselung erzeugt werden kann.

#### 3.2.1. Symmetrische Kryptographie

Bei der symmetrischen Kryptographie wird derselbe Schlüssel (eine mathematisch erzeugte, sehr lange Zahl) zur Verschlüsselung sowie zur Entschlüsselung verwendet.

Dabei muss man sichergehen können, dass nur die Gesprächsteilnehmer den Schlüssel haben. Dieser muss deshalb vertraulich weitergegeben werden, was oftmals schwierig ist.

Beispiele für symmetrische Kryptographie sind die in Kapitel 3.1 genannten, klassischen Verschlüsselungsmethoden: Die Caesar-Verschlüsselung und die Vigenère-Verschlüsselung.

(internet-class, 2016)

### 3.2.2. Asymmetrische Kryptographie

Anders als bei der symmetrischen Kryptographie, wird bei der asymmetrischen Kryptographie nicht derselbe Schlüssel zur Verschlüsselung und Entschlüsselung verwendet.

Stattdessen wird ein mathematisch verbundenes Schlüsselpaar verwendet. Von diesem Schlüsselpaar ist ein Schlüssel öffentlich, der sogenannte *public key*. Der andere Schlüssel ist privat, dies ist der *private key*.

Dabei müssen folgende Bedingungen gelten:

- Der *public key* ist für alle Kommunikationspartner frei zugänglich, der *private key* muss geheim bleiben.
- Nur mit dem *private key* können Nachrichten, welche mit dem dazugehörigen *public key* verschlüsselt wurden, entschlüsselt werden.
- Nur mit dem *public key* können Nachrichten, welche mit dem dazugehörigen *private key* verschlüsselt wurden, entschlüsselt werden.
- Der *private key* kann nicht von dem *public key* abgeleitet werden.

Für beide Gesprächsteilnehmer wird nun ein solches Schlüsselpaar generiert.

Um eine Nachricht zu versenden, wird wie folgt vorgegangen:

1. Die Nachricht wird in ein verschlüsselbares Format konvertiert und mit dem eigenen *private key* verschlüsselt. Dadurch kann vom Empfänger sichergestellt werden, dass die Nachricht tatsächlich vom Sender versendet wurde, da nur er den *private key* besitzt.  
(Dieser Schritt ist nicht direkt für die Verschlüsselung notwendig, sondern dient nur der Authentifizierung des Senders.)
2. Die Nachricht wird mit dem *public key* des Empfängers verschlüsselt.  
(Dieser Schritt ist die eigentliche Verschlüsselung.)

Um die erhaltene Nachricht zu entschlüsseln, geht man folgendermassen vor:

1. Man entschlüsselt mit dem eigenen *private key* die Nachricht.  
(Dies macht Schritt 2 des Versendens rückgängig.)
2. Man entschlüsselt mit dem *public key* des Senders die Nachricht.  
(Dies macht Schritt 1 des Versendens, die Authentifizierung, rückgängig.)
3. Die Nachricht wird zurück zu Text konvertiert.

Die Verschlüsselung sowie die Schlüssel-Generation sind hierbei komplizierte, mathematische Prozesse.

Bei der Verschlüsselung handelt es sich um einen Algorithmus, welcher die Nachricht auf viele verschiedene Arten verändert und somit verschlüsselt, jedoch sicherstellt, dass die Nachricht immer noch entschlüsselt werden kann.

(Computerphile, Public Key Cryptography - Computerphile [Video], 2014)

### 3.2.3. Hybrid-Kryptographie

Bei der Hybrid-Kryptographie wird sowohl symmetrische als auch asymmetrische Kryptographie verwendet. Dies hat mehrere Vorteile, da symmetrische Kryptographie besser geeignet für grosse Datenmengen und schneller als asymmetrische Kryptographie ist, asymmetrische Kryptographie jedoch viel sicherer ist.

Hybrid-Kryptographie funktioniert in sogenannten *Sessions*. Eine *Session* (engl. Sitzung) kann unterschiedlich lange dauern. Im Falle der Nachrichtenverschlüsselung in einer Messaging-App dauert eine *Session* normalerweise vom Öffnen der App bis zum Schliessen der App.

Eine *Session* läuft wie folgt ab:

1. Asymmetrischer Schlüsselaustausch am Anfang der *Session*:
  - a. Beide Gesprächspartner generieren, wie in Kapitel 3.2.2 gezeigt, je ein asymmetrisches Schlüsselpaar.
  - b. Ein Gesprächspartner, der *Session Initiator*, generiert einen zufälligen, symmetrischen *session key*.
  - c. Der *session key* wird mit dem *public key* des Empfängers verschlüsselt und dem Empfänger geschickt.
  - d. Der Empfänger entschlüsselt den *session key* mit dem eigenen *private key*.  
Nun besitzen beide Gesprächsteilnehmer den *session key*.
  - e. Der Empfänger schickt dem *Session Initiator* eine mit dem *session key* verschlüsselte Bestätigung, dass der *session key* erhalten wurde.  
Wenn diese Nachricht von dem *Session Initiator* erfolgreich entschlüsselt werden kann, hat der Schlüsselaustausch funktioniert.
2. Nun werden Nachrichten mit schneller, symmetrischer Verschlüsselung durch den *session key* verschickt.
3. Am Ende der *Session* wird der verwendete *session key* verworfen.

(Skillset, 2016)

### 3.3. Mathematik, Algorithmen und Protokolle

Die folgenden Unterkapitel erklären die Modulo-Arithmetik, den *Square-and-Multiply*-Algorithmus und den Diffie-Hellman-Schlüsselaustausch (DHM-Protokoll).

Diese sind alle grundlegend im Gebiet der Kryptographie und ihr Verständnis ist wichtig.

#### 3.3.1. Modulo-Arithmetik

Bei der Modulo-Arithmetik geht es im Grunde um die Reste von Divisionen sowie um das Prinzip der *Kongruenz*.

Zwei Zahlen  $a$  und  $b$  sind *kongruent modulo  $n$* , wenn  $a / n$  und  $b / n$  denselben Rest ergeben.

Dies lässt sich vereinfacht so darstellen:

$$a \equiv b \pmod{n}$$

„ $\equiv$ “ = kongruent zu

Dies ist praktisch, da der Rest nie grösser sein kann als  $n$ . Somit können grosse Zahlen effizient berechnet werden, was Anwendung im *Square-and-Multiply*-Algorithmus findet.

Falls zwei Zahlen den gleichen Rest bei der Division mit  $n$  ergeben, kann man nur aus dem Rest nicht schliessen, worum es sich bei der ursprünglichen Zahl handelte, was verschiedene Verschlüsselungsmethoden ermöglicht.

(blackpenredpen, 2018)

### 3.3.2. Square-and-Multiply-Algorithmus

#### Prinzip

*Square-and-Multiply*, auch Binäre Exponentiation genannt, ist ein mathematischer Algorithmus, welcher effizientes Ausrechnen von natürlichen Potenzen, also Rechnungen in Form von  $x^k$  mit einem natürlichen Exponenten  $k$  ermöglicht. Natürlich ist eine Zahl, wenn sie zu den positiven ganzen Zahlen gehört.

Der Algorithmus wurde bereits in Indien um ca. 200 v. Chr. angewendet.

Das Prinzip des Algorithmus ist, die Anzahl benötigter Multiplikationen zu verringern, indem oft quadriert und danach multipliziert wird.

#### Binärsystem (Binärdarstellung)

Das Binärsystem ist, wie unser Dezimalsystem, ein Zahlensystem. Im Vergleich zum Dezimalsystem, welches die Basis 10 besitzt, rechnet man im Binärsystem mit der Basis 2. Das heisst, dass im Binärsystem nur zwei Zahlen zur Verfügung stehen: 0 und 1.

In dem uns gängigen Dezimalsystem bestehen Zahlen aus Zehnerpotenzen: Einer, Zehner, Hunderter, usw. Das Binärsystem verwendet Zweierpotenzen: Einer, Zweier, Vierer, Achter, usw.

#### Beispiel: Die Zahl 26

Im Dezimalsystem besteht die Zahl 26 aus 2 Zehner und 6 Einer.

Anders ausgedrückt heisst das:  $10 * 2 + 1 * 6 = 26$

Im Binärsystem besteht die Zahl 26 aus: 1 Sechszehner, 1 Achter, 0 Vierer, 1 Zweier und 0 Einer.

Anders ausgedrückt heisst das:  $16 * 1 + 8 * 1 + 4 * 0 + 2 * 1 + 1 * 0 = 11010$

Somit ist die Binärdarstellung der Zahl 26: 11010

Die Kenntnis der Binärdarstellung ist grundlegend für die Anwendung des *Square-and-Multiply*-Algorithmus.

#### Vorgang des Square-and-Multiply-Algorithmus zitiert nach Wikipedia (Stand Juli 2021)

- Umwandlung des Exponenten  $k$  in die zugehörige Binärdarstellung.
- Ersetzen jeder **0** durch **Q** und jeder **1** durch **QM**.
- Nun wird **Q** als Anweisung zum Quadrieren und **M** als Anweisung zum Multiplizieren aufgefasst.
- Somit bildet die resultierende Zeichenkette von links nach rechts gelesen eine Vorschrift zur Berechnung von  $x^k$ . Man beginne mit 1, quadriere für jedes gelesene **Q** das bisherige Zwischenergebnis und multipliziere es für jedes gelesene **M** mit  $x$ .

Da die Binärdarstellung von  $k > 0$  immer mit der Ziffer 1 beginnt – und so ebenfalls die Anweisung mit **QM** beginnt –, ergibt sich für die erste Anweisung **QM** in jedem Fall das Zwischenergebnis  $1^2 * x = x$ .

Aus diesem Grund ergibt sich eine leicht vereinfachte Variante, bei der die erste Anweisung **QM** durch  $x$  ersetzt wird.

Ende Zitat

#### Beispiel

```

k = 23 => k = 10111                                x^23 = x^23

=> QM Q QM QM QM   |   Erstes QM streichen (siehe oben)
=> Q QM QM QM

      Q      Q      M      Q      M      Q      M
x ----> x^2 ----> x^4 ----> x^5 ----> x^10 ----> x^11 ----> x^22 ----> x^23

```

### Binäre Modulo-Exponentiation

Falls man den *Square-and-Multiply*-Algorithmus in Kombination mit der Modulo-Arithmetik anwenden will, kann man eine einfache Modifikation am Algorithmus vornehmen, um zu verhindern, dass die berechneten Zahlen zu gross werden.

Dafür wird nach jedem Quadrieren und nach jedem Multiplizieren den Rest der Division des bisherigen Zwischenergebnisses mit dem Modulo gebildet. Dies ist dann das neue Zwischenergebnis.

Diese Methode wird sehr oft in der Kryptographie verwendet, zum Beispiel bei der asymmetrischen Verschlüsselung oder beim Diffie-Hellman-Schlüsselaustausch.

(Binäre Exponentiation, 2021)

### 3.3.3. Diffie-Hellman-Schlüsselaustausch (DHM-Protokoll)

#### Allgemeines

Der Diffie-Hellman-Schlüsselaustausch bzw. der **Diffie-Hellman-Merkle**-Schlüsselaustausch (kurz DHM-Protokoll) ermöglicht es, einen gemeinsamen geheimen Schlüssel über eine öffentliche, abhörbare Leitung zu vereinbaren.

Dieser Schlüssel kann im Anschluss für symmetrische Kryptographie verwendet werden.

Dies hat ähnliche Vorteile, wie die Hybrid-Kryptographie (siehe Kapitel 3.2.3) bezüglich Sicherheit und Geschwindigkeit. Darüber hinaus bietet das DHM-Protokoll zusätzliche Sicherheit, da es wie oben erwähnt auch in öffentlichen, abhörbaren Netzen funktioniert.

Die Hybrid-Kryptographie benötigt dagegen eine authentifizierte Leitung.

Das DHM-Protokoll wurde von Whitfield Diffie und Martin Hellman entwickelt. Veröffentlicht wurde es als erstes asymmetrisches Kryptoverfahren im Jahre 1976. Ralph Merkle hatte wichtige Vorarbeiten für das DHM-Protokoll geleistet, weshalb auch sein Name in der Abkürzung zu finden ist.

Der Diffie-Hellman-Schlüsselaustausch basiert auf der diskreten Exponentialfunktion, deren Umkehroperation der diskrete Logarithmus ist.

Die diskrete Exponentialfunktion ist in gewissen Fällen eine Einwegfunktion, was bedeutet, dass die Berechnung einer Exponentialrechnung aus verschiedenen Parametern mit Hilfe der diskreten Exponentialfunktion einfach und effizient ist.

Die Umkehroperation hingegen, das heisst das Berechnen der ursprünglichen Parameter (vor allem des Exponenten) mit Hilfe des diskreten Logarithmus, ist sehr schwierig und zeitintensiv.

$x \rightarrow a^x$	einfach und effizient zu berechnen
$a^x \rightarrow x$	schwierig und zeitintensiv zu berechnen

(Diffie-Hellman-Schlüsselaustausch, 2021)

#### Ablauf

Der Grundsatz des DHM-Protokolls ist es, öffentliche Schlüssel zu teilen, welche dann in Kombination mit den richtigen privaten Schlüsseln, bei beiden Teilnehmern den gleichen geheimen Schlüssel ergeben.

Am Anfang eines DHM-Schlüsselaustausches einigen sich die Gesprächsteilnehmer auf einige mathematische Parameter, welche verwendet werden:

- $g$ , der *generator*
- $n$ , eine sehr grosse Primzahl (z.B. 256-bit: ca. 80 Zeichen lang)

Diese Parameter werden ausgetauscht.

Danach wird wie folgt vorgegangen:

1. Unsere beiden Gesprächsteilnehmer, Alice und Bob, generieren mithilfe des Parameters  $n$  einen eigenen, zufälligen *private key*:  
 $a$  für Alice und  $b$  für Bob.

2. Aus beiden *private keys* werden mit Hilfe des *generators*  $g$  öffentliche *public keys* generiert:  
 $A$  für Alice und  $B$  für Bob.  
 Dies geschieht durch die diskrete Exponentialfunktion, was es praktisch unmöglich macht aus dem *public key* den ursprünglichen *private key* herauszufinden, da dafür der diskrete Logarithmus verwendet werden müsste (siehe vorherige Seite).  
 Die beiden *public keys* werden freigegeben und ausgetauscht.
3. Alice nimmt nun Bobs *public key* und fügt ihren *private key* hinzu. Bob tut das gleiche mit Alices *public key* und seinem eigenen *private key*.
4. Nun besitzen Alice und Bob je die gleiche "Mischung" aus  $a$ ,  $b$  und  $g$ , welche dem gemeinsamen geheimen Schlüssel, genannt  $s$  entspricht. Dieser kann nun für symmetrische Kryptographie verwendet werden.

(Computerphile, Secret Key Exchange (Diffie-Hellman) - Computerphile [Video], 2017)

### Mathematik

Mathematische Erklärung des Ablaufs für Interessierte.

1. Die privaten Schlüssel  $a$  und  $b$  sind zufällige Zahlen zwischen 1 und  $n$ .
2. Die öffentlichen Schlüssel  $A$  und  $B$  werden folgendermassen berechnet:
  - $A \equiv g^a \pmod{n}$        $\Leftrightarrow$        $A$  ist gleich dem Rest von  $\frac{g^a}{n}$
  - $B \equiv g^b \pmod{n}$        $\Leftrightarrow$        $B$  ist gleich dem Rest von  $\frac{g^b}{n}$
3. Das Hinzufügen des eigenen *private key* zu dem *public key* des Gesprächsteilnehmers, zur Berechnung des gemeinsamen geheimen Schlüssels  $s$  funktioniert folgendermassen:
  - Alice:  $s \equiv B^a \pmod{n} \equiv (g^b)^a \pmod{n}$        $\Leftrightarrow$        $s \equiv g^{ba} \pmod{n}$
  - Bob:  $s \equiv A^b \pmod{n} \equiv (g^a)^b \pmod{n}$        $\Leftrightarrow$        $s \equiv g^{ab} \pmod{n}$

Die gemeinsamen geheimen Schlüssel, welche Alice und Bob generieren sind identisch, da  $b * a = a * b$  und somit auch  $g^{ba} = g^{ab}$  gilt.

(Computerphile, Diffie Hellman -the Mathematics bit- Computerphile [Video], 2017)

### Man-in-the-Middle-Attack

Der DHM-Schlüsselaustausch bietet keinen Schutz vor einem "Spitzel" (nennen wir ihn Max), welcher sich in der "Mitte" des Austausches, d.h. im öffentlichen Bereich, befindet.

