

Dokumentation zur schulischen Projektarbeit

**SagMa**

Anwendung zum Austauschen von Nachrichten

Abgabedatum : Frankfurt, den 18.05.2016

**Projektmitarbeiter:**

Tobias Stensbeck

Steven Pohl

Kevin Kahlsdorf

**Inhaltsverzeichnis:**

[1 Einleitung 3](#_Toc451201657)

[1.1 Projektumfeld 3](#_Toc451201658)

[1.2 Projektziel 3](#_Toc451201659)

[1.3 Projektbegründung 3](#_Toc451201660)

[1.4 Projektschnittstellen 3](#_Toc451201661)

[2 Projektplanung 3](#_Toc451201662)

[2.1 Projektphasen 3](#_Toc451201663)

[2.2 Abweichungen vom Pflichtenheft 4](#_Toc451201664)

[2.3 Ressourcenplanung 4](#_Toc451201665)

[2.4 Entwicklungsprozess 4](#_Toc451201666)

[3 Entwurfsphase 5](#_Toc451201667)

[3.1 Zielplattform 5](#_Toc451201668)

[3.2 Entwurf der Benutzeroberfläche 5](#_Toc451201669)

[3.3 Geschäftslogik 5](#_Toc451201670)

[3.4 Maßnahmen zur Qualitätssicherung 6](#_Toc451201671)

[4 Implementierungsphase 7](#_Toc451201672)

[4.1 Implementierung der Benutzeroberfläche 7](#_Toc451201673)

[4.2 Implementierung der Geschäftslogik 8](#_Toc451201674)

[5 Fazit 14](#_Toc451201675)

1. Einleitung
   1. Projektumfeld

* Das Projekt ist ein Teil der schulischen Ausbildung für Fachinformatiker Anwendungsentwicklung und ist daher für alle Azubis obligatorisch und wird im Auftrag des verantwortlichen Lehrers ausgeführt.
  1. Projektziel
* Bestand des Projektes ist es, den Austausch von Nachrichten zwischen verschiedenen Benutzern zu ermöglichen.
* Ein Benutzer soll sich anmelden können, um daraufhin mit anderen aktiven Benutzern Nachrichten austauschen zu können. Des Weiteren kann ein Benutzer andere Benutzer in eine Gruppe einladen und alle eingeladenen Benutzer können dann über einen Gruppenchat gleichzeitig miteinander kommunizieren.
  1. Projektbegründung
* Dieses Projekt beruht nicht auf einem wirtschaftlichen Interesse für keinen der Beteiligten.
* Der Sinn dieses Projekts ist die Vorbereitung auf die spätere Abschlussprüfung, in welcher ebenfalls alle Bestandteile, wie in diesem Projekt, gefordert werden.
* Die Projektmitarbeiter (hier: die Schüler) sollen so ein Gefühl für eine Projektarbeit bekommen und zur Kenntnis nehmen, welche Bereiche zu einer entsprechenden Aufgabe gehören.
  1. Projektschnittstellen
* Das Programm zum Austauschen von Nachrichten ist im derzeitigen Umfang nicht an andere Systeme gebunden. Es existieren also keine technischen Schnittstellen.
* Das Projekt wurde, nach Vorlage des Pflichtenhefts(siehe Anhang), durch Herrn Eick genehmigt und wird vor der Klasse FI4D präsentiert.

1. Projektplanung
   1. Projektphasen

* Die Umsetzung des Projekts erfolgte im Zeitraum vom 25.11.2015 - 18.05.2016.

Allerdings lag der Fokus der Arbeit am Projekt auf der Zeit in der Berufsschule und daher im Zeitraum vom 01.02.2016 – 26.02.2016 sowie vom 09.05.2016 – 11.05.2016.

