29.04.2021 - 30.04.2021

Projektwoche 2021
GhostChat

Bearbeitet von:

Mustafa

Phil

Daniel

Chihab

Inhaltsverzeichnis

Ausgangsituation

Durch das vermehrte Aufkommen von Datenlecks bei größeren Sozialen Medien und die Sicherheit der Daten bei einigen Messengern nicht gewährleistet ist, wollten wir einen Messenger aufbauen, welcher die Kriterien erfüllt, die wir uns wünschen und mit Nutzerdaten sicher umgeht.

Wir haben uns dazu entschieden das Projekt Open Source zu gestalten, damit ist der Code öffentlich zugänglich. Dadurch kann jeder nachvollziehen, was mit seinen Daten passiert. Diese Methode gefährdet die Sicherheit der Daten nicht, da trotz all dem Preisgeben die Verschlüsselung der Nachrichten nicht geknackt werden kann. Die aktiven Nutzer sind höchstwahrscheinlich keine besonders große Menge, dennoch sind es Personen, die einen hohen Wert auf ihre Datensicherheit legen. Unteranderem können auch Unternehmen den Messenger benutzen, um z.B. interne Gespräche zu führen oder um den Kundensupport über diesen Messenger zu führen.

In unserem Team hat niemand so richtig eine feste Rolle, da wir alle gemeinsam an einem Punkt arbeiten, damit jeder auf demselben Stand ist.
Jeder beteiligt sich an dem Projekt mit seinen Ideen und diese werden dann ausdiskutiert und zu einer bestmöglichen Form ausgeführt.
Jeder aus unserer Gruppe funktioniert als lokaler Tester, damit wir die Anwendung auf verschiedenen Umgebungen und Endgeräten testen können.
Außerdem tauschen wir unsere erarbeiteten Dinge über ein Repository auf der GitHub Plattform miteinander aus.

Die einzige Schnittstelle, die wir für unser Projekt benötigt haben, ist eine Library namens Sodium, welche wir für die Verschlüsslung verwenden. Der Rest basiert auf PHP, JavaScript, Html, Css und die Anwendung wird über ein Apache Web Server gehostet.

Beachtet man den finanziellen Aspekt des Projekts würde es sich eher weniger rentieren. Dies sieht man an folgender Grafik:

Wie man der Grafik entnehmen kann, belaufen sich die Gesamtkosten auf 129,87 €. Das ergibt sich aus den individuellen Stundenlöhnen, die alle Projektmitglieder erhalten sowie durch die Betriebskosten für einen Server, welche sich auf 5,00 € im Monat belaufen. Für die Gesamtkosten haben wir nur einen Monat mit in die Rechnung einbezogen, da wir den Server theoretisch nur für maximal ein paar Wochen bräuchten.
Dadurch dass die Betriebskosten nur 5,00 € pro Monat betragen würden, wäre das Projekt auch günstig in der Zukunft für die Nutzer zu betreiben. Die Nutzer könnten dann kostenlos einen sicheren Messenger verwenden, wo sie vollen Einblick in die Source erhalten und dadurch guten Gewissens den Messenger nutzen können.

Des Weiteren haben wir auch nicht monetäres Nutzen beachtet, da man mit unserem Chat innerhalb eines Unternehmens sehr gut kommunizieren kann, da der Chat verschlüsselt ist und dementsprechend für Dritte nicht einsehbar ist. Damit wird kein Datenschutzrecht verletzt und so können über wichtige Dinge gesprochen werden oder auch über kurze Dinge, dann müsste man nicht immer den E-Mailverkehr nutzen.

Anforderungsdefinition

Wir haben uns dazu entschieden einen Messenger zu kreieren, welcher einfach zu bedienen ist und für jede Person verständlich. Dazu haben wir uns selbst als Aufgabe gestellt, dass der Messenger einem hohen Sicherheitsstandard entspricht, und welcher Userdaten mit Respekt behandelt, indem so wenig Nutzerdaten gesammelt werden wie möglich. Unter anderem sollten Textnachrichten nach dem sie gelesen wurden automatisch gelöscht werden, damit sie anschließend nicht mehr zurückverfolgt werden können. Dabei soll der Nutzen hinter dem Messenger nicht in Vergessenheit geraten und dient zum sicheren Kommunizieren.