Max versucht, die Kommunikation zu verändern bzw. zu verhindern, indem er z.B. die Nachrichten abändert oder gar nicht erst weiterleitet.

Dies tut Max, indem er, gleich wie Alice und Bob, einen eigenen *private key* sowie einen *public key* generiert und diesen dann zum Beispiel mit Alice teilt und angibt, es wäre der *public key* von Bob. Dadurch kann Max, ohne dass Alice es weiss, einen gemeinsamen geheimen Schlüssel mit Alice erstellen.

Tut Max nun das Gleiche mit Bob und erstellt auch mit ihm einen gemeinsamen geheimen Schlüssel, so kann er nun die Nachrichten zwischen Alice und Bob empfangen, lesen, abspeichern, ggf. abändern und weiterleiten, ohne dass Alice und Bob etwas davon mitbekommen.

Um dieses Problem zu beheben, müssen sich die Gesprächspartner authentifizieren können.

Dafür wird der generierte *public key*, vor der Veröffentlichung, noch mit dem jeweiligen *private key* verschlüsselt. Da nur der ursprüngliche Ersteller des *private key* diesen hat bzw. haben sollte, kann der Gesprächspartner davon ausgehen, dass es sich um die richtige Person handelt.

(Computerphile, Key Exchange Problems - Computerphile [Video], 2017)

## Implementierung

Die folgenden Kapitel beschreiben die Implementierung der oben genannten Theorie in eine Web-Applikation (Kurz: Web-App) zur verschlüsselten Übertragung von Textnachrichten.

### 4. Material und Methoden

Die Web-App besteht aus zwei elementaren Teilen - der Server und die Website, welche von den Benutzerinnen aufgerufen wird. Deren Aufbau und Funktionsweise werden im Folgenden genauer beschrieben.

#### 4.1. (Web-) Server

Ein Server ist zuständig für den Datenaustausch zwischen *Clients* (siehe Kapitel 4.2) bzw. Benutzerinnen sowie zwischen den Clients und dem Server selbst. Zusätzlich wird er gebraucht, um lokal gespeicherte Dateien zu lesen und deren Inhalt an die Clients weiterzugeben.

Server, genauer Webserver, werden heutzutage überall verwendet. Jede öffentliche Website in Gebrauch läuft auf einem Webserver und wird teilweise von Benutzerinnen täglich sehr oft abgerufen. Server müssen deshalb auf leistungsfähigen Computer laufen.

Der in dieser Arbeit verwendete Webserver wurde in *Python* geschrieben und wird lokal auf dem PC einer Benutzerin *gehostet*. (Anleitung: siehe Kapitel B.1)

##### 4.1.1. Funktionsweise

Der Webserver erhält *Requests* von einem Client, auf welche er eine Antwort, die *Response*, gibt.

Die *Requests* können einen von folgenden vier Typen sein:

- **GET:** Der Client bittet um Daten, welche er nicht von der Benutzerin oder vom Web erhalten kann.  
**GET** wird in der Web-App verwendet, um die Nachrichten sowie die verfügbaren Benutzerinnen zu erhalten.
- **POST:** Der Client schickt dem Server neue Daten, welche vom Server verarbeitet werden. Mit **POST** werden neue Benutzerinnen von dem Server registriert und lokal abgespeichert. Daraufhin erhält der Client sein *uuid*. (siehe Kapitel 4.2)  
Ebenfalls wird **POST** zum Senden der Nachrichten verwendet, welche dann auf dem Server lokal gespeichert und weitergeleitet werden.
- **PUT:** Der Client möchte vom Server gespeicherte Daten ändern.  
**PUT** wird in der Web-App nicht verwendet.
- **DELETE:** Der Client möchte vom Server gespeicherte Daten löschen.  
**DELETE** wurde in der Entwicklung der Web-App zum Löschen aller Schlüssel verwendet, diese Funktion ist jedoch in der fertigen Web-App nicht verfügbar.

Die vom Server erhaltene Antwort kann nun beim Client auf der Website dargestellt werden.

##### 4.1.2. *api\_utils.py*

Der verwendete Webserver basiert auf dem Python-Framework *api\_utils.py*<sup>2</sup>, geschrieben von Marco Schmalz für verschiedene Web-Applikationen. Das Framework erlaubt das einfache Erstellen eines funktionstüchtigen Webserver, welcher auf einem beliebigen Netzwerk und *Port* läuft.

##### 4.1.3. *server.py*

Der Webserver wurde in der Datei *server.py* geschrieben und ist für die in Kapitel 4.1.1 beschriebenen Aufgaben zuständig. Zusätzlich bereitet er die lokalen Dateien zur Datenspeicherung vor.

---

<sup>2</sup> Version von Mattia Metzler erhältlich unter:

[https://github.com/MaGaMe19/Maturaarbeit-App/blob/master/api\\_utils.py](https://github.com/MaGaMe19/Maturaarbeit-App/blob/master/api_utils.py)



#### 4.1.4. Lokale Dateien

Die Web-Applikation benötigt zwei lokale, das heisst auf dem Computer gespeicherte, Dateien namens *data.json* und *users.json*, welche im gleichen Ordner wie die Datei *server.py* gespeichert werden und im JSON-Dateiformat geschrieben werden.

In *data.json* werden alle Nachrichten gespeichert.

In der Nachricht sind der Nachrichtentyp, die Senderin und die Empfängerin (als *uuid*) sowie der Inhalt der Nachricht enthalten.

```
{
  "type": "message",
  "from": "e0a3dbb1-12f6-46ab-aef2-d25162241019",
  "to": "d43d90cd-bfca-4332-8455-cf567ace626b",
  "content": {
    "0": 224,
    "1": 101,
    "2": 127
  }
}
```

Als Beispiel die linksstehende Nachricht.

Sie enthält verschlüsselt das Wort „Hoi“ und wurde von der Benutzerin mit dem *uuid* „e0a3dbb1...“ and die Benutzerin mit dem *uuid* „d43d90cd...“ gesendet.

Abbildung 2: Beispiel einer Nachricht (Screenshot aus *data.json*).

*users.json* wird verwendet, um alle Benutzerinnen zu speichern und sie über ihren *uuid* abrufen zu können. Dabei ist der *uuid* ausschlaggebend, d.h. zwei Benutzerinnen können zwar den gleichen Namen, nicht aber den gleichen *uuid* haben.

## 4.2. Client

Ein Client ist ein Benutzer einer Web-App oder Ähnliches, welcher auf einen Server zugreift. Im Falle dieser Web-App ist er einer der verfügbaren Benutzer, welche zurzeit aktiv sind.

Ein Client kann ein beliebiges Gerät sein, solange der verwendete Webbrowser *JavaScript* unterstützt und das Gerät mit dem gleichen Netzwerk wie der Webserver verknüpft ist.

Der Benutzer interagiert mit der Web-App über eine einzige Website, welche im gleichen Netzwerk wie der Webserver läuft.

Jeder Benutzer kann beim ersten Aufrufen der Web-App einen Namen eingeben, welcher fortan als Benutzername verwendet wird. Ebenfalls erhält der Benutzer dann sein *uuid*.

### 4.2.1. Vue.js

Die Website der Web-App ist - wie jede moderne Website - in *HTML*, *CSS* und *JavaScript* geschrieben.

Für letzteres wurde als Unterstützung *Vue.js*<sup>3</sup> verwendet, ein JavaScript-Framework, welches die Erstellung von Websites erleichtert und selbst viele Funktionen bietet, um Websites unter anderem interaktiv zu machen. Ebenfalls wird durch die Verwendung von *Vue.js* der Quellcode der Website um ein Vieles kürzer und verständlicher.

<sup>3</sup> Erhältlich unter: <https://vuejs.org/>

#### 4.2.2. Local Storage

Im *Local Storage* einer Website werden vom Webbrowser Daten gespeichert, welche wichtig für die Verwendung der Website sind. Dies ist besonders nützlich, wenn man zum Beispiel Einstellungen des Nutzers speichern möchte.

In der Web-App wird der *Local Storage* für folgende Daten verwendet:

- **Benutzername:** Der Benutzername wird gespeichert, um zu überprüfen, ob sich ein Benutzer bereits bei einem vorherigen Besuch der Website angemeldet hat. Ebenfalls wird er benötigt, um die Nachrichten korrekt darzustellen.
- **uuid:** Der *uuid* wird für die Identifizierung des Benutzers, sowie für das Versenden von Nachrichten verwendet.
- **Schlüssel:** Die zur Verschlüsselung benötigten Schlüssel werden ebenfalls im *Local Storage* gespeichert, jeder Schlüssel unter dem *uuid* des zugehörigen Benutzers. Zusätzlich wird der Status des DHM-Schlüsselaustausches (siehe Kapitel 3.3.3) gespeichert, da dies zur korrekten Darstellung der Benutzer benötigt wird.

Es wird empfohlen, diese im *Local Storage* gespeicherten Daten, **nicht** zu löschen, da dies zu Fehlfunktion der Web-App führt.

### 4.3. Kommunikation und Verschlüsselung

Die Kommunikation mit dem Server sowie die Verschlüsselung der Nachrichten werden beide vom Client übernommen und beruhen auf externen Programmen.

#### 4.3.1. Kommunikation - Axios

Für die Kommunikation bzw. den Datenaustausch mit dem Server wird Axios<sup>4</sup> verwendet. Axios ist ein HTTP-Client, welcher erlaubt, *Requests* (siehe Kapitel 4.1.1) an einen Server zu senden.

Axios ist *open-source* und wurde hauptsächlich von Matt Zabriskie entwickelt.<sup>5</sup>

Alternativ zu Axios könnte man andere HTTP-Request-Dienste wie z.B. XMLHttpRequest<sup>6</sup>, das von Microsoft entwickelt wurde und in fast jedem Webbrowser standardmässig unterstützt wird. Axios ist jedoch sehr viel einfacher handzuhaben und hat zusätzliche, praktische Funktionen.

#### 4.3.2. Verschlüsselung - jschacha20.js

Zur Verschlüsselung der Nachrichten dient *jschacha20.js* eine JavaScript-Implementierung von dem ChaCha20-Verschlüsselungsalgorithmus.

ChaCha20 ist ein symmetrischer Verschlüsselungsalgorithmus, welcher 2008 von Daniel J. Bernstein entwickelt wurde und auf Salsa20, ebenfalls von Daniel J. Bernstein entwickelt, basiert.

(Salsa20, 2021)

Die JavaScript-Implementierung wurde 2017 von Mykola Bubelich geschrieben und ist ebenfalls *open-source*.<sup>7</sup>

ChaCha20 ist einer der praktischsten und schnellsten symmetrischen Verschlüsselungsalgorithmen in JavaScript und somit gut für die Web-App geeignet.

Der genaue Verschlüsselungsvorgang ist sehr spannend, sprengt jedoch leider den Rahmen dieser Arbeit.

---

<sup>4</sup> Erhältlich unter: <https://axios-http.com/docs/intro> | <https://unpkg.com/axios> | <https://github.com/axios/axios>

<sup>5</sup> Alle Entwickler: <https://github.com/orgs/axios/people>

<sup>6</sup> Dokumentation: <https://developer.mozilla.org/de/docs/Web/API/XMLHttpRequest>

<sup>7</sup> Erhältlich unter: <https://github.com/thesimj/js-chacha20/blob/master/src/jschacha20.js>

## 5. Der Weg zur Web-Applikation

Die folgenden Unterkapitel zeigen, wie die fertige Web-Applikation Schritt für Schritt entstanden ist. Dies beinhaltet Prototypen der einzelnen Komponenten, sowie den gesamten Verlauf der Entwicklung.

### 5.1. Prototypen

Im Verlauf der Entwicklung der Web-App wurden mehrere Prototypen und Programme bzw. Websites zum Zwecke des *Proof of Concept* erstellt.

Alle diese Prototypen können auf GitHub<sup>8</sup> eingesehen und heruntergeladen werden.

#### 5.1.1. Verschlüsselung mit *Fernet* in Python

Mitte März 2021 entstand ein Prototyp der Verschlüsselung mithilfe von *Fernet* in Python. *Fernet* ist ein symmetrischer (siehe Kapitel 3.2.1) Verschlüsselungs-Algorithmus aus *pyca/cryptography*<sup>9</sup>, einer Erweiterung für Python, welche u.a. viele kryptographische Algorithmen enthält.

Beim Starten des Programmes wird um das Passwort gebeten. Danach kann man entweder eine Nachricht speichern oder eine Nachricht laden.

```
cmd: fernet_demonstration.py
Passwort: Mattia
Gewünschte Aktion? Nachricht [s]peichern / Nachricht [l]aden: s

Nachricht: Guten Tag
Name der Zieldatei (ohne Dateityp): wichtig
Die Nachricht wurde in der Datei 'wichtig.enc' gespeichert.

cmd: fernet_demonstration.py
Passwort: Mattia
Gewünschte Aktion? Nachricht [s]peichern / Nachricht [l]aden: l

Name der Ursprungsdatei (ohne Dateityp): wichtig

Entschlüsselte Nachricht:
Guten Tag

Nachricht wurde aus der Datei 'wichtig.enc' gelesen.

cmd: fernet_demonstration.py
Passwort: Baum
Gewünschte Aktion? Nachricht [s]peichern / Nachricht [l]aden: l

Name der Ursprungsdatei (ohne Dateityp): wichtig
Fehler: Dieses Passwort ist nicht korrekt um 'wichtig.enc' zu entschlüsseln.
```

Beim **Speichern** der Nachricht kann man den Inhalt der Nachricht sowie den Dateinamen eingeben.

Beim **Laden** der Nachricht gibt man den Namen der Datei ein, aus der die Nachricht gelesen werden soll. Stimmt das Passwort, so wird die Nachricht entschlüsselt.

Benutzt man ein **falsches Passwort**, erhält man eine Fehlermeldung und die Nachricht wird nicht entschlüsselt.

Abbildung 3: Screenshot des Prototyps in der Windows-Befehlszeile.

Die Nachrichten werden im Dateiformat *enc* (kurz für *encrypted*) im gleichen Ordner wie die Datei *fernet\_demonstration.py*, welche den Prototypen enthält, gespeichert. Die Nachrichten sind - ohne das korrekte Passwort - nicht zu entschlüsseln.

Diese Art der Verschlüsselung mit *Fernet* wurde in der fertigen Web-App nicht verwendet, da stattdessen *jschacha20.js* verwendet wurde.

<sup>8</sup> Erhältlich unter: <https://github.com/MaGaMe19/Maturaarbeit-Prototypen>

<sup>9</sup> Erhältlich unter: <https://cryptography.io/en/latest/>

### 5.1.2. User Interface

Anfangs Mai 2021 wurde ein Proof of Concept des *User Interface* (Abk. *UI*, engl. Benutzeroberfläche) erstellt. Das UI ist der Teil einer Applikation oder eines Programms, womit die Benutzerin interagiert.

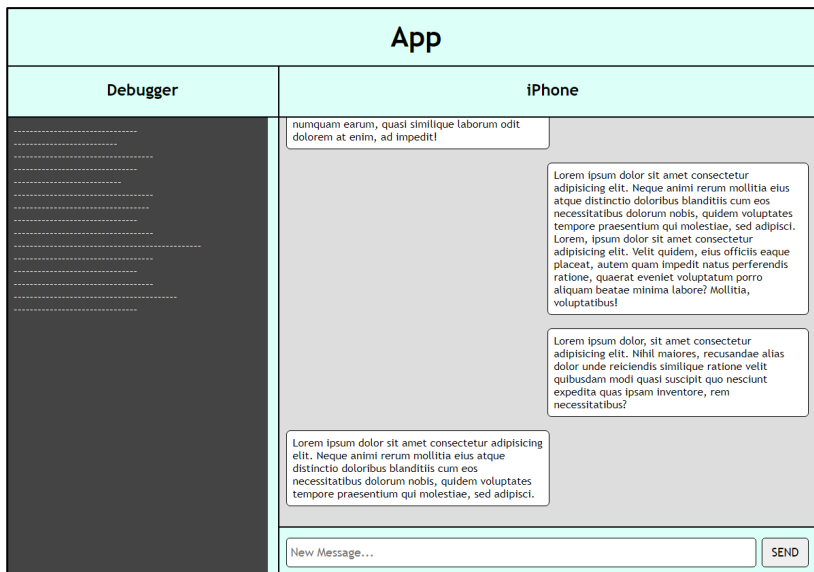


Abbildung 4: Screenshot des Proof of Concept des UI - jeglicher Text ist Blindtext.