* 1. Abweichungen vom Pflichtenheft
* Änderung der Produktumgebung:

Ein Zugriff auf eine MySQL bzw. Microsoft SQL Server Datenbank ist nicht notwendig, da keine Datenbank zum Speichern der Passwörter verwendet wird. Zum Projektstart war dies noch nicht ins Detail geklärt und es wurden zwei Möglichkeiten zum Abspeichern der Passwörter in Betracht gezogen. Die erste Möglichkeit war das Abspeichern in der Datenbank, die zweite das Abspeichern in einer Datei.

Gemeinsam wurde entschieden keine Datenbank zu verwenden, da der Aufwand nicht im Verhältnis zum Nutzen steht. Es gibt nur eine geringe Zahl an Benutzern und daher lohnt sich der Einsatz einer Datenbank nicht. Als Folge wurde die Umsetzung mit dem Abspeichern in einer Datei gewählt.

Eine Implementierung mit Hilfe einer Datenbank wäre trotzdem noch möglich, dazu müsste lediglich eine Klasse für den Datenbankzugriff entwickelt werden, welche das Interface AuthStore implementiert. Es handelte sich also nicht um eine kritische Entscheidung sondern es wurde die Dateivariante gewählt, da sie für den derzeitigen Verwendungszweck am Effizientesten war.

Wenn benötigt, wäre die Umstellung auf eine Datenbank zum Abspeichern möglich.

* 1. Ressourcenplanung
* Für die Umsetzung des Projektes wurden folgende Ressourcen verwendet :

1. Arbeitsplatz incl. PC, Tastatur, Maus und Internetzugang

2. Eclipse als Integrated Development Environment (IDE)

3. Java Runtime Enviroment 8 oder neuer

4. Git & GitHub für die Versionsverwaltung

* 1. Entwicklungsprozess
* Aufteilung der verschiedenen Aufgaben innerhalb des Projektteams, sowie ständige interne Kommunikation in Form von Gesprächen und der Nutzung von WhatsApp (Anlegen einer entsprechenden Gruppe).
* Aufteilung der Aufgaben wie folgt :

Erstellung der serverseitigen Anwendungen : Tobias Stensbeck.

Erstellung der Benutzeroberfläche(mit allen Funktionalitäten) : Steven Pohl.

Erstellung der Dokumentation, sowie des Authentifizierungsprozesses: Kevin Kahlsdorf.

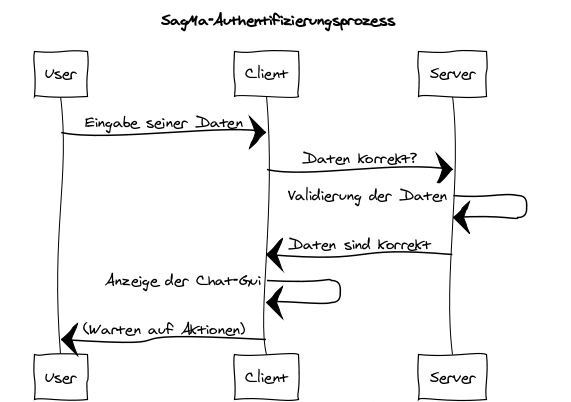
1. Entwurfsphase
   1. Zielplattform

* Die Auswahl der Zielplattform fand im Einklang unter den Projektmitarbeitern statt. Es wurde diskutiert, was benötigt wird bzw. ob etwas umsetzbar sei. Als Ergebnis wurde sich auf die Verwendung der Programmiersprache JAVA festgelegt, da diese alle geforderten Bedingungen erfüllt (Server, Client, GUI, etc.) und das Wissen der Projektmitarbeiter hier bereits sehr fortgeschritten ist.
  1. Entwurf der Benutzeroberfläche
* Das Wichtigste bei der Erstellung der GUI war die Übersichtlichkeit, wobei jedoch gleichzeitig ein angenehmes Umfeld geschaffen werden sollte. Die Funktionalitäten sollten leicht erkennbar und somit leicht bedienbar sein.