Unser Messenger besitzt verschiedene funktionale Anforderungen und nicht funktionale Anforderungen.
Zu den funktionalen Anforderungen gehören, dass man sich einen Account erstellen kann, Nachrichten senden, sein Avatar ändern, das Passwort ändern und Freundschaftsanfragen an oder ablehnen kann. Zu den nicht funktionalen Anforderungen gehören, dass die Nachrichten verschlüsselt und entschlüsselt werden. Außerdem ist einer der Anforderungen, dass die Nachrichten gelöscht werden, sobald der Nutzer sie sieht. Zu guter Letzt wird der Chat alle 30 Sekunden aktualisiert, um den User konstant auf dem Laufenden zu halten.

Die Anforderungsdefinition und Anforderungsverteilung hatten sich schon zum Teil aus dem vorher bereitgestellten Gantt Diagramm ergeben. Zum Start des Projektes sind wir dann die einzelnen Punkte noch einmal durchgegangen und haben diese genauer definiert und in Reihenfolge und Kategorie eingeteilt. Dann haben wir uns darauf geeinigt wer welche Aufgaben übernimmt.
Von unserem geplanten Vorgehen sind wir nur dann kurz abgewichen, wenn es an einer Stelle Probleme gab oder andere Sachen dazwischengekommen sind.

Da unser Projekt im Rahmen einer Projektwoche der Schule entstanden ist, kamen bei uns keine Geschäftsprozesse zum Einsatz.

Planung

Im Rahmen unserer Projektplanung haben wir folgende Aufgaben definiert: Die Erstellung eines Git Repositories, im Frontend, die Login Seite, Registrierungsseite, Chat Seite, und die Kontakt Verwaltung. Die Logik im Backend für den Login und die Registrierung, sowie Kontaktverwaltung, Chatsystem und die Datenbank. Zusätzlich mussten wir einen Server einrichten, wobei wir einen Webspace und eine Datenbank anlegen mussten, sowie eine passende Domain verwaltet werden muss. Das Git Repository, sowie Frontend wurde von Phil erstellt. Dabei war das Git Repository schnell angelegt und nach 30 Minuten hat es für alle funktioniert. Danach hat er im Frontend die Login und Registrierungs Seite gebaut, welche jeweils 1 Stunde in Anspruch nahm. Danach kam für ihn das Aufwendigste, welches die Chat Seite war, dafür wurden 3,5 Stunden beansprucht. Zuletzt musste noch eine Kontaktverwaltung erstellt werden welche nochmal 2 Stunden beanspruchte. Im Backend hat die Logik

für Login und Registrierung Chihab übernommen. Dafür hat er jeweils 3,5 Stunden gebraucht. Danach müsste er noch den Server einrichten, wo er jeweils für die Webspace, Datenbank und Domain, 1 Stunde benötigte. Die Kontaktverwaltung wurde von Daniel bearbeitet und hat 4 Stunden beansprucht.
Außerdem hat er noch die Datenbank aufgesetzt und entwickelt, was auch nochmal 4 Stunden in Anspruch nahm. Das Chat System war einer der größten Abschnitte im Projekt und wurde von Mustafa übernommen. Dafür musste er volle 8 Stunden in Anspruch nehmen.