Das Proof of Concept war inspiriert von anderen bekannten Messaging-Apps, wie z.B. WhatsApp oder Facebook Messenger.

Im linken Teil des UI befindet sich ein *Debugger*, welcher Zwischenschritte der Verschlüsselung sowie anderer Abläufe anzeigen sollte.

Im rechten Teil befindet sich der eigentliche Chat-Bereich, welcher mit einer Leiste für neue Nachrichten versehen ist.

Dieses Proof of Concept wurde erstellt, als noch nicht klar war, ob die fertige Web-App mehr als zwei Benutzerinnen erlauben wird. Da dies aber schlussendlich der Fall war, konnte dieses Proof of Concept nicht umgesetzt werden, da er nur auf zwei Benutzerinnen ausgelegt ist.

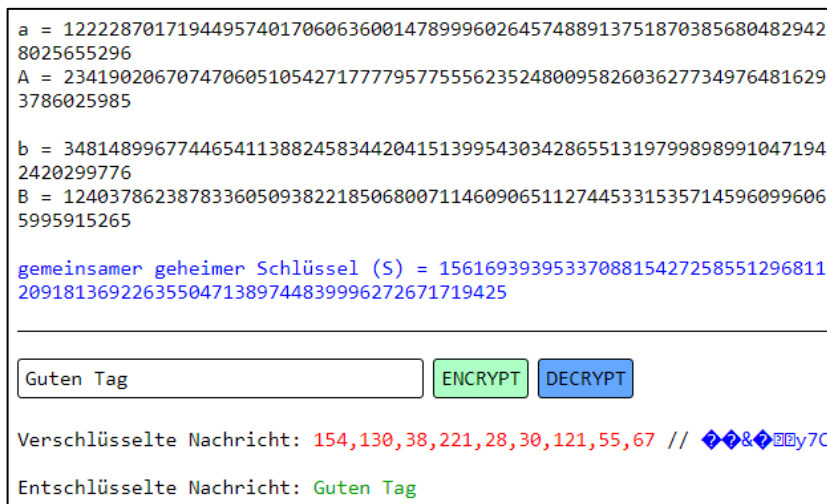
Funktionalität besitzt dieses Proof of Concept keine, es diente nur um herauszufinden, ob ein solches Design umsetzbar wäre. Dies hat sich jedoch durch die Anzahl Benutzerinnen von selbst geklärt, wie oben erwähnt. Die fertige Web-App hat zwar ein einfacheres, jedoch auch funktionelleres Design.

### 5.1.3. DHM-Schlüsselaustausch und Verschlüsselung mit *jschacha20.js*

Ebenfalls anfangs Mai 2021 wurde ein *Proof of Concept* des Diffie-Hellman-Schlüsselaustausches sowie der Verschlüsselung mit *jschacha20.js* erstellt.

Es besteht aus einer Website, welche beim Start zwei zufällige Schlüsselpaare und aus diesen dann den gemeinsamen geheimen Schlüssel generiert (siehe Kapitel 3.3.3).

Danach kann dieser Schlüssel verwendet werden um mit *jschacha20.js* (siehe Kapitel 4.3.2) Nachrichten zu ver- und entschlüsseln. Im Gegensatz zur fertigen Web-App, werden diese Nachrichten aber nirgends gespeichert und es ist auch kein Server involviert.



Im oberen Teil der Website findet der Schlüsselaustausch statt. Zu sehen sind die Schlüsselpaare  $\{a; A\}$  und  $\{b; B\}$  sowie  $s$  der gemeinsame geheime Schlüssel.

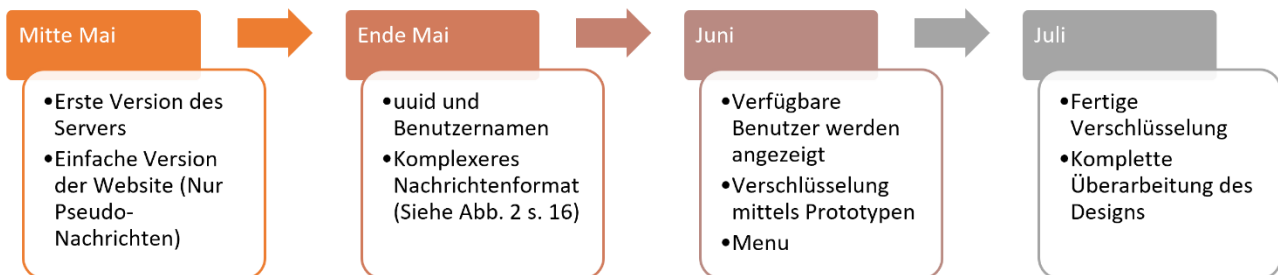
Im unteren Teil befindet sich die Verschlüsselung der Nachrichten mit dem oben generierten Schlüssel  $s$ . In diesem Beispiel wurde die Nachricht „Guten Tag“ ver- und entschlüsselt.

Abbildung 5: Screenshot des Proof of Concept des DHM-Schlüsselaustausches sowie einer verschlüsselten Nachricht.

Das Erstellen dieses *Proof of Concept* war ein wichtiger Schritt in der Entwicklung der Web-App, da es essenziell war sicherzugehen, dass der DHM-Schlüsselaustausch und die Verschlüsselung mit den gewählten Methoden umsetzbar ist, bevor der Rest der Web-App entwickelt wurde.

## 5.2. Zeitleiste

Die untenstehende Zeitleiste zeigt, welche Fortschritte wann erreicht wurden. Für genauere Informationen dienen die beiden folgenden Kapitel.



## 5.3. Back-End - *server.py*

Das *Back-End* einer Applikation bezeichnet die Schicht, in der Daten ausgetauscht werden. Dies umfasst oftmals den Server und alle ihn involvierende Prozesse.

Das Back-End dieser Web-App ist in Python in der Datei *server.py* geschrieben und wurde am 11. Mai 2021 erstellt. Zu dieser Zeit bestand die Datei nur aus wenigen Zeilen Code und war nur dazu gedacht, um etwas auszuprobieren und um das Konzept der *Requests* (siehe Kapitel 4.1.1) zu verstehen.

Im Verlauf der folgenden zwei Wochen wurden die ersten Funktionen hinzugefügt. Diese beinhalteten u.a. das Erstellen und Speichern von Pseudo-Nachrichten in einer lokalen Datei und die Möglichkeit, einen Benutzernamen zu wählen, welcher als „Absender“ verwendet wurde.

Gegen Ende Mai erhielt schliesslich jede Benutzerin bei der Anmeldung ein eigener *uuid*, ein sehr wichtiger Teil der Web-App. Alle Benutzerinnen werden zudem lokal abgespeichert. Ebenfalls entstand ein neues Format für die Nachrichten, welches demjenigen gezeigt in Abbildung 2 auf Seite 17 sehr ähnlich ist.

Anfangs Juni wurden die letzten relevanten Änderungen vorgenommen um die Verschlüsselung, welche im *Front-End* abläuft, im Server zu unterstützen.

Ebenfalls werden nun die beiden lokalen Dateien *data.json* und *users.json* beim Starten des Servers automatisch vorbereitet.

## 5.4. Front-End - *app.html* & *app-style.css*

Das *Front-End* einer Applikation bezeichnet die Schicht, auf welche die Benutzerin zugreift. Über das Front-End interagiert die Benutzerin mit der App sowie mit anderen Benutzerinnen. Es ist heutzutage fast immer grafisch gestaltet und besteht bei einer Web-App aus einer oder mehreren Websites.

Die Website dieser Web-App ist in HTML, CSS und JavaScript in den Dateien *app.html* und *app-style.css* geschrieben.

Sie wurde am 18. Mai 2021 erstellt und bestand dann nur aus der Datei *app.html*, da das Styling erst Mitte Juni in die separate Datei *app-style.css* verlegt wurde.

Zu dieser Zeit besass die Website fast keine Funktionalität, erst im Verlauf der folgenden Woche wurde die Unterstützung der damaligen Funktionen des Back-End hinzugefügt. Es konnten nun Pseudo-Nachrichten angezeigt, hinzugefügt und gelöscht werden.

Ebenfalls wurde dann die Website mit Hilfe von Vue.js (siehe Kapitel 4.2.1) umprogrammiert, was die Entwicklung um ein Vieles einfacher machte.

Zudem konnte nun ein Benutzername gewählt werden, welcher den Nachrichten vorabgesetzt wurde.

Da gegen Ende Mai jede Benutzerin vom Back-End ein *uuid* erhielt, musste dies auch von der Website unterstützt werden. Der *uuid* wird deshalb nun zusammen mit dem Benutzernamen im *Local Storage* (siehe Kapitel 4.2.2) gespeichert. Das neue Nachrichtenformat, welches auch zu dieser Zeit entstand, musste auch vom Front-End korrekt verwendet werden.

Eine grosse und wichtige Änderung war das Anzeigen der verfügbaren Benutzerinnen auf der Website, was anfangs Juni 2021 geschah. Man konnte nun die Empfängerin der Nachricht aus der Liste auswählen und dessen *uuid* wurde zu der Nachricht hinzugefügt.

Bis anhin wurden die Nachrichten vom Back-End in ihr endgültiges Aussehen auf der Website umgewandelt, nun findet dieser Prozess jedoch auf dem Front-End statt, was einfachere und schönere Darstellung erlaubt.

Der nächste sehr wichtige Schritt war die Verschlüsselung, deren Anfänge am 10. Juni implementiert wurden. Zu dieser Zeit konnte eine Benutzerin einer anderen Benutzerin eine zufällig generierte Zahl schicken, welche unter dem korrekten *uuid* im *Local Storage* abgespeichert wurde. Dies war ein Prototyp des Diffie-Hellman-Schlüsselaustausches, welcher Ende Juni erweitert wurde, um den Schlüsselaustausch mit zufälligen Zahlen nachzuahmen. Dazu mussten die abgespeicherten Schlüssel um einen Status erweitert werden, welcher aussagte, in welchem Schritt des Schlüsselaustausches man sich befand.

Am 30. Juni wurde nebst der Aufnahme von *jschacha20.js* der vollständige, korrekte Schlüsselaustausch fertiggestellt. Auch wurde die Liste der Benutzerinnen um Knöpfe erweitert, um den Schlüsselaustausch zu initialisieren oder um ihn zu bestätigen, falls er von einer anderen Benutzerin gestartet wurde.

In der Zwischenzeit wurden viele Styling-Änderungen vorgenommen sowie ein Menu implementiert, welches über die Web-App und deren Funktionen Auskunft gibt.

Die Verschlüsselung der Nachrichten wurde am 5. Juli abgeschlossen. Alle an einzelne Benutzerinnen gerichtete Nachrichten werden mit dem zugehörigen Schlüssel verschlüsselt und bei der Empfängerin wieder sicher entschlüsselt. Nachrichten, welche nicht an die die Website verwendende Benutzerin gerichtet sind, können nicht entschlüsselt werden und bleiben somit verschlüsselt (siehe Abbildung 6 rechts unten).

Dies umfasst alle Funktionen des fertigen Front-End.

Danach wurde Ende Juli das Design der Website komplett überarbeitet und ist nun viel übersichtlicher und schöner gestaltet. Ebenfalls passt sich die Website an fast alle Bildschirmgrössen an. Nur auf einem Smartphone ist das Verwenden der Website nicht sehr praktisch.

Abschliessend ein Vergleich zwischen einem früheren Punkt der Entwicklung und der fertigen Website:

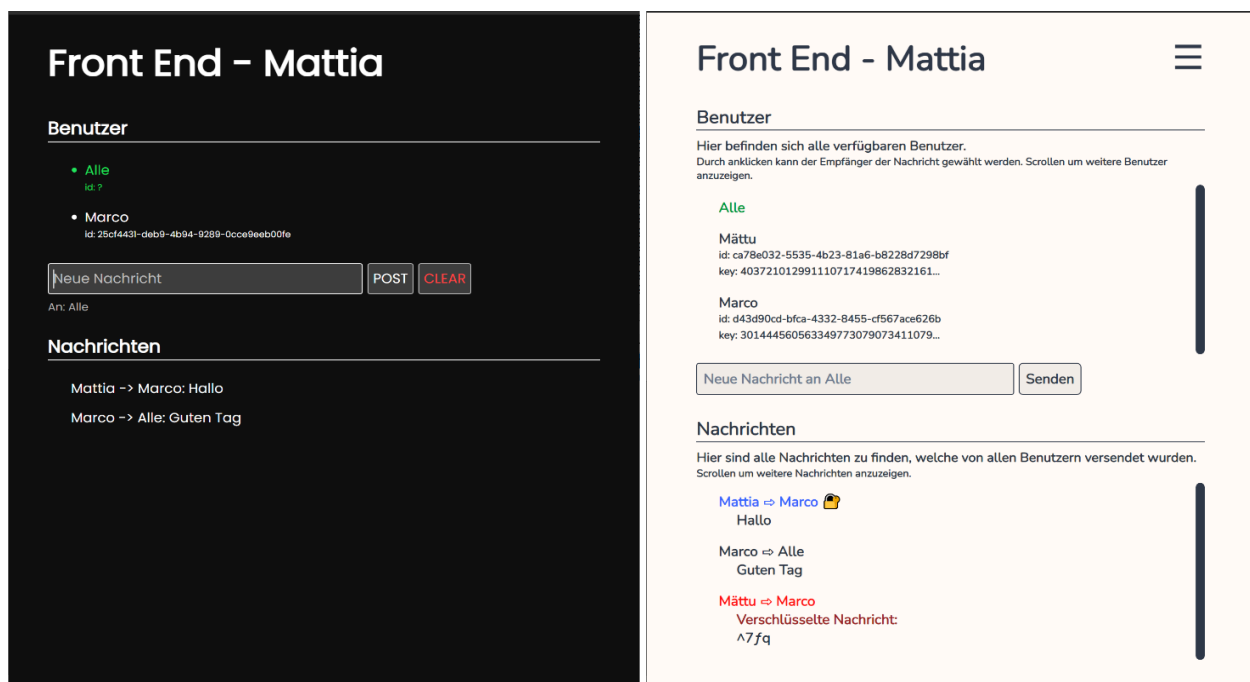


Abbildung 6: Vergleich der Web-App zwischen der Version des 10. Juni 2021 (links) und derjenigen des 1. August 2021 (rechts). Beides Screenshots



## 6. Endprodukt und Testing

Die fertige Messaging-Web-App sieht wie folgt aus (zukünftige Änderungen sind vorbehalten):