Die Bedienung soll ersichtlich und ohne Hinzunahme einer Bedienungsanleitung gewährleistet werden.

Um dies zu gewährleisten wurden gemeinsam einige Grundstrukturen festgelegt:

* + Eine Menüleiste am oberen Rand.
  + Ein Fenster zum Anzeigen der gesendeten und empfangenen Nachrichten am linken Rand gehalten, aber als größtes Feld.
  + Ein Fenster zur Eingabe von Nachrichten, sowie ein Button zum Abschicken unterhalb des Fensters mit den empfangenen Nachrichten.
  + Eine Übersicht der Chats an der rechten Seite
  + Eine Übersicht der angemeldeten Benutzer und Gruppen an der rechten Seite unterhalb des Fensters für die Übersicht der Chats.
  1. Geschäftslogik
* Bei Start der Anwendung wird der Benutzer aufgefordert einen Server, sowie Benutzername und Passwort einzugeben. Werden gültige Daten eingegeben, wird dem Benutzer die GUI (siehe 3.2) angezeigt und ihm stehen die Funktionalitäten zur Verfügung.
* Sequenzdiagramm, am Beispiel der Anmeldung (vereinfacht, keine Methodennamen zur besseren Verständlichkeit):



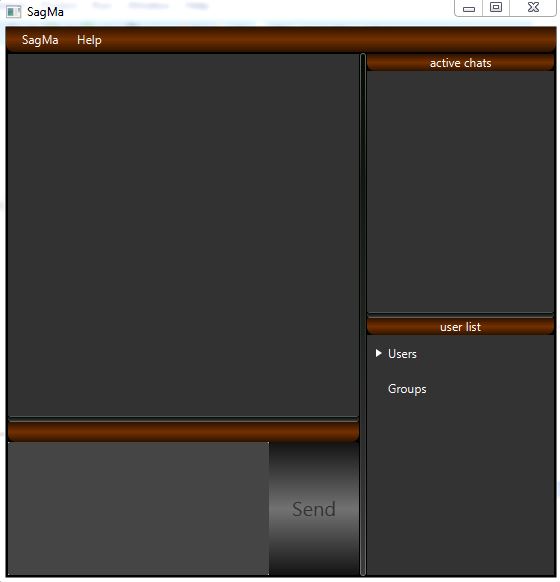
* Dies ist natürlich nur ein kleiner Ausschnitt, aber diese grundsätzliche Funktionsweise lässt sich auf weitere Funktionen übertragen. Der Benutzer gibt etwas ein oder wählt eine Funktion, der Client kommuniziert mit dem Server und führt die Aktion entsprechend aus.
  1. Maßnahmen zur Qualitätssicherung
* Um die Qualität des Produkts zu sichern wurden mehrere Testarten verwendet:

1. Kontinuierliches manuelles Testen während der Entwicklung.
2. Anwendertests mit den folgenden Testfällen, alle erfolgreich :
3. Registrierung eines Benutzers
4. Anzeige anderer aktiver Benutzer
5. Versenden einer Nachricht an einen anderen Benutzer
6. Empfangen von Nachrichten
7. Erstellung von Gruppen
8. Hinzufügen von Gruppenmitgliedern
9. Schließen des Fensters -> Logout
10. Speichern der Registrierungsdaten in einer Datei
11. Authentifikation eines Benutzers (anmelden, nicht registrieren)
12. Implementierungsphase
    1. Implementierung der Benutzeroberfläche

* Bei der Implementierung der Benutzeroberfläche wurde eng am Entwurf festgehalten und sich auf die Funktionalität und Übersichtlichkeit fokussiert.

Außerdem kann der Benutzer die Breite der einzelnen Fenster seinen Wünschen anpassen.

Screenshot der Benutzeroberfläche nach erfolgtem Einloggen/Registrieren:



Menüleiste

Übersicht der aktiven Chats