Durchführung

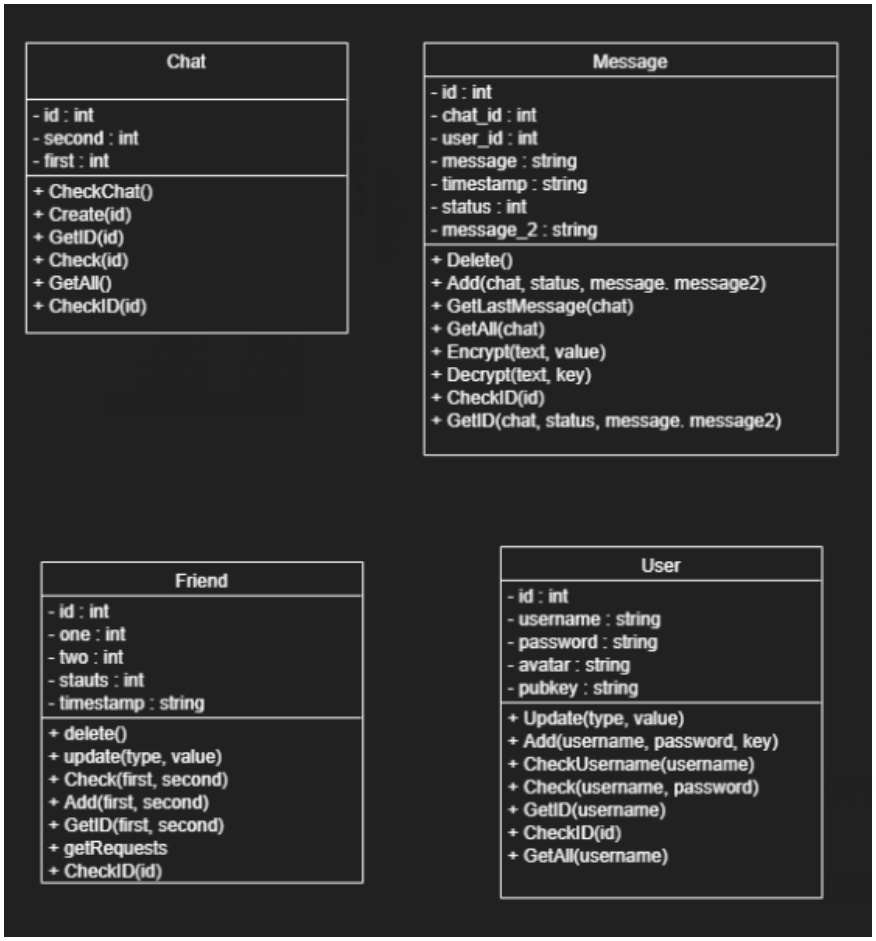
Für die technische Umsetzung des Projekts haben wir HTML, PHP, CSS & JavaScript genutzt. Im Frontend kam HTML, CSS & JavaScript zum Einsatz.

Im Backend entschieden wir uns dann für PHP, da PHP die Sprache ist, die uns am meisten bekannt war und am meisten gelegen hat. Des Weiteren haben wir uns für PHP entschieden, da es unserer Ansicht nach am besten umzusetzen war.
Zudem lässt sich PHP einfach und ohne großen Aufwand einrichten.
Außerdem ist ein weiterer Grund für PHP, dass bei der Programmierung in den meisten Fällen keine Rücksicht auf das Betriebssystem genommen werden muss, welches die Arbeit natürlich deutlich vereinfacht.

Das Frontend wurde mit Bootstrap (HTML, CSS Library) & JQuery (JavaScript Library) ergänzt. Wir haben uns für Bootstrap entschieden, da es für uns das beste Grid System enthält und uns damit bei der Gestaltung einiges an Arbeit erspart. JQuery wurde verwendet, um einen übersichtlicheren und simpleren Javascript Code schreiben zu können. Hierbei wurde uns auch einiges an Arbeit erspart z.B. durch die einfache POST & GET Integration von JQuery.
Zum Verschlüsseln der Texte des Messengers wurde die PHP Library Sodium verwendet. Dies ergab sich durch eine Recherche im Internet. Auf mehreren Internetseiten unter anderem auf Stack Overflow wurde die Library als einfachste Integration von der RSA Verschlüsselung in PHP genannt. Wir haben uns die Library angesehen und festgestellt, dass es genau dem entspricht, was wir uns vorgestellt haben. Sodium generiert automatisch einen Private und Public Key die bei der RSA Verschlüsselung benötigt werden, was bei den meisten anderen RSA Integrationen nicht der Fall ist. Zudem ist es bereits in PHP integriert und muss nicht nachinstalliert werden. Daher war es für uns eine logische Wahl.

Als Datenbank System kommt MariaDB zum Einsatz. Dies ist ein Fork von MYSQL dem weltweit am meisten verbreiteten relationalen Datenbankverwaltungssystem. Dies verwenden wir hauptsächlich da es Open Source ist und uns kostenlos zur Verfügung steht. Zudem unterstützt es die Datenbanksprache SQL fast zu 100% und bietet da durch große Flexibilität.