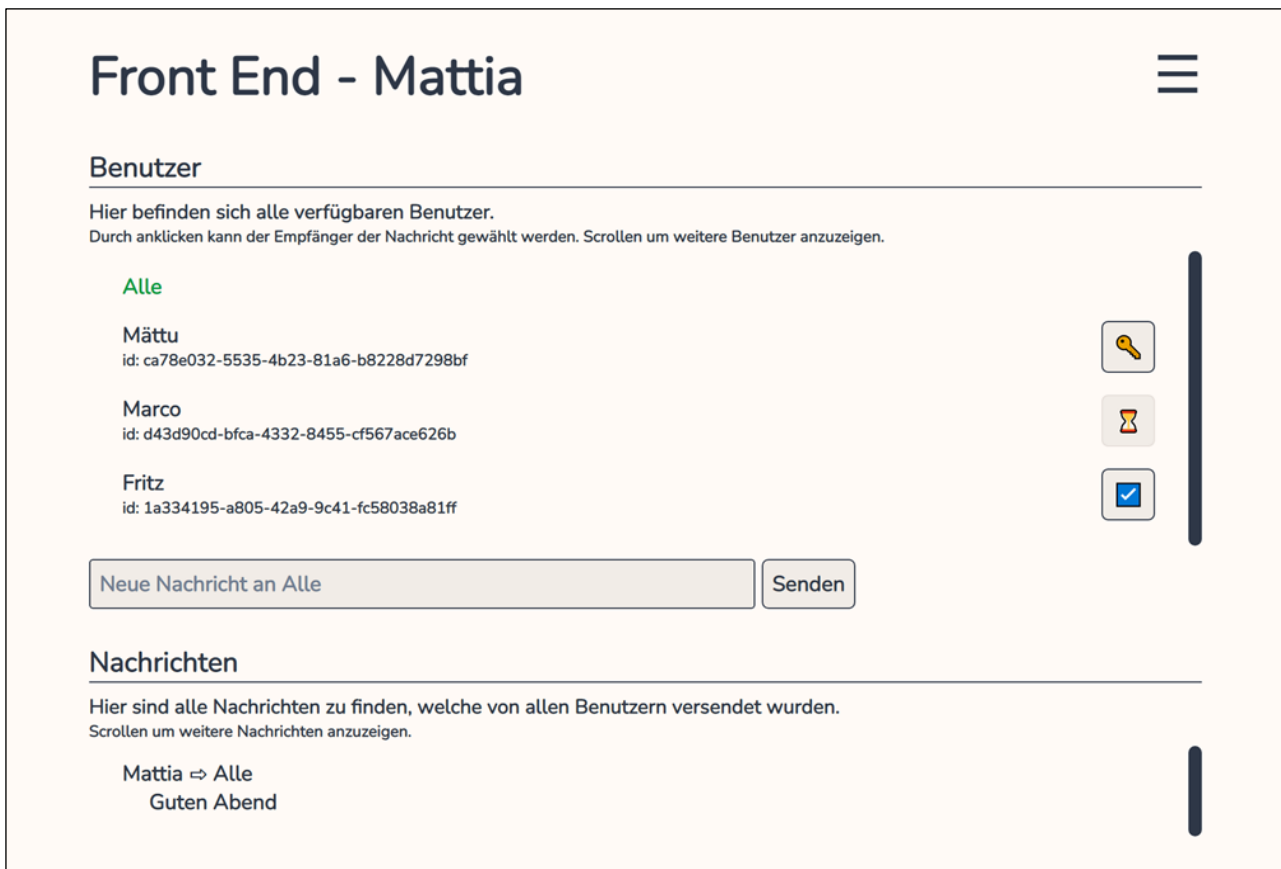


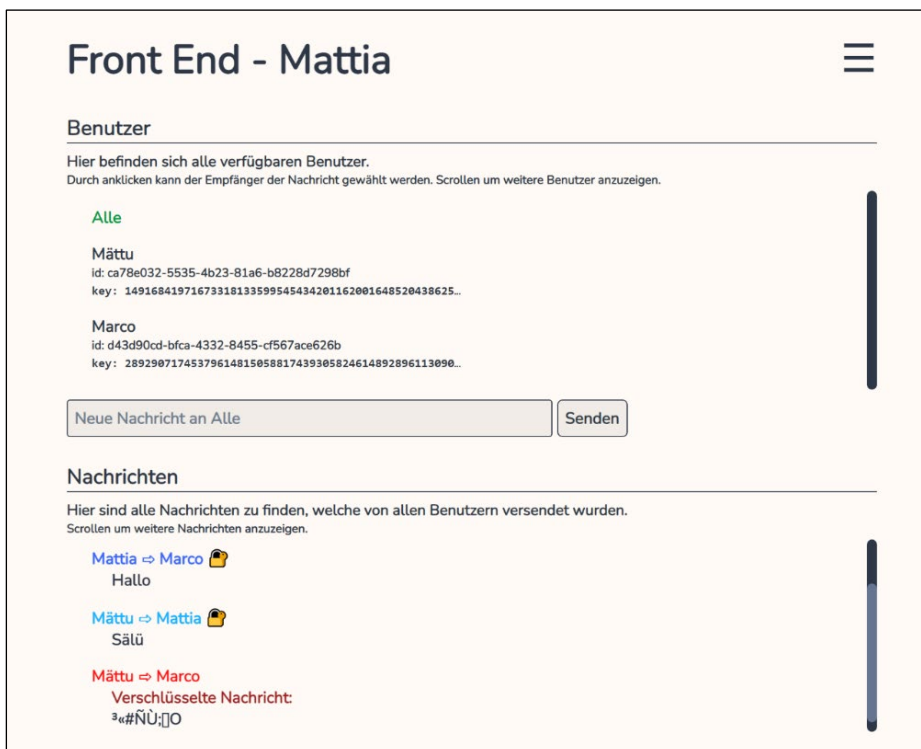
Abbildung 7: Die fertige Web-App (Stand 03. August 2021) mit den drei verschiedenen Status des DHM-Schlüsselaustausches. (Screenshot)

Zuoberst befindet sich der Name des aktuellen Benutzers. Darunter findet man die Liste der verfügbaren Benutzer, aus welchen man den gewünschten Empfänger auswählen kann. Dieser wird dann in grün hervorgehoben. Rechts neben jedem Benutzer befindet sich der Status des Schlüsselaustausches mit dem jeweiligen Benutzer. Die Zeichen bedeuten von oben nach unten folgendes:

- **Schlüssel - Schlüsselaustausch initialisieren (Standard)**  
Der Schlüsselaustausch wurde noch nicht initialisiert, somit können noch keine Nachrichten versendet werden. Durch Klicken auf dieses Symbol wird der Schlüsselaustausch initialisiert und die **Sanduhr** wird angezeigt.
- **Sanduhr - Antwort wird abgewartet**  
Der Schlüsselaustausch wurde, von dem dieses Symbol sehende, Benutzer initialisiert. Nun wird die Antwort des angefragten Benutzers abgewartet, der Nachrichtenaustausch ist solange nicht möglich. Bei dem angefragten Benutzer wird das **Kontrollkästchen** angezeigt.
- **Kontrollkästchen - Schlüsselaustausch bestätigen**  
Der Schlüsselaustausch wurde von dem Benutzer links neben diesem Symbol initialisiert. Der Benutzer, der dieses Symbol sieht, kann durch Anklicken den Schlüsselaustausch bestätigen. Nun wird kein Symbol mehr angezeigt, dafür wird unterhalb der *uuid* des Benutzers der zugehörige Schlüssel (*key*) abgedruckt. Nun ist verschlüsselter Nachrichtenaustausch möglich.

Unterhalb der Liste der Benutzer befindet sich das Eingabefeld zum Versenden einer neuen Nachricht. Durch Drücken der Eingabe-Taste oder durch drücken auf den Knopf „Senden“ wird die Nachricht gesendet.


Nach dem Schlüsselaustausch und mit einigen verschickten Nachrichten, sieht die Web-App folgendermassen aus:



Wie auf der vorherigen Seite erwähnt, werden nun die zugehörigen Schlüssel unterhalb des *uuid* der jeweiligen Benutzer angezeigt. Diese Schlüssel sind die, durch den DHM-Schlüsselaustausch generierten, gemeinsamen geheimen Schlüssel. Aus ästhetischen Gründen werden sie nicht in voller Länge angezeigt, durch darüberfahren mit der Maus, wird jedoch der ganze Schlüssel sichtbar.

Abbildung 8: Die fertige Web-App (Stand 03. August 2021) mit einigen verschickten Nachrichten. (Screenshot)

Im unteren Teil der Web-App findet man alle Nachrichten, welche versendet wurden. Diese haben verschiedene Farben und Symbole, abhängig von dem Sender und Empfänger der Nachricht:

- **Dunkelblau** Diese Nachricht wurde von Ihnen gesendet.
- **Hellblau** Diese Nachricht wurde an Sie gesendet.
- **Rot** Diese Nachricht ist nicht lesbar (verschlüsselt), da sie nicht an Sie gesendet wurde.
-  Diese Nachricht wurde sicher verschlüsselt. Hat eine Nachricht dieses Symbol nicht, so wurde sie unverschlüsselt an Alle gesendet.

Wenn die unterste, das heisst die neueste Nachricht an den betrachtenden Benutzer gesendet wurde, wird oben auf der Website neben dem Namen ein grüner Punkt angezeigt. Durch Klicken auf diesen Punkt wird die Nachrichtenliste automatisch nach unten gescrollt. Dies dient als eine Art Benachrichtigung.

Durch Drücken der Tastenkombination `shift + ctrl + i` können in den Browsern Google Chrome, Firefox und Microsoft Edge die Entwickler-Tools angezeigt werden. Im Abschnitt *Konsole* werden bei vielen verschiedenen Aktionen in der Web-App Informationen über diese Aktion angezeigt. Diese Informationen sind in **blau** abgedruckt.

## 6.1. Testing

Getestet wurde die Web-App mehrere Male in verschiedenen Netzwerken, u.a. ein mobiler Hotspot. In all diesen Netzwerken funktionierte die Web-App einwandfrei.

Getestete Webbrowser sind Google Chrome, Mozilla Firefox, Microsoft Edge und Safari (MacOS). Die Website funktioniert auf all diesen Webbrowsern und wurde teilweise für sie optimiert.

Ebenfalls getestet wurden verschiedene Betriebssysteme. Diese sind Windows 10, MacOS, iOS und iPad OS. Das Benutzen der Website auf Smartphones wird Stand 01.10.2021 nicht empfohlen, da die Website (noch) nicht für diese Geräte optimiert ist. Das *Hosten* des Webserver wurde nur mit Windows 10 getestet.



## 7. Evaluation und Bemerkungen

### 7.1. Erreichte Ziele

Die Ziele des *Minimum Viable Product* der Web-App, welche in Kapitel 2 genannt wurden, wurden allesamt erreicht.

- **Funktionalität:**  
Die Web-App verwendet Hybrid-Kryptographie auf Basis eines Diffie-Hellman-Schlüsselaustausches, welche sehr sichere Verschlüsselung mit hoher Geschwindigkeit bietet.  
Es können viele verschiedene Browser auf verschiedenen Geräten miteinander in einem grossen Chat kommunizieren, Bilder und Audiodateien kann man jedoch nicht versenden.
- **Web-Applikation:**  
Die Web-App ist eher minimalistisch und in meiner Meinung schön gestaltet, mit relativ selbsterklärenden Zeichen und Farben. Falls etwas unklar ist, kann jedoch die Bedeutung im Menu (erreichbar über den Knopf oben rechts, siehe Abbildung 8, Seite 24) nachgelesen werden.
- **Server:**  
Der Server ist einfach zu starten und falls nötig anzupassen. Die Website wird lokal über Visual Studio Code gehostet (Anleitung siehe Kapitel B.1), was wenige Sekunden dauert und die Website wird automatisch aktualisiert.

### 7.2. Zukunft der Web-App

Wie in der Zielsetzung in Kapitel 2 erwähnt, bestehen einige Pläne für die Zukunft der Web-App, wie zum Beispiel ein öffentlicher Server anstelle eines lokalen Servers. Dies würde ermöglichen, dass Benutzer von der ganzen Welt miteinander kommunizieren könnten. Dafür müsste der Server jedoch etwas effizienter sein und auf guter Hardware laufen. Möglich wäre das durch einen gemieteten Server z.B. von Google<sup>10</sup> oder Amazon<sup>11</sup>. Jedoch müssten dann Anpassungen an der Sicherheit der Web-App vorgenommen werden, unter anderem durch ein besseres Anmeldungssystem, mit Benutzernamen und Passwort.

In zukünftigen Versionen der Web-App könnten Bilder und Audio-Dateien verschickt werden, die Benutzeroberfläche könnte anpassbar sein und es könnten Push-Nachrichten auf dem Gerät angezeigt werden, wenn man eine neue Nachricht erhält. Ebenfalls sollte die Web-App für Smartphones optimiert werden.

Des Weiteren könnte eine Ansicht oder ein Fenster hinzugefügt werden, welches live anzeigt, was gerade im Hintergrund der Web-App oder auf dem Server passiert, anstatt dass dies nur in der Konsole des Browsers zu sehen ist.

Eine tolle zukünftige Anwendung der Web-App wäre als Mittel der Kommunikation im Informatik-Unterricht am Gymnasium Neufeld, eine Idee von Herrn Schmalz.

### 7.3. Bemerkungen

Geplant war ursprünglich noch die Implementation einer Man-in-the-Middle-Attack (siehe Kapitel 3.3.3, Seite 15). Da dies jedoch nichts direkt mit dem eigentlichen Ziel der Maturaarbeit, nämlich einer Implementation von End-zu-End-Verschlüsselung zu tun hat, belies ich dieses Thema bei einer theoretischen Erklärung.

Ebenfalls konnte die Web-App nicht im zeitlichen Rahmen der Arbeit für Smartphones optimiert werden, da dies ein komplett neues Design erfordert hätte.

---

<sup>10</sup> Mehr Informationen: <https://cloud.google.com/>

<sup>11</sup> Mehr Informationen: <https://aws.amazon.com/>

## 8. Schlusswort

Dank sorgfältiger Planung und frühzeitiger Entwicklung konnten alle Ziele des *MVP* für die Web-App in der zur Verfügung stehenden Zeit erreicht werden. Es gibt aber immer Luft nach oben und es könnten noch viel mehr Funktionen hinzugefügt werden, wie im Kapitel 7.2 beschrieben.

Leider wurde während der Entwicklung teilweise Zeit verschwendet mit Funktionen oder Prototypen, welche nie verwendet wurden und/oder nicht viel zu der eigentlichen Web-App beigetragen haben.

Deshalb müsste man sich bei Wiederholung einer solchen Arbeit mehr auf das Wesentliche fokussieren, was durch noch klarere Ziele erreicht werden könnte.

Trotzdem ist das Endprodukt funktionstüchtig und zufriedenstellend.

Dank wöchentlichen Besprechungen über den Fortschritt der Arbeit mit Herrn Schmalz wurde sichergestellt, dass sich die Entwicklung im Grossen und Ganzen in die richtige Richtung bewegte. Durch gute Inputs von Herrn Schmalz wurde die Entwicklung oftmals um vieles verschnellert.

Auch das Ziel, mithilfe dieser Arbeit und der entwickelten Web-App die End-zu-End-Verschlüsselung relativ einfach zu erklären und darzustellen wurde meiner Meinung nach erreicht.

Für das Verfassen der schriftlichen Arbeit hätte von Anfang an etwas mehr Zeit eingeplant werden können, da dies mehr Aufwand beinhaltet als ursprünglich geplant.

Das Verfassen wurde jedoch dadurch erleichtert, dass zu Beginn der Arbeit bereits sehr viel recherchiert wurde, was am Ende Zeit beim Verfassen des Theorie-Teils sparte.

Ich habe durch diese Arbeit sehr viel neues gelernt und mein Wissen in vielen Gebieten vertieft. Nicht nur im Gebiet der Kryptographie, sondern generell beim Programmieren sehe ich nun eine Aufgabe oder Herausforderung mit anderen Einstellungen und kann so effizienter arbeiten.

Ich hatte viel Spass und Freude am Entwickeln der Web-App, was zu einem grossen Teil daraus resultiert, dass ich schon immer sehr interessiert an der Informatik war. Ich wurde hiermit erneut bestätigt, dass es der richtige Entscheid war, ein Gebiet der Informatik als Thema der Maturaarbeit zu wählen.

Um auf das Beispiel von WhatsApp aus der Einleitung zurückzukommen:

Nach tieferen Nachforschungen bin ich auf ein *White Paper* von WhatsApp gestossen, welches die Verschlüsselung sehr detailliert erklärt. WhatsApp benutzt ebenfalls Hybrid-Kryptographie, jedoch mit viel mehr Schlüsseln mit unterschiedlicher Lebensdauer. Für fast jede Nachricht wird ein neuer, einmaliger Schlüssel generiert. Somit können ungewollt veröffentlichte Schlüssel praktisch keine Nachrichten entschlüsseln. Als Verschlüsselungsalgorithmus verwendet WhatsApp AES (**A**dvanced **E**ncryption **S**tandard), einen weitverbreiteten, symmetrischen Verschlüsselungsalgorithmus, welcher mittlerweile standardisiert wurde. Also kann man abschliessend mit gutem Gewissen sagen, dass die Verschlüsselung von WhatsApp einen sehr sicheren Nachrichtenaustausch ermöglicht. (WA\_Security\_WhitePaper, 2020)

Ich hoffe, dass Sie fortan, wenn Sie die Meldung aus Abbildung 1 auf Seite 6 sehen, wissen worum es sich tatsächlich handelt.

*“A little bit of math can accomplish what all the guns and barbed wire can't:  
a little bit of math can keep a secret.” - E. Snowden, 2019*

## 9. Links, Quellen und Literatur

### Links

GitHub von Mattia Metzler <https://github.com/MaGaMe19>

YouTube-Videos

Playlist von Mattia Metzler:

<https://youtube.com/playlist?list=PLXK3JSMMVB4PPL3VAo5fNIRHX9w-KtdIP>



### Quellen- und Literaturverzeichnis

*Binäre Exponentiation*. (14. April 2021). Abgerufen am 22. Juli 2021 von Wikipedia:

[https://de.wikipedia.org/w/index.php?title=Bin%C3%A4re\\_Exponentiation&oldid=210920726](https://de.wikipedia.org/w/index.php?title=Bin%C3%A4re_Exponentiation&oldid=210920726)

blackpenredpen. (23. April 2018). What does  $a \equiv b \pmod{n}$  mean? Basic Modular Arithmetic, Congruence [Video]. Von <https://www.youtube.com/watch?v=6dZLq77gSGU> abgerufen

*Caesar-Verschlüsselung*. (22. September 2021). Abgerufen am 20. Juli 2021 von Wikipedia:

<https://de.wikipedia.org/w/index.php?title=Caesar-Verschl%C3%BCsslung&oldid=215808052>

Computerphile. (22. Juli 2014). Public Key Cryptography - Computerphile [Video]. Von

[https://www.youtube.com/watch?v=GSIDS\\_lvRv4](https://www.youtube.com/watch?v=GSIDS_lvRv4) abgerufen

Computerphile. (20. Dezember 2017). Diffie Hellman -the Mathematics bit- Computerphile [Video]. Von

[https://www.youtube.com/watch?v=Yjrfm\\_oRO0w](https://www.youtube.com/watch?v=Yjrfm_oRO0w) abgerufen

Computerphile. (30. März 2017). End to End Encryption (E2EE) - Computerphile [Video]. Von

<https://www.youtube.com/watch?v=jkV1KEJGKRA> abgerufen

Computerphile. (29. Dezember 2017). Key Exchange Problems - Computerphile [Video]. Von

<https://www.youtube.com/watch?v=vsXMMT2CqqE> abgerufen

Computerphile. (15. Dezember 2017). Secret Key Exchange (Diffie-Hellman) - Computerphile [Video]. Von

<https://www.youtube.com/watch?v=NmM9HA2MQGI> abgerufen

*Diffie-Hellman-Schlüsselaustausch*. (15. Juli 2021). Abgerufen am 23. Juli 2021 von Wikipedia:

<https://de.wikipedia.org/w/index.php?title=Diffie-Hellman-Schl%C3%BCsselaustausch&oldid=213890443>

internet-class. (17. Oktober 2016). What is symmetric encryption? [Video]. Von

<https://www.youtube.com/watch?v=8Ov4HyncJU0> abgerufen

*Salsa20*. (14. Juli 2021). Abgerufen am 19. Juli 2021 von Wikipedia:

<https://en.wikipedia.org/w/index.php?title=Salsa20&oldid=1033504897>

Skillset. (02. Mai 2016). Hybrid Cryptography (CISSP Free by Skillset.com) [Video]. Von

[https://www.youtube.com/watch?v=VPvZbMXfv\\_0](https://www.youtube.com/watch?v=VPvZbMXfv_0) abgerufen

Snowden, E. (2019). *Permanent Record*. Metropolitan Books.

*Vigenère-Chiffre*. (31. Mai 2021). Abgerufen am 20. Juli 2021 von Wikipedia:

<https://de.wikipedia.org/w/index.php?title=Vigen%C3%A8re-Chiffre&oldid=212550131>

*WA\_Security\_WhitePaper*. (22. Oktober 2020). Abgerufen am 19. Oktober 2021 von WhatsApp:

[https://scontent.whatsapp.net/v/t39.8562-34/122249142\\_469857720642275\\_2152527586907531259\\_n.pdf/WA\\_Security\\_WhitePaper.pdf?ccb=1-5&\\_nc\\_sid=2fbf2a&\\_nc\\_ohc=CmpnWJSJpMQAX8eqZSb&\\_nc\\_ht=scontent.whatsapp.net&oh=b38d9aade576c94976253d65e1f2a888&oe=61732419](https://scontent.whatsapp.net/v/t39.8562-34/122249142_469857720642275_2152527586907531259_n.pdf/WA_Security_WhitePaper.pdf?ccb=1-5&_nc_sid=2fbf2a&_nc_ohc=CmpnWJSJpMQAX8eqZSb&_nc_ht=scontent.whatsapp.net&oh=b38d9aade576c94976253d65e1f2a888&oe=61732419)

## A. Glossar

Alle im Text *schräggeschriebenen* Wörter sind hier zu finden.

Ausnahmen:

- Dateinamen
- Eigennamen
- Wörter, welche direkt im Text erklärt werden.
- Wörter, welche in Fremdsprachen geschrieben sind.

**Client** Ein Client ist ein Benutzer einer Web-App oder Ähnliches, welcher auf einen Server zugreift.

**CSS** CSS (**C**ascading **S**tyle **S**heets) ist eine Programmier- bzw. Stylesheet-Sprache, welche die Darstellung von Dokumenten, zum Beispiel Websites, vorgibt. CSS ist sehr einfach für Menschen zu lesen und besteht immer aus einer Eigenschaft und deren Wert.

```
body {  
    background-color: red;  
    color: blue;  
}
```

Der obenstehende Code würde zum Beispiel den Hintergrund der gesamten Website (*body*) rot färben und den Text blau färben.

**Debugger** Ein Debugger erlaubt es beim Programmieren, ein Programm Schritt für Schritt durchzugehen, um so etwaige Fehler zu finden.

**Framework** Frameworks sind Programmgerüste, welche selbst keine fertigen, ausführbaren Programme sind, jedoch den Rahmen für eine Anwendung o.ä. liefern.

**hosten** Ein Server kann auf einem Computer gehostet werden und anschliessend zur Kommunikation mit anderen Computern verwendet werden. Ebenfalls kann eine Datei, meistens eine Website, auf einem Server gehostet werden.

**HTML** HTML (**H**yper**T**ext **M**arkup **L**anguage) ist eine weit verbreitete Markup-Sprache, d.h. eine Textformat, welches präsentiert anders aussieht als das Textdokument selbst. HTML wird grösstenteils für Websites verwendet und kann von CSS und/oder JavaScript unterstützt werden. HTML verwendet sogenannte Elemente, welche unterschiedliche Eigenschaften haben und einen Inhalt haben können.

```
<html>  
<head>  
    <title>Website</title>  
</head>  
<body>  
    <p>Guten Tag</p>  
</body>  
</html>
```

Der obenstehende Code erstellt eine Website mit dem Titel „Website“ und dem Text „Guten Tag“. Dies kann mit dem vorherigen CSS kombiniert werden, wodurch die Website blauen Text auf rotem Hintergrund enthalten würde.

**JavaScript**

JavaScript (Abk. JS) ist eine Programmiersprache, welche von sehr vielen Websites genutzt wird. Sie dient u.a. zur Interaktivität oder erlaubt zum Beispiel dynamische Websites, welche sich an die Benutzer anpassen.

```
function changeColor() {  
    var body = document.body;  
    body.style.color = "green";  
}
```

Der obenstehende Code enthält eine Funktion, um die Farbe des Textes auf der Website nach grün zu ändern. Der Code in einer Funktion kann durch Aufrufen der Funktion mehrmals verwendet werden, ohne dass er erneut geschrieben werden muss.

**JSON**

JSON (JavaScript Object Notation) ist ein Dateiformat, welches einfach für Menschen zu lesen ist und von sehr vielen Computern und Programmiersprachen verstanden wird. Sein Aufbau ist ähnlich zu dem eines Lexikon bzw. Nachschlagewerk; Werte werden unter Begriffen abgespeichert und können wieder abgerufen werden.

```
{  
    "name": "Max Mustermann",  
    "alter": 42,  
    "hobbies": ["Tennis", "Klavier"]  
}
```

Das obenstehende JSON-Dokument enthält Informationen über eine Person. Im letzten Eintrag *hobbies* wird eine Liste verwendet.

**open-source**

Ist ein Programm oder Dienst open-source, so ist der Quellcode der Öffentlichkeit zugänglich. Oftmals kann auch jeder dazu beitragen.

**Port**

Die Ports eines Netzwerks können mit verschiedenen Personen in einem Haus verglichen werden - weiss man zwar die Adresse des Hauses, muss man immer noch spezifizieren, von welcher Person man spricht. Ähnlich können verschiedene Ports verwendet werden, um unterschiedliche Websites oder Dienste auf einem Server laufen zu lassen.

In diesem Projekt wurde der Port 3000 für den Webserver und der Port 5500 für die Website verwendet.

**Proof of Concept**

Etwas, das als Proof of Concept dient, wird verwendet, um zu zeigen, dass eine Idee umsetzbar ist bzw. funktioniert. Oftmals werden einzelne Komponenten eines Projekts zuerst als Proof of Concept erstellt und erst nachdem alle Komponenten einzeln funktionieren, wird alles zusammengeführt.

Im Falle dieser Web-App wurden beispielsweise die Verschlüsselung und das DHM-Protokoll sowie das UI einzeln als Proof of Concept entwickelt und erst in der fertigen Web-App mit den anderen Komponenten zusammengeführt.

**Python**

Python ist eine viel genutzte, höhere Programmiersprache, welche für sehr viele Zwecke genutzt werden kann, da es auch viele Erweiterungen dafür gibt.

Python ist vor allem für Anfänger gut geeignet, da die Sprache einfach zu lesen ist.

```
age = input("How old are you? ")

if age > 50:
    print("You are pretty old")
else:
    print("You are quite young")
```

Das obenstehende Programm fragt den Benutzer nach dem Alter und antwortet je nach Eingabe unterschiedlich.

**uuid**

Der **universal unique identifier** (*engl.* universeller, einzigartiger Identifikator) besteht aus einer zufälligen Kombination von Buchstaben und Zahlen in einer fixen Länge. Er wird verwendet, um Clients zu identifizieren und um sicherzustellen, dass alle Clients voneinander unterschieden werden können.

Beispiel einer uuid:

```
e0a3dbb1-12f6-46ab-aef2-d25162241019
```

**White Paper**

Ein White Paper ist ein Bericht oder eine Anleitung, welche den bzw. die Leser:in oftmals in Kürze, aber trotzdem ausführlich über ein komplexes Thema informiert.

## B. Anhang

### 1. Anleitung

Hier wird erklärt, wie Sie die Web-App selbst hosten und somit benutzen können. Diese Erklärung gilt primär für Windows 10.

**Achtung:** Die App erstellt lokale Dateien auf Ihrem Computer. Benutzen auf eigene Gefahr.

#### Web-App von GitHub herunterladen

Als erstes müssen Sie alle Dateien der Web-App von GitHub herunterladen.<sup>12</sup> Klicken Sie dazu auf den grünen Knopf „Code“ und wählen Sie anschliessend die Option „Download ZIP“. Sie sollten den ZIP-Ordner „Maturaarbeit-App-master.zip“ nun auf Ihrem Computer im Ordner „Downloads“ finden. Öffnen Sie den ZIP-Ordner durch Doppelklick. Darin finden Sie den Ordner „Maturaarbeit-App-master“. Ziehen Sie diesen Ordner mit gedrückter Maustaste an einen anderen Speicherort, z.B. auf Ihren Desktop.

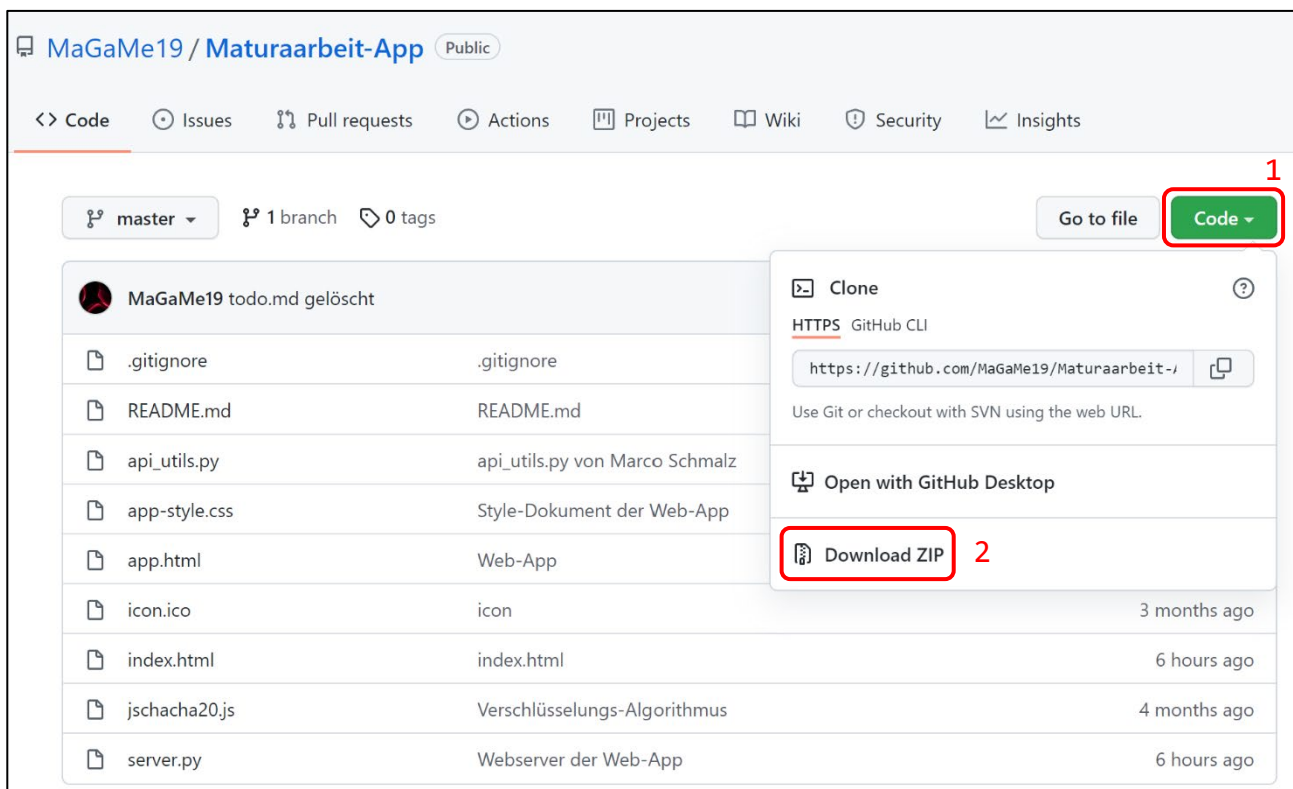


Abbildung 9: Screenshot aus dem GitHub-Profil von Mattia Metzler (Benutzername: MaGaMe19)

#### Visual Studio Code installieren

Nun sollten Sie den Datei-Editor Visual Studio Code auf ihrem Computer installieren.<sup>13</sup> Laden Sie den Installer herunter und folgen Sie dessen Anweisungen. Visual Studio Code können Sie verwenden um allerlei Dateien, hauptsächlich ausführbare Skripte, zu bearbeiten.

Starten Sie nun Visual Studio Code.

<sup>12</sup> <https://github.com/MaGaMe19/Maturaarbeit-App>

<sup>13</sup> <https://code.visualstudio.com/>

### Live Server Erweiterung für Visual Studio Code installieren

Um die Web-App zu hosten, benötigen Sie die *Live Server* Erweiterung für Visual Studio Code, entwickelt von *Ritwick Dey*. Suchen Sie dafür im Menu *Extensions* auf der linken Seite nach „ritwickdey.liveserver“. Installieren Sie die Erweiterung und starten Sie ggf. Visual Studio Code neu.