Betrachtet man es allgemeiner haben wir auf eine relationale Datenbank gesetzt, da es die einzige Datenbank Variante ist, mit der wir uns in der Gruppe auskennen und es somit eine nahliegende Entscheidung war.



Hier der Entwurf der Softwarearchitektur

Bei der Entwicklung eines „sicheren“ Messengers steht natürlich die Sicherheit im Vordergrund. Für unser Projekt mussten wir uns Gedanken darüber machen, wie wir mit nicht allzu komplizierten Komponenten und in einer relativ kurzen Zeit einen Messenger auf die Beine stellen können der eine hohe Sicherheit bietet. Wir mussten uns überlegen welche Verschlüsselungsmethoden für uns in Frage kämen, da die Texte ja nicht nur verschlüsselt werden sollten sondern vom User auch entschlüsselt werden können, ohne dass ein dritter diese Möglichkeit hat. Neben der Sicherheit muss die ganze Anwendung natürlich, auch einfach zu bedienen sein und übersichtlich gestaltet werden.

Während der Arbeit wurden natürlich, auch Ideen verworfen. Ursprünglich hab es die Idee, dass nicht nur Text Nachrichten versendet werden können, sondern auch Bilder oder andere Dateien. Während der Arbeit haben wir jedoch gemerkt, dass wir dies Zeitlich zu einer hohen Wahrscheinlichkeit nicht schaffen würden und haben uns daher erstmal nur auf Text Nachrichten beschränkt. Auch für die Umsetzung gab es ursprünglich eine andere Idee. So sollte, dass versenden und empfangen von

Nachrichten über Websockets geschehen. Auch diese Idee wurde verworfen, weil keiner aus der Gruppe vorher mit Websockets gearbeitet hatte und wieder unter Berücksichtigung der Zeit kamen wir zu dem Entschluss, dass wir mit Ajax Abfragen mehr oder weniger den, selben Effekt erzielen, ohne uns etwas Neues aneignen zu müssen. Die Performance wird dadurch zwar schlechter, was jedoch erst bei einer großen Menge an Usern eine wirkliche Auswirkung hätte. Diesen Nachteil haben wir in Kauf genommen. Zudem gab es auch die Überlegung das Projekt in Python zu programmieren. Auch hier haben wir jedoch festgestellt, dass unser Kenntnisstand es uns nicht ermöglichen würde in der uns zur Verfügung stehenden Zeit das Projekt so umzusetzen, wie wir es uns gewünscht haben. Daher setzen wir letzten Endes auf PHP. Einen wirklichen Nachteil gibt es hier nicht, außer dass man mit Python auf mehr Library hätte zurückgreifen können. Da wir aber auch in PHP die Library hatten die wir benötigten sahen wir keine weiteren Probleme.

Darstellung fachlicher und technischer Zusammenhänge

Die wichtigste Komponente der Anwendung ist der Chat.
Der User kann einen anderen User als Freund hinzufügen.
Bestätigt der andere User diese Anfrage, können beide einen Chat starten.
Wenn dies getan wird, wird in der Datenbank unter der Tabelle chat ein neuer Eintrag generiert, der eine Chat ID enthält und die beiden User ID's. Anhand der Chat ID werden später die Nachrichten die versendet werden dem richtigen Chat zugeordnet. Hat man einen Chat gestartet, wird man zum entsprechenden Chat Fenster weitergeleitet. Hier kann man nun Nachrichten verschicken und empfangen. Wird eine Nachricht verschickt, wird diese im Backend mit dem öffentlichen Schlüssel beider User verschlüsselt und erst dann in der Datenbank gespeichert, damit dritte nicht mitlesen können. Wenn die Nachricht gelesen werden soll, wird der Datenbank Eintrag geladen und erst auf dem Rechner, des Users wieder entschlüsselt. Dies geschieht mit dem Privaten Schlüssel des Users, der im lokalen Speicher des Browsers gespeichert ist. Serverseitig kann die Nachricht also nicht entschlüsselt werden! Zudem wird die Nachricht, nach dem sie gelesen wurde restlos aus der Datenbank entfernt und es kann nicht nachvollzogen werden, dass überhaupt eine Nachricht gesendet oder empfangen wurde.