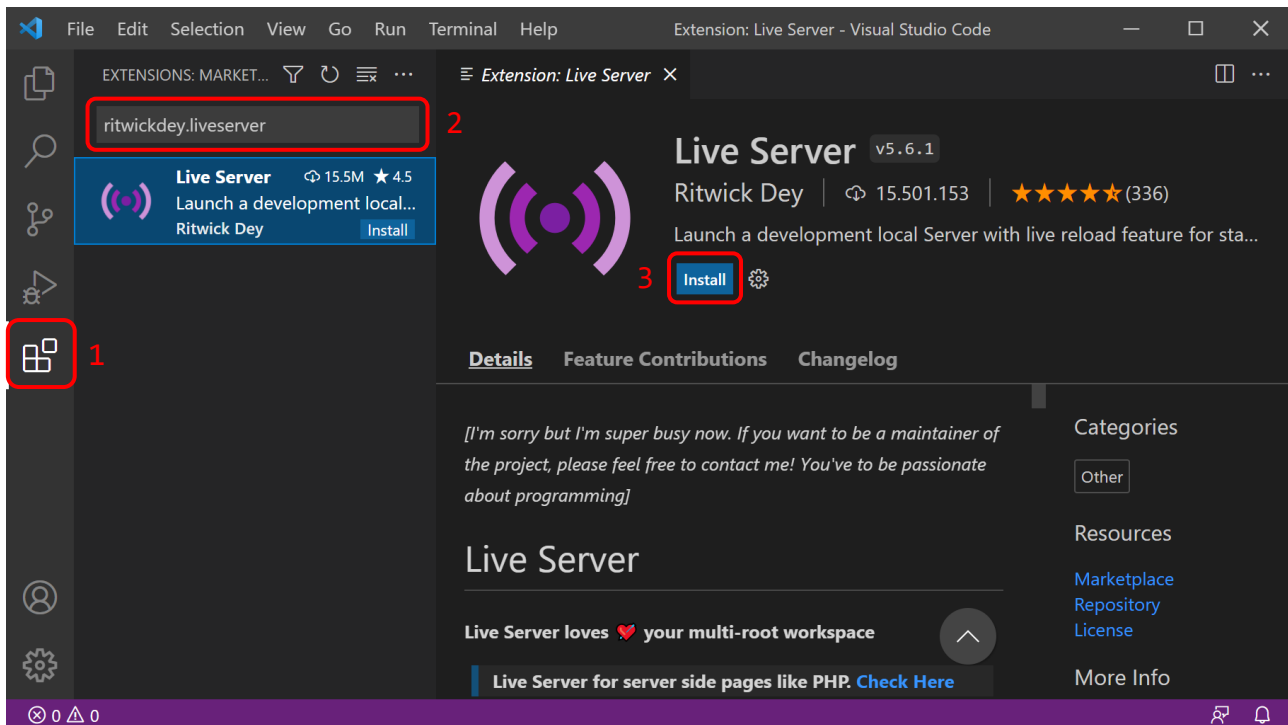


Abbildung 10: Die Live Server Erweiterung für Visual Studio Code (Screenshot)

### Proxy einrichten

Führen Sie, während Sie Visual Studio Code geöffnet haben, die Tastenkombination `ctrl + ,` aus, wodurch sich die Einstellungen von Visual Studio Code öffnen. Alternativ können Sie in der Menu-Leiste oben links wie folgt navigieren: `File > Preferences > Settings`

Öffnen Sie auf der linken Seite des neuen Fensters den Abschnitt *Extensions* und klicken Sie auf *Live Server Config*. Scrollen Sie bis zum Abschnitt *Settings: Proxy*. Ändern Sie die Einstellungen, indem Sie auf das Stift-Symbol rechts klicken, zu folgenden Werten:

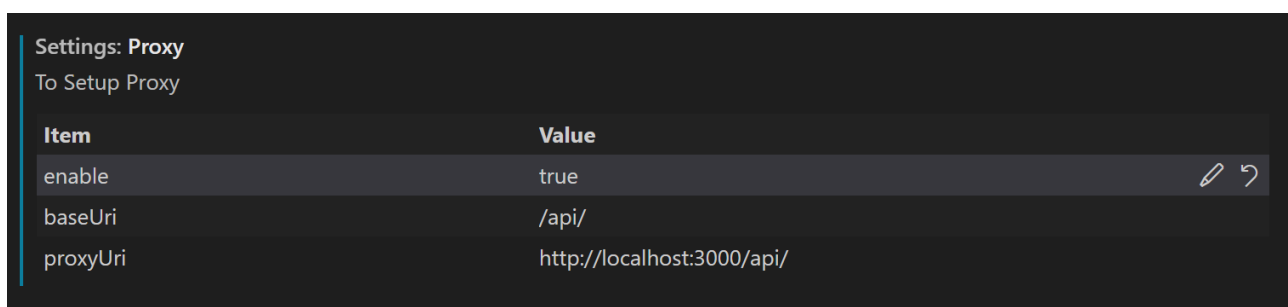


Abbildung 11: Screenshot der korrekten Einstellungen der Proxy



## Python installieren

Da der Server der Web-App in Python geschrieben ist, müssen Sie Python auf Ihrem Computer installieren.<sup>14</sup> Folgen Sie dem Installationsprogramm. Falls die Option *pip* im Schritt *Optional Features* nicht angewählt ist, wählen Sie diese an.

Öffnen Sie nun Visual Studio Code und installieren Sie die Erweiterung *Python* von Microsoft. Suchen Sie dazu im *Extensions*-Menu nach „ms-python.python“. Installieren Sie die Erweiterung und starten Sie ggf. Visual Studio Code neu.

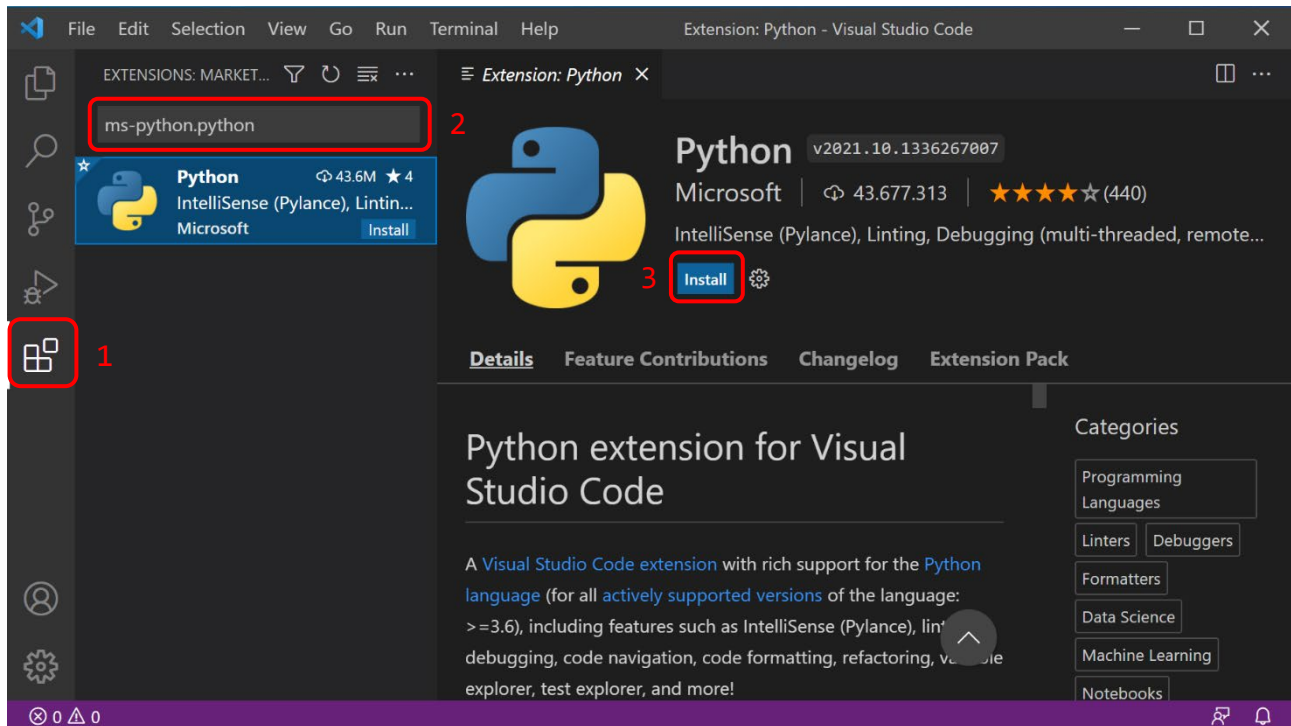


Abbildung 12: Die Python Erweiterung für Visual Studio Code (Screenshot)

<sup>14</sup> <https://www.python.org/downloads/>

## Web-App starten

Navigieren Sie in der Menu-Leiste von Visual Studio Code **File > Open Folder** und wählen Sie den im ersten Schritt heruntergeladenen Ordner „Maturaarbeit-App-master“ aus. Alle Dateien dieses Ordners sollten sich in Visual Studio Code öffnen. Rechtsklicken Sie in der Liste der Dateien links auf die Datei `server.py` und wählen Sie die Option **Run Python File in Terminal**. Öffnen Sie nun die Datei `app.html` und klicken Sie in der Leiste unten rechts auf den Knopf **Go Live**.

Nun sollte sich die Web-App in ihrem Standard-Browser öffnen und Sie können diese verwenden. Falls Sie sich nicht automatisch öffnet, starten Sie die Website <http://localhost:5500/>.

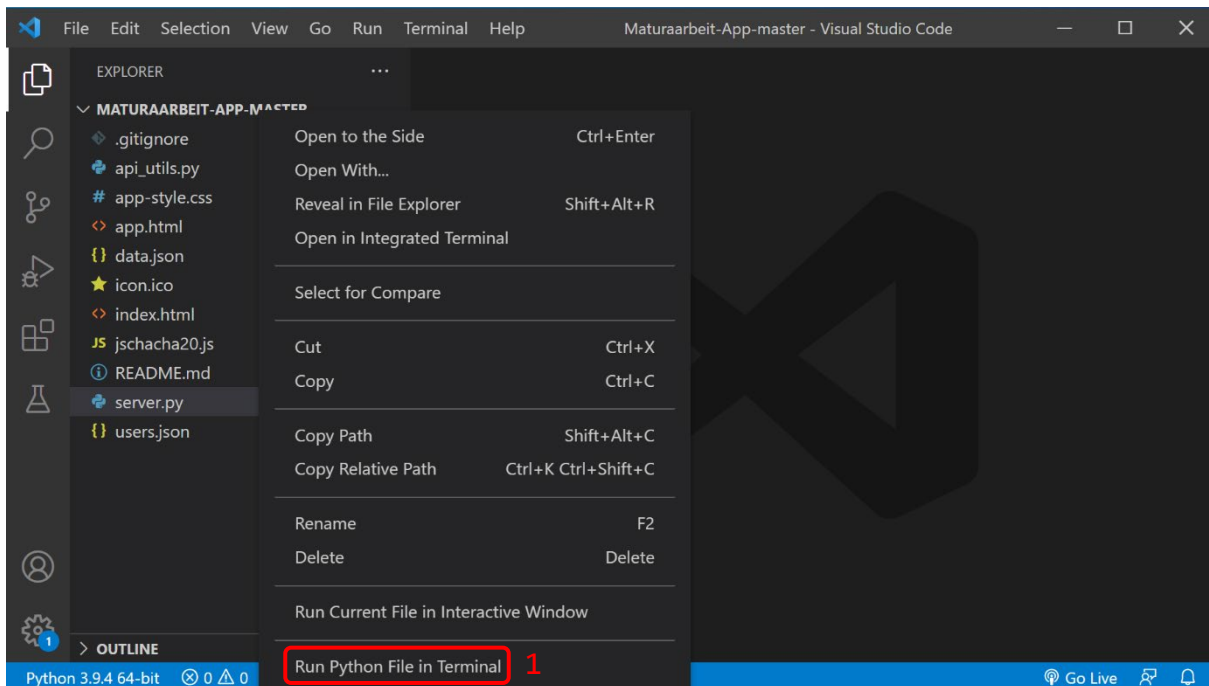


Abbildung 14: Screenshot aus Visual Studio Code mit dem Rechtsklick-Menü von `server.py` geöffnet.

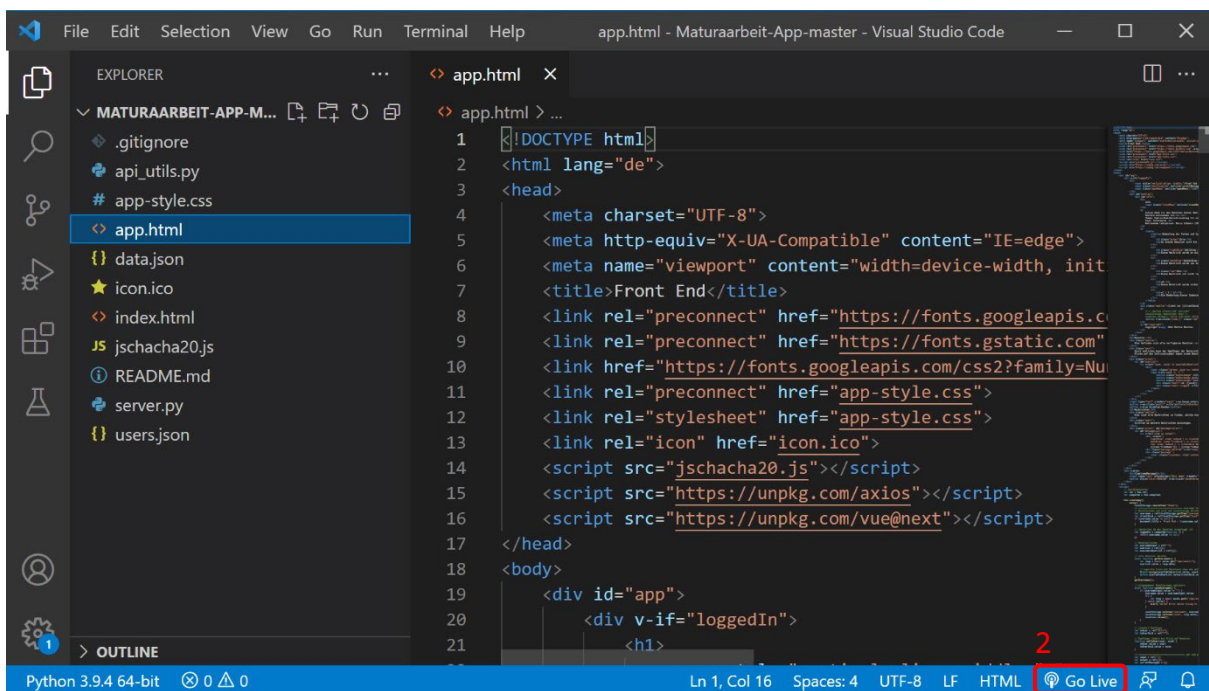


Abbildung 13: Screenshot aus Visual Studio Code mit `app.html` geöffnet.

**Achtung:** Die Web-App läuft momentan auf dem *localhost*, ein Server, welcher nur auf Ihrem Computer erreichbar ist. Das heisst, Sie können die Web-App nur mit verschiedenen Browsern und nicht mit verschiedenen Geräten nutzen. Wollen Sie die Web-App mit verschiedenen Geräten nutzen, müssen Sie die verwendete IP-Adresse (momentan *localhost* bzw. 127.0.0.1) in folgenden Orten ändern:

- In den Einstellungen von Visual Studio Code unter *Live Server Config* im Abschnitt *Extensions*:
  - Im Feld *Settings: Host*
  - Im Abschnitt *Settings: Proxy* ändern Sie im Feld *proxyUri* nur das Wort *localhost*
- In Zeile 679 in *api\_utils.py* nur das Wort *localhost*. Öffnen Sie dazu *api\_utils.py* in Visual Studio Code.

Ändern müssen Sie diese IP-Adresse zu derjenigen, welche Ihr Computer in Ihrem Netzwerk besitzt. Diese finden Sie in den Einstellungen Ihres Computers unter *Netzwerk und Internet* indem Sie auf *Eigenschaften* bei Ihrem Netzwerk klicken. Scrollen Sie ganz nach unten und finden Sie den Wert *IPv4-Adresse*.

Diese könnte z.B. so aussehen: 192.168.1.156

Alle weiteren Benutzer müssen nun in diesem Beispiel nur noch die Website <http://192.168.1.156:5500/> auf ihrem Gerät starten.

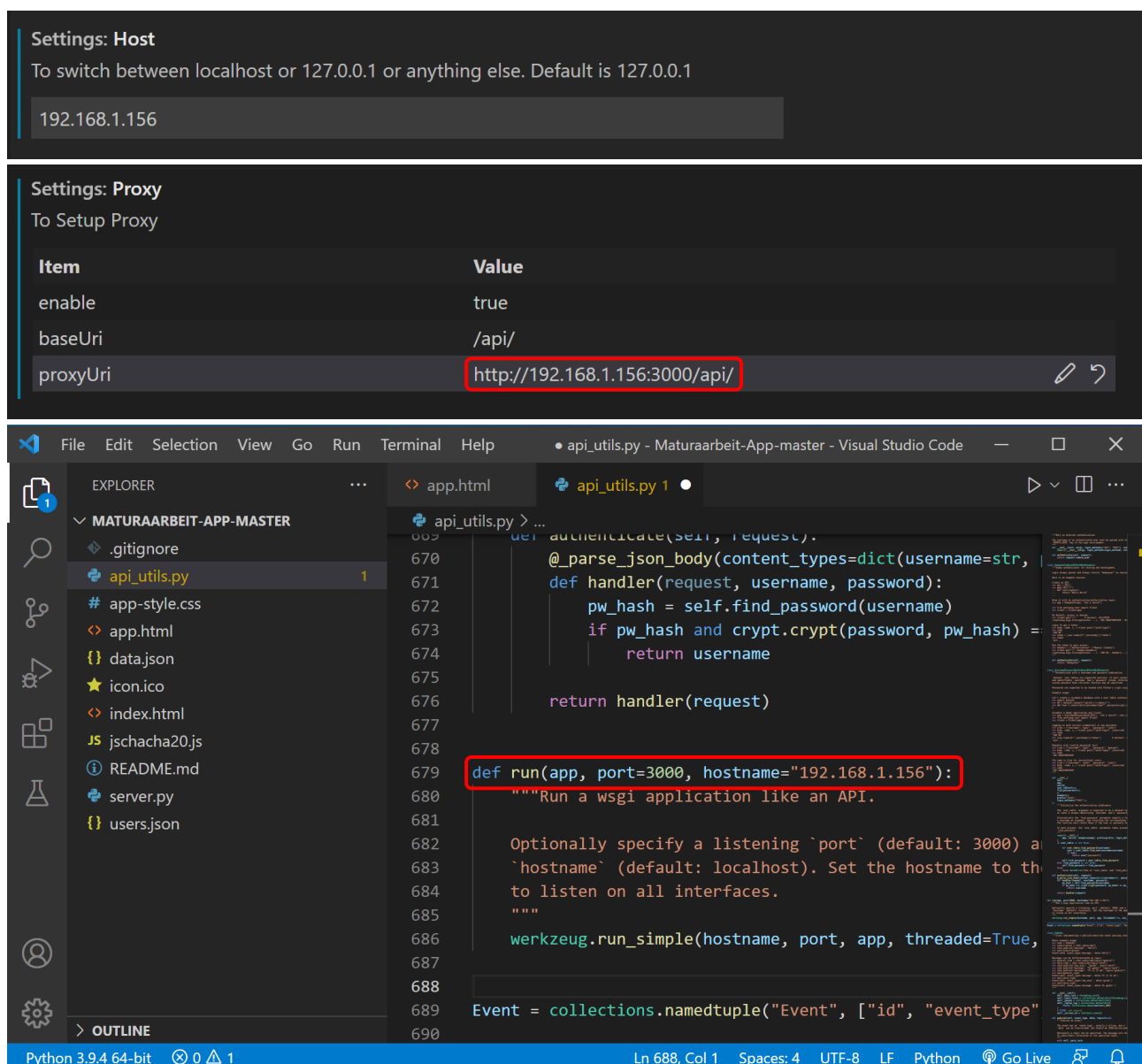


Abbildung 15: Mehrere Screenshots aus Visual Studio Code und *api\_utils.py*, welche die Orte zeigen, an denen die IP-Adresse geändert werden muss.

## 2. Quellcode

*server.py*

```
import json
import api_utils
import os
from uuid import uuid4

def main():
    """
    Diese Funktion stellt den Server hinter der Web-App dar. Dafür wird durch die Datei
    api_utils.py ein Webserver gestartet.\n
    Der Webserver nimmt requests (GET, POST, DELETE) vom Client entgegen und antwortet
    mit responses. Diese responses können anschliessend auf dem Client verwendet
    werden.\n
    """
    api = api_utils.API()

    # Dateien für Nachrichten und Benutzer
    filename = "data.json"
    filenameUsers = "users.json"

    # Dateien vorbereiten falls sie noch nicht existieren
    if not os.path.exists(filename):
        with open(filename, "w") as f:
            json.dump([], f)

    if not os.path.exists(filenameUsers):
        with open(filenameUsers, "w") as f:
            json.dump({
                "?": "Alle"
            }, f, indent=4)

    # Alle Nachrichten abrufen
    @api.GET("/api/")
    def get(request):
        with open(filename) as f:
            dataList = json.load(f)
        return dataList

    # Eine neue Nachricht hinzufügen
    @api.POST("/api/")
    def post(request, content, fromUser:str, toUser:str, type:str):
        # aktuelle Nachrichtenliste aufrufen
        with open(filename) as f:
            entryList = json.load(f)
        # Headers zur neuen Nachricht hinzufügen
        entryList.append({
            "type": type,
            "from": fromUser,
            "to": toUser,
            "content": content
        })

        # Liste aktualisieren
        with open("data.json", "w") as f:
            json.dump(entryList, f, indent=4)
```

```

    # Debug Nachricht für Client
    return f'Server: Nachricht "{content}" mit Sender "{getUsers(None)[fromUser]}"
    und Empfänger "{getUsers(None)[toUser]}" wurde zu den Nachrichten hinzugefügt.'

# Liste der Benutzer an Clients schicken
@api.GET("/api/users/")
def getUsers(request):
    with open(filenameUsers) as f:
        userList = json.load(f)

    return userList

# Neue Benutzer abspeichern
@api.POST("/api/users/")
def saveUsers(request, name:str):
    newUuid = str(uuid4()) # uuid (Universal Unique Identifier) erstellen
    with open(filenameUsers) as f:
        userList = json.load(f)
        # Benutzername wird unter dem uuid abgespeichert
        userList[newUuid] = name

    with open(filenameUsers, "w") as f:
        json.dump(userList, f, indent=4)

# uuid wird an den Benutzer übergeben
return newUuid

# Alle Nachrichten löschen
@api.DELETE("/api/")
def delete(request):
    with open(filename, "w") as f:
        json.dump([], f)

    return f"Server: Alle Nachrichten wurden gelöscht."

api_utils.run(api)

# Sicherstellen, dass der Server nicht durch importieren der Datei gestartet wird.
if __name__ == "__main__":
    main()
else:
    print("Dieses Skript muss ausgeführt werden, nicht importiert.")

```

### app.html

```

<!DOCTYPE html>
<html lang="de">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Front End</title>
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>

```

```

<link
href="https://fonts.googleapis.com/css2?family=Nunito&family=Poppins&family=Quicksand&family=Ubuntu&display=swap" rel="stylesheet">
<link rel="preconnect" href="app-style.css">
<link rel="stylesheet" href="app-style.css">
<link rel="icon" href="icon.ico">
<script src="jschacha20.js"></script>
<script src="https://unpkg.com/axios"></script>
<script src="https://unpkg.com/vue@next"></script>
</head>
<body>
  <div id="app">
    <div v-if="loggedIn">
      <h1>
        <span style="vertical-align: middle;">Front End - {{username}}</span>
        <span class="notification" onclick="scrollMessagesDown(true)" v-if="notification">🔔</span>
        <span class="openMenu" onclick="openMenu()">⚙️</span>
      </h1>
      <div id="overlay">
        <div id="info">
          <h1>
            Info
            <span class="closeMenu" onclick="closeMenu()">✖️</span>
          </h1>
          <p>
            Vielen Dank für das Benutzen dieser Web-App, welche im Rahmen
            der Maturaarbeit von Mattia Metzler am Gymnasium
            Neufeld entstanden ist.<br><br>
            Thema: End-zu-End-Verschlüsselung für eine Messaging-Web-
            App<br>
            Fach: Informatik <br>
            Betreuende Lehrperson: Marco Schmalz (SHM) <br>
          </p>
          <p>
            <table>
              <caption>Bedeutung der Farben und Symbole</caption>
              <tr>
                <td class="green">Grün</td>
                <td>An diesen Benutzer wird die Nachricht
                gesendet.</td>
              </tr>
              <tr>
                <td class="lightBlue">Hellblau</td>
                <td>Diese Nachricht wurde an dich gesendet.</td>
              </tr>
              <tr>
                <td class="darkBlue">Dunkelblau</td>
                <td>Diese Nachricht wurde von dir gesendet.</td>
              </tr>
              <tr>
                <td class="red">Rot</td>
                <td>Diese Nachricht ist nicht lesbar (verschlüsselt),
                da sie nicht an dich gesendet wurde.</td>
              </tr>
              <tr>
                <td>🔒</td>
            </table>
          </p>
        </div>
      </div>
    </div>
  </div>

```

```

        <td>Diese Nachricht wurde sicher ver- und wieder
entschlüsselt.</td>
    </tr>
    <tr>
        <td>🔑 | ⌚ | ☑️</td>
        <td>Die Bedeutung dieser Symbole kann jeweils durch
darüberfahren mit der Maus angezeigt werden.</td>
    </tr>
</table>
</p>
<div class="smaller">client id: {{clientId}}</div>
<p>
    <!-- <button class="red" onclick="
    localStorage.removeItem('keys');
    location.reload();">Alle Schlüssel löschen</button> -->
    <button v-on:click="clear()" class="red">Alle Nachrichten
löschen</button>
</p>
<p id="copyright">
    Copyright &copy; 2021 Mattia Metzler.
</p>
</div>
</div>
<h3>Benutzer</h3>
<div class="smaller">
    Hier befinden sich alle verfügbaren Benutzer.<br>
</div>
<div class="small">
    Durch anklicken kann der Empfänger der Nachricht gewählt werden.
    Scrollen um weitere Benutzer anzuzeigen.<br>
    Klicke auf das Schlüsselsymbol neben einem Benutzer, um den
    Schlüsselaustausch mit diesem Benutzer zu initialisieren.
</div>
<div class="scroll">
    <ul id="userList">
        <li v-for="(user, uuid) in availableUserList" v-
on:click="setToUser(user, uuid)">
            <div>
                <span :class="{green: uuid === toUserUuid}">{{user}}</span>
                <span v-if="uuid != '?'">
                    <button class="keyExchange" title="Schlüsselaustausch
initialisieren" v-if="getState(uuid) == 'not-initialised'" v-
on:click="initialiseKeyExchange(uuid)">🔑</button>
                    <button class="keyExchange pending-sent" title="Antwort
wird abgewartet" v-if="getState(uuid) == 'pending-sent'">⌚</button>
                    <button class="keyExchange" title="Schlüsselaustausch
bestätigen" v-if="getState(uuid) == 'pending-received'" v-
on:click="confirmKeyExchange(uuid)">☑️</button>
                    <div class="small">id: {{uuid}}</div>
                    <div class="small cropped" v-if="getState(uuid) ==
'completed'">key: {{keyArray[uuid]["content"]}}</div>
                </span>
            </div>
        </li>
    </ul>
</div>

```



```

        <input type="text" v-model="input" v-on:keyup.enter="post()"
:placeholder="'Neue Nachricht an ' + [[toUser]]">
        <button v-on:click="post()" v-if="getState(toUserUuid) == 'completed' ||
toUserUuid == '?'">Senden</button>
        <button v-else disabled>Senden</button>
        <h3>Nachrichten</h3>
        <div class="smaller">
            Hier sind alle Nachrichten zu finden, welche von allen Benutzern
versendet wurden.
        </div>
        <div class="small">
            Scrollen um weitere Nachrichten anzuzeigen.
        </div>
        <div class="scroll" id="message-scroll">
            <ul id="messageList">
                <li v-for="item in output">
                    <div :class="{
                        lightBlue: item['toUuid'] == clientUuid,
                        darkBlue: item['fromUuid'] == clientUuid && item['toUuid']
!= '?',
                        red: item['toUuid'] != clientUuid && item['toUuid'] != '?'
&& item['fromUuid'] != clientUuid}">
                        {{item["fromName"]}} ⇌ {{item["toName"]}}</div>
                        <div class="message darkred" v-if="item['toUuid'] != clientUuid
&& item['fromUuid'] != clientUuid && item['toUuid'] != '?'">Verschlüsselte
Nachricht:</div>
                        <div class="message" >
                            <span :class="{rainbow: item['content'] ==
'\{\0_0}/'}">{{item["content"]}}</span>
                        </div>
                    </li>
                </ul>
            </div>
        </div>
        <div v-else>
            <h1>{{welcomeMessage}}</h1>
            <input type="text" placeholder="Dein Name" v-model="usernameInput" v-
on:keyup.enter="saveUsername()">
            <button style="color:#00b300" v-on:click="saveUsername()">SAVE</button>
        </div>
    </div>
    <script>
        // Vue-Komponenten
        var ref = Vue.ref;
        var computed = Vue.computed;

        Vue.createApp({
            setup() {
                localStorage.removeItem("theme");
                /* ===== username handling
===== */
                // Benutzername und uuid vom Localstorage abrufen, Titel der Website
ändern

                let username = ref(localStorage.getItem("username"));
                let clientUuid = ref(localStorage.getItem("uuid"));
                if (username.value != null) {
                    document.title = `Front End - ${username.value}`;

```



```

    }

    // überprüfen ob der Benutzer eingeloggt ist
    let loggedIn = computed(function () {
        return username.value != null;
    })

    // Benutzerlisten
    let usernameInput = ref("");
    let userList = ref({});
    let availableUserList = ref({});

    // alle Benutzer abrufen
    async function getUsers() {
        var resp = await axios.get("/api/users/");
        userList.value = resp.data;

        // separate Liste mit Benutzern ohne den aktuellen Client, um diese
        // auf der Website anzuzeigen
        Object.assign(availableUserList.value, userList.value);
        delete availableUserList.value[clientUid.value];
    }
    getUsers();

    // eingegebener Benutzername speichern
    async function saveUsername() {
        if (usernameInput.value != "") {
            username.value = usernameInput.value;
            try {
                var resp = await axios.post("/api/users/", {name:
username.value});
            } catch (error) {
                alert(`Server Error while trying to save username
(${error})`)
            }

            localStorage.setItem("username", username.value);
            localStorage.setItem("uuid", resp.data);
            location.reload();
        }
    }

    // standard Empfänger
    let toUser = ref("Alle");
    let toUserUuid = ref("?");

    // Empfänger ändern bei Klick auf Benutzer
    function setUser(user, uuid) {
        toUser.value = user;
        toUserUuid.value = uuid;
    }

    /* ===== get and post ===== */
    var input = ref("");
    var output = ref([]);
    var currentMessages = [];

```

```

// alle Nachrichten abrufen
async function get() {
  /* Funktionsweise:
   - axios schickt eine request an den Server,
   - await wartet bis die response zurück kommt
   - data ist die antwort des Servers
   - das Ganze muss in einer async Funktion ablaufen
  */
  var resp = await axios.get("/api/");
  let messageList = [];
  if (resp.data.length != 0) {
    for (const listEntry of resp.data) {
      // Wenn es sich um eine Nachricht handelt, wird sie auf der
Website angezeigt

      if (listEntry["type"] == "message") {
        let content = listEntry["content"]
        let messageArray = new
Uint8Array(Object.values(listEntry["content"]));

        // ist die Nachricht an den momentanen Benutzer
gerichtet, wird sie entschlüsselt mit dem Schlüssel im Localstorage
        if (listEntry["to"] == clientUuid.value) {
          content = decrypt(listEntry["from"], messageArray);
        }
        // stammt die Nachricht vom aktuellen Benutzer, wird
sie ebenfalls entschlüsselt
        else if (listEntry["from"] == clientUuid.value &&
listEntry["to"] != "?") {
          content = decrypt(listEntry["to"], messageArray);
        }
        // ist die Nachricht weder an den momentanen Benutzer
gerichtet noch stammt sie vom aktuellen Benutzer, wird sie nicht entschlüsselt, nur in
Text gewandelt

        else if (listEntry["to"] != "?") {
          content = decAscii.decode(messageArray);
        }
        // secret - siehe Titelseite MA
        if (content.toLowerCase() == "5c7b305f307d2f") {
          content = "\\{0_0}/";
        }
        messageList.push({
          "fromUuid": listEntry["from"],
          "fromName": userList.value[listEntry["from"]],
          "toUuid": listEntry["to"],
          "toName": userList.value[listEntry["to"]],
          "content": content
        })
        currentMessages = messageList;

        output.value = messageList;

        // handelt es sich um einen Schlüsselaustausch so wird der
Schlüssel gespeichert

      } else if (listEntry["type"] == "keyExchange" &&
listEntry["to"] == clientUuid.value) {
        if (!(getState(listEntry["from"]) == "completed")) {