Wie in unserem UML Diagramm zu sehen ist, besteht unsere Anwendung aus vier Klassen.

User: Die User Klasse ist die wichtigste. Sie besteht aus einer ID die fortlaufend generiert wird, einem Usernamen, einem Passwort und dem öffentlichen Schlüssel des Users. Diese Klasse ist von keiner anderen Klasse abhängig.

Friend: Die Klasse Friend verwaltet die Kontaktliste und Freundschaftsanfragen des Users. In ihr werden eine ID die fortlaufend generiert wird, zwei User ID's, ein Status (Status der Freundschaftsanfrage) und ein Timestamp (Zeit der Versendung der Anfrage) gespeichert. Diese Klasse ist abhängig von der User Klasse, da sie ohne die ID's der User nicht funktionieren kann.

Chat: Die Chat Klasse verwaltet die Chats. In ihr werden eine ID die fortlaufend generiert wird und zwei User ID's gespeichert. Sie befindet sich in Abhängigkeit zur User Klasse, wegen den User ID's und in einer Abhängigkeit zur Friend Klasse, da kein Chat gestartet werden kann ohne das die beiden User befreundet sind. Die Abhängigkeit zur Friend Klasse, kann jedoch auch ohne Probleme ausgebaut werden, falls die Einschränkung nicht benötigt werden sollte.
Message: Zur guten Letzt gibt es die Message Klasse. In ihr werden eine ID die fortlaufend generiert wird, eine Chat ID, eine User ID, ein Timestamp (Zeit der versendeten Nachricht), ein Status (sichtbare oder unsichtbare Nachricht), und zwei Nachrichten. Zwei Nachrichten, weil die versendete Nachricht einmal mit dem öffentlichen Schlüssel des Senders und einmal mit dem öffentlichen Schlüssel des Empfängers gespeichert wird, damit beide sie lesen können. Die Klasse Message befindet sich in Abhängigkeit zur Klasse Chat & User, da sie von beiden Klassen eine ID benötigt. Beide Klassen sind zwingend notwendig und können nicht ausgebaut werden.

```php
//Verschlüsseln
public static function Encrypt($text, $value) {
    global $user;

    $value = base64_decode($value);

    $encrypted = sodium_crypto_box_seal(
        $text,
        $value
    );

    return $encrypted;
}

//Entschlüsseln
public static function Decrypt($text, $key) {

   $text = base64_decode($text);
   $key = base64_decode($key);

    $decrypted = sodium_crypto_box_seal_open(
        $text,
        $key
    );

    return $decrypted;
}
```

Encrypt: Mit der Funktion Encrypt wird die Nachricht verschlüsselt. Hierfür werden der Funktion der Text und die öffentliche Schlüssel des Users übergeben. Der öffentliche Schlüssel kommt im Base64 Format muss daher erstmal wieder in seine normale Form konvertiert werden. Anschließend werden Text und Schlüssel der Funktion sodium_crypto_box_seal der Sodium Libary übergeben. Hier erfolgt die Verschlüsselung und der Verschlüsselte Text wird von der Funktion als Wert zurückgegeben.

Decrypt: Mit der Funktion Decrypt werden die Nachrichten wieder entschlüsselt um gelesen werden zu können. Übergeben wird hier der verschlüsselte Text und private Schlüssel des Users. Beide kommen im Base64 Format und müssen daher erstmal konvertiert werden. Anschließend wird es in die Funktion sodium_crypto_box_seal_open aus der Sodium Libary übergeben. Diese Funktion entschlüsselt den Text und gibt diesen als Wert zurück.

```php
$message1 = Message::Encrypt($text, $row->pubkey);
$message2 = Message::Encrypt($text, $user->pubkey);


Message::Add($chat->id, 1, base64_encode($message1), base64_encode($message2));
```

Hier wird zuerst eine Variable definiert mit dem Namen message1. In dieser Variable wird die zuvor erläutete Funktion Encrypt aufgerufen. Übergeben werden der Text und der Öffentliche Schlüssel des Empfänger Users, welche aus der Datenbank geladen werden.
Anschließend wird das selbe noch einmal für den Versender der Nachricht gemacht.