```

```

// der erhaltene öffentliche Schlüssel wird
abgespeichert
    saveKey(listEntry["from"], listEntry["content"],
"pending-received")
    colorLog(`Schlüsselaustausch wurde von
${userList.value[listEntry["from"]]} initialisiert.\nErhaltener öffentlicher Schlüssel:
${listEntry["content"]}`);
    }
    } else if (listEntry["type"] == "keyExchangeConfirmation"
&& listEntry["to"] == clientUuid.value) {
        // der Schlüsselaustausch wurde vom Gesprächspartner
bestätigt
        if (!(getState(listEntry["from"]) == "completed")) {
            completeKeyExchange(listEntry["from"],
listEntry["content"])
        }
    }
}
// überprüfen ob die Letzte Nachricht an den momentanen
Benutzer gerichtet ist, falls ja, Benachrichtigung anzeigen.
if (currentMessages.length != 0 && clientUuid.value ==
currentMessages[currentMessages.length - 1]["toUuid"]) {
    notification.value = true;
} else {
    notification.value = false;
}
} else {
    output.value = [];
}
}
get();

// eine neue Nachricht hinzufügen
async function post() {
    if (input.value != "" && getState(toUserUuid.value) == "completed")
{
        // überprüfen ob der Schlüsselaustausch mit dem Empfänger
abgeschlossen ist, falls ja, Nachricht verschlüsseln
        if (toUserUuid.value != "?") {
            content = encrypt(toUserUuid.value, input.value);
        } else {
            content = input.value;
        }
        // Nachricht an Server schicken
        try {
            resp = await axios.post("/api/", {
                type: "message",
                content: content,
                fromUser: clientUuid.value,
                toUser: toUserUuid.value});
            colorLog(`Client: Nachricht "${content}" mit dem Ziel
"${userList.value[toUserUuid.value]}" wurde an den Server gesendet.`);
            colorLog(resp.data);
        } catch (error) {
            alert(`Server Error while trying to send message
(${error})`)
        }
    }
}

```

```

        input.value = "";
        scrollMessagesDown(false);
    }
}

/* ===== Encryption
===== */

let crypto = window.crypto;
let enc = new TextEncoder("utf-8"); // utf-8 zu Uint8Array, benötigt um
Nachrichten zu verschlüsseln
let dec = new TextDecoder("utf-8");
// Uint8Array zu Ascii für verschlüsselte Nachrichten damit sie schöner
aussehen

let decAscii = new TextDecoder("ascii");

var nonceArray = new Uint8Array(12); // zufällige Nonce (= number used
once), benötigt um Nachrichten zu verschlüsseln

// square-and-multiply-Algorithmus (siehe Maturaarbeit, Kapitel 3.3.2.)
function squareMultiply(x, k, mod) {
    var y = x % mod; // Zwischenergebnis
    var kBinary = k.toString(2); // exponent in binär
    kBinary = kBinary.replace(kBinary[0], ""); // erstes "1" entfernen
    -> mit x starten (1^2 * x = x)

    for (const i of kBinary) {
        y = (y * y) % mod; // für jedes "Q" wird quadriert
        if (i == 1) {
            y = (y * x) % mod; // für jedes "M" wird mit x
multipliziert
        }
    }
    return y;
}

var g = 5n; // generator
// primzahl, 256-Bit
https://asecuritysite.com/encryption/random3?val=256
var n =
40607624323698004944288610351048360638553718145867992171012767459255954639447n;

// Nachrichten verschlüsseln
function encrypt(uuid, message) {
    let s = BigInt(keyArray.value[uuid]["content"]);

    // Schlüssel in Uint8Array umwandeln durch Bit-Shifting
    let keyUint8Array = new Uint8Array(32);
    for (let i = 0; i < keyUint8Array.length; i++) {
        keyUint8Array[i] = Number(s % 256n);
        s = s >> 8n;
    }

    const encryptor = new JSChaCha20(keyUint8Array, nonceArray); //
chacha20-Verschlüsselungsalgorithmus

    let rawEncryptedMessage = encryptor.encrypt(enc.encode(message));

```

```

        colorLog(`Client: Nachricht "${message}" wurde mit dem Schlüssel
"${BigInt(keyArray.value[uuid]["content"])}" zugehörig zum Benutzer
"${userList.value[uuid]}" verschlüsselt.`)
        return rawEncryptedMessage;
    }

    // Nachricht entschlüsseln
    function decrypt(uuid, messageUint8) {
        let s = BigInt(keyArray.value[uuid]["content"]);

        // Schlüssel in Uint8Array umwandeln durch Bit-Shifting
        let keyUint8Array = new Uint8Array(32);
        for (let i = 0; i < keyUint8Array.length; i++) {
            keyUint8Array[i] = Number(s % 256n);
            s = s >> 8n;
        }

        const decryptor = new JSChaCha20(keyUint8Array, nonceArray);
        return dec.decode(decryptor.decrypt(messageUint8));
    }

    // Schlüssel vom Localstorage abrufen
    let keyArray = ref(JSON.parse(localStorage.getItem("keys")) || {});

    // Schlüsselaustausch initialisieren
    async function initialiseKeyExchange(toUuid) {
        var privateKey = BigInt(Math.floor(Math.random() * Number(n))); //
privater Schlüssel
        saveKey(toUuid, privateKey, "pending-sent"); // privater Schlüssel
im Localstorage speichern

        var publicKey = squareMultiply(g, privateKey, n); // öffentlicher
Schlüssel

        try {
            // öffentlicher Schlüssel dem Empfänger übermitteln
            await axios.post("/api/", {
                type: "keyExchange",
                content: publicKey.toString(),
                fromUser: clientUuid.value,
                toUser: toUuid})

            } catch (error) {
                alert(`Server Error while trying to initialise key exchange
(${error})`)
            }

        colorLog(`Schlüsselaustausch mit "${userList.value[toUuid]}"
initialisiert.\nPrivater Schlüssel: ${privateKey}\nÖffentlicher Schlüssel:
${publicKey}`);
    }

    // Schlüsselaustausch bestätigen
    async function confirmKeyExchange(uuid) {
        var privateKey = BigInt(Math.floor(Math.random() * Number(n))); //
privater Schlüssel
        var publicKey = squareMultiply(g, privateKey, n); // öffentlicher
Schlüssel

```

```

    try {
      // öffentlicher Schlüssel dem Empfänger übermitteln
      await axios.post("/api/", {
        type: "keyExchangeConfirmation",
        content: publicKey.toString(),
        fromUser: clientId.value,
        toUser: uuid});
    } catch (error) {
      alert(`Server Error while trying to confirm key exchange
    (${error})`)
    }

    // gemeinsamer geheimer Schlüssel berechnen und abspeichern
    (Öffentlicher Schlüssel des Empfängers befindet sich im Localstorage)
    var sharedSecretKey =
squareMultiply(BigInt(keyArray.value[uuid]["content"]), privateKey, n);
saveKey(uuid, sharedSecretKey, "completed");

    colorLog(`Schlüsselaustausch mit "${userList.value[uuid]}"
bestätigt.\nPrivater Schlüssel: ${privateKey}\nÖffentlicher Schlüssel:
${publicKey}\nGemeinsamer Geheimer Schlüssel: ${sharedSecretKey}`);
  }

  // Schlüsselaustausch abschliessen, nachdem der Gesprächspartner ihn
bestätigt hat
  function completeKeyExchange(uuid, publicKey) {
    // gemeinsamer geheimer Schlüssel berechnen und abspeichern
    (eigener privater Schlüssel befindet sich im Localstorage, öffentlicher Schlüssel des
Empfängers erhält man)
    var sharedSecretKey = squareMultiply(BigInt(publicKey),
    BigInt(keyArray.value[uuid]["content"]), n);
    saveKey(uuid, sharedSecretKey, "completed");

    colorLog(`Schlüsselaustausch wurde von "${userList.value[uuid]}"
bestätigt.\nGemeinsamer Geheimer Schlüssel: ${sharedSecretKey}`);
  }

  // Funktion um Schlüssel abzuspeichern
  function saveKey(uuid, key, state) {
    keyArray.value[uuid] = {
      "content": key.toString(),
      "state": state
    };
    localStorage.setItem("keys", JSON.stringify(keyArray.value));
  }

  // Funktion um den Status des Schlüsselaustausches zu erhalten
  function getState(uuid) {
    if (uuid == "?") {
      return "completed"
    } else {
      if (!(uuid in keyArray.value)) {
        return "not-initialised"
      } else {
        return keyArray.value[uuid]["state"]
      }
    }
  }

```

```

    }
  }

  /* ===== utility functions ===== */
  // Alle Nachrichten löschen
  async function clear() {
    try {
      resp = await axios.delete("/api/");
      colorLog(resp.data);
    } catch (error) {
      alert(`Server Error while trying to clear messages
(${error})`);
    }
  }

  // Benachrichtigung
  var notification = ref(false);

  // Nachrichten jede Sekunde aktualisieren
  setInterval(() => (get(), getUsernames()), 1000); // reload

  // Begrüßungsnachricht auf der Anmeldeseite mit richtiger Zeit
  var welcomeMessage = ref("Guten Tag");
  var currentHour = parseInt((new Date).toString().substring(0, 2));
  if (currentHour < 12) {
    welcomeMessage.value = "Guten Morgen";
  } else if (currentHour >= 17) {
    welcomeMessage.value = "Guten Abend";
  }

  /* ===== return ===== */
  return {
    post,
    input,
    output,
    clear,
    loggedIn,
    username,
    clientId,
    usernameInput,
    saveUsername,
    getUsernames,
    userList,
    availableUserList,
    setUser,
    toUser,
    toUserId,
    keyArray,
    initialiseKeyExchange,
    confirmKeyExchange,
    getState,
    notification,
    welcomeMessage
  }
}

```

```

    }).mount("#app");

    // Menu öffnen und schliessen
    var overlay = document.querySelector("#overlay");
    function openMenu() {
        overlay.style.top = 0;
    }
    function closeMenu() {
        overlay.style.top = "-2000px";
    }

    // Nachrichten nach unten scrollen, wenn man eine neue Nachricht hinzufügt
    var messageList = document.querySelector("#message-scroll");
    function scrollMessagesDown(instantly) {
        if (instantly)
            messageList.scrollTop = messageList.scrollHeight
        else
            setTimeout(() => (messageList.scrollTop = messageList.scrollHeight),
1000);
    }
    scrollMessagesDown(false);

    // Farbige Erklärungen in der Konsole
    function colorLog(logText) {
        console.log(`%c${logText}`, "color:#698bfc")
    }
}
</script>
</body>
</html>

```

### app-style.css

```

:root {
    --color-light: #ffffaf;
    --color-input: #f1ece7;
    --color-input-hover: #e7e0db;
    --color-input-active: #ddd6d0;
    --color-dark: #2e3747;
    --color-disabled: #a7a7a7;
    --scroll-max-height: 21vh;
    --body-padding-side: 6vw;
    --transition-time: .3s;
}

body {
    padding: 15px var(--body-padding-side) 0 var(--body-padding-side);
    color: var(--color-dark);
    background-color: var(--color-light);
    overflow-x: hidden;
    scroll-behavior: smooth;
}

h1 {
    font-size: 2.5em;
    margin-top: 0;
}

```



```

h3 {
  border-bottom: 1px solid var(--color-dark);
  font-size: 1.3em;
  margin-bottom: 8px;
}
* {
  font-family: 'Nunito', sans-serif; /* available fonts: Poppins, Nunito, Quicksand,
  Ubuntu */
  font-weight: 600;
  /* font-size: 0.98em; */
  outline: none;

  scrollbar-color: #657592 var(--color-dark);
  scrollbar-highlight-color: #97a6c2;
  scrollbar-width: thin;
}
::-webkit-scrollbar {
  width: 10px;
  background-color: var(--color-dark);
  border-radius: 5px;
}
::-webkit-scrollbar-thumb {
  background-color: #657592;
  border-radius: 5px;
}
::-webkit-scrollbar-thumb:hover {
  background-color: #97a6c2;
}

/* ===== input
===== */
input {
  margin: 10px 5px 5px 0;
  font-size: 1em;
  background-color: var(--color-input);
  padding: 7px;
  color: var(--color-dark);
  border: 1px solid var(--color-dark);
  border-radius: 3px;
  width: 50vw;
}
input:focus {
  background-color: var(--color-input-hover);
}
::placeholder {
  color: #717e92;
}
button {
  min-width: 40px;
  text-align: center;
  font-size: 1em;
  background-color: var(--color-input);
  padding: 7px;
  border-radius: 5px;
  margin-right: 5px;
  transition: background-color var(--transition-time);
}

```

```

button:hover {
  background-color: var(--color-input-hover);
}
button:enabled {
  color: var(--color-dark);
  border: 1px solid var(--color-dark);
  cursor: pointer;
}
button:disabled {
  color: var(--color-disabled);
  border: 1px solid var(--color-disabled);
}
button:disabled:hover {
  background-color: var(--color-input);
}
button:active {
  background-color: var(--color-input-active);
}
button.pending-sent {
  border: 1px solid var(--color-input-hover);
  cursor: default;
}
button.pending-sent:hover {
  background-color: var(--color-input);
}

/* ===== messages
===== */
.scroll {
  overflow-y: scroll;
}
.scroll #messageList {
  max-height: 24vh;
}
.scroll #userList {
  max-height: 22vh;
}
#message-scroll {
  scroll-behavior: smooth;
}
#messageList {
  list-style: none;
  display: grid;
  margin: 10px 0 0 -15px;
}
#messageList li {
  padding-bottom: 15px;
  word-break: break-word;
}
#userList {
  list-style: none;
  display: grid;
  margin-left: -15px;
}
#userList li {
  cursor: pointer;
  margin-bottom: 15px;
}

```

```

}
#userlist li:last-of-type {
    margin-bottom: 0;
}
.keyExchange {
    float: right;
    margin-right: 25px;
}
.small {
    font-size: 12px;
    margin-bottom: 3px;
}
.smaller {
    font-size: 15px;
}
.cropped {
    text-overflow: ellipsis;
    white-space: nowrap;
    width: 250px;
    overflow: hidden;
    transition: width var(--transition-time) ease-out;
}
.cropped:hover {
    width: 60vw;
}
.message {
    margin-left: 20px;
}
.notification {
    font-size: 0.7em;
    vertical-align: middle;
    margin-left: 15px;
    cursor: pointer;
}

/* ===== colors
===== */
.red {
    color: #e40000 !important;
}
.darkred {
    color: #9c2323;
}
.green {
    color: #009c3c !important;
}
.lightBlue {
    color: #008ed6 !important;
}
.darkBlue {
    color: #3765fb !important;
}
.rainbow {
    background-image: linear-gradient(80deg, #ff0000, #FF7F00, #d3d331, #009200,
#0000FF, #4B0082, #9400D3);
    background-clip: text;
    -webkit-background-clip: text;

```

```

    color: transparent;
    font-size: 1.5em;
}
div.lightBlue::after, div.darkBlue::after {
    content: " 📁 ";
}
.hidden {
    display: none;
}

/* ===== menu
===== */
.openMenu {
    float: right;
    cursor: pointer;
    transition: color var(--transition-time);
    vertical-align: middle;
}
.closeMenu {
    float: right;
    cursor: pointer;
    transition: color var(--transition-time);
}
.openMenu:hover, .closeMenu:hover {
    color: #4680ff;
}
#overlay {
    display: flex;
    justify-content: center;
    align-items: center;
    z-index: 1;
    width: 100vw;
    height: 100vh;
    position: absolute;
    left: 0;
    top: -2000px;
    background: rgba(85, 85, 85, 0.9);
    transition: top var(--transition-time) ease-out;
}
#overlay p {
    margin-top: 30px;
    font-size: 1.2em;
}
#info {
    z-index: 2;
    width: 86vw;
    background-color: var(--color-light);
    border-radius: 8px;
    font-size: 0.9em;
    padding: 20px 20px 0 20px;
}
caption {
    margin-bottom: 10px;
    text-align: left;
}
table {
    border-collapse: collapse;

```

```
    font-size: 1em;
}
td {
    padding: 5px 30px 5px 5px;
    margin: 0;
    border: 1px dashed #555555;
}

/* ===== responsive
===== */
@media screen and (min-width: 1000px) {
    body {
        padding: 15px calc(var(--body-padding-side) + 8vw) 0 calc(var(--body-padding-side) + 8vw);
    }
    #info {
        width: 71vw;
        font-size: 1em;
    }
}
@media screen and (min-width: 1500px) {
    body {
        padding: 15px calc(var(--body-padding-side) + 13vw) 0 calc(var(--body-padding-side) + 13vw);
    }
    #info {
        max-width: 61vw;
    }
}
@media screen and (min-height: 800px) {
    .scroll #messageList {
        max-height: 35vh;
    }
}
```



### **Selbständigkeitserklärung Maturaarbeit**

Ich erkläre hiermit, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Das Informationsblatt «Plagiatserkennung» ist mir bekannt und somit auch die Konsequenzen eines Teil- oder Vollplagiats.

Ort und Datum: \_\_\_\_\_

Unterschrift der Verfasserin /des Verfassers: \_\_\_\_\_