Im letzten Schritt wird die Funktion Add der Message Klasse aufgerufen, mit dem Parameter ChatID, Status, und die beiden Nachrichten. Diese Funktion schreibt die übergebenen Parameter anschließend in die Datenbank Tabelle chat_message. Die zuvor Verschlüsselten Nachrichten werden im Base64 Format übergeben, da ansonsten Probleme mit den verwendeten Zeichen auftreten könnten.

```
localStorage.setItem('<?php echo $user->username ?>', data.key);
```

LokalStorage ist ein Lokaler Speicher des Browsers. Mit der Funktion setItem können dort Daten abgelegt werden. Wir nutzen diese Funktion um den privaten Schlüssel des Users zu speichern. Hierzu übergeben wir den Namen des Users als Name des Speichers um diesen anschließend wieder zu finden und den Schlüssel selbst. Die Daten des lokalen Speichers sind sowohl an Domain als auch an Protokoll (http/https) der Seite gebunden und können daher auf anderen Seite nicht aufgerufen werden.

```
mykey = localStorage.getItem('<?php echo $user->username ?>');
```

Mit localStorage wird wieder der lokale Speicher des Browsers angesprochen. Mit der Funktion getItem können zuvor gespeicherte Daten abgerufen werden. Wir nutzen es in diesem Fall, um den privaten Schlüssel des Users aufzurufen. Als Parameter muss der Name des Speichers angegeben werden, was in unserem Fall der Username ist. Dieser wird aus der Datenbank geladen.

Kontrolle / Test

Die Anwendung wurde von jedem Mitglied aus der Gruppe getestet. Getestet wurde indem, wir die Anwendung aus der Sicht eines Users genutzt haben und somit sehen konnten, wo noch nachgebessert werden muss und was bereits funktioniert. Auch haben wir einige Fälle nachgespielt, die auftreten könnten und geschaut, wie sich unsere Anwendung in diesen Fällen verhält. Programmierte Testvarianten kommen bei uns nicht zum Einsatz, da wir in dem Bereich keine Erfahrung haben und uns somit nicht auskennen.

Den Großteil der gewünschten Anforderungen wurden von uns umgesetzt. Auch wenn wir bei bestimmten Funktionen einige Kompromisse eingegangen sind. Diese Kompromisse sind in den vorherigen Beschreibungen zu entnehmen. Schlussendlich

haben wir, jedoch eine Anwendung, die dem vorher definierten Anforderungen entspricht.

Review des Produkts

Zum Abschluss des Projektes haben wir unsere Anwendung in einem Git Repository gespeichert und diese Dokumentation verfasst. Beides werden wir dem Fachlehrer zukommen lassen.

Während des Projektes entstand natürlich auch die Idee für ein Folgeprojekt, welches nach Abschluss dieses Projekts umgesetzt werden könnte. Die Idee ist es unsere Anwendung, die aktuell nur auf dem PC in einem Webbrowser läuft auch auf Smartphones anzubieten. Dies würde in Form einer App geschehen, die Entwickelt werden müsste. Außerdem wäre es möglich die aktuelle Anwendung zu erweitern z.B. eine Funktion für das Versenden von Dateien und Bildern einzufügen oder die Funktion von Sprachnachrichten oder ähnlichem.

Eine Anwenderdokumentation wird in Form einer Präsentation gestaltet und der Klasse vorgestellt.

Fazit

Alles in allem verlief das Projekt ohne größere Probleme. Am schwierigsten fiel und das Dranbleiben. Umso länger man am Projekt gearbeitet hat umso mehr hat die Konzentration nachgelassen. Der größte Erfolg des Projektes ist natürlich die fertige Anwendung und das diese Anwendung so wie gewünscht umgesetzt werden konnte und funktioniert.

Für das nächste Mal würden wir uns die Arbeit besser aufteilen und z.B. die Dokumentation bereits neben der Projektarbeit schreiben und nicht erst im Anschluss, wenn kaum Konzentration vorhanden ist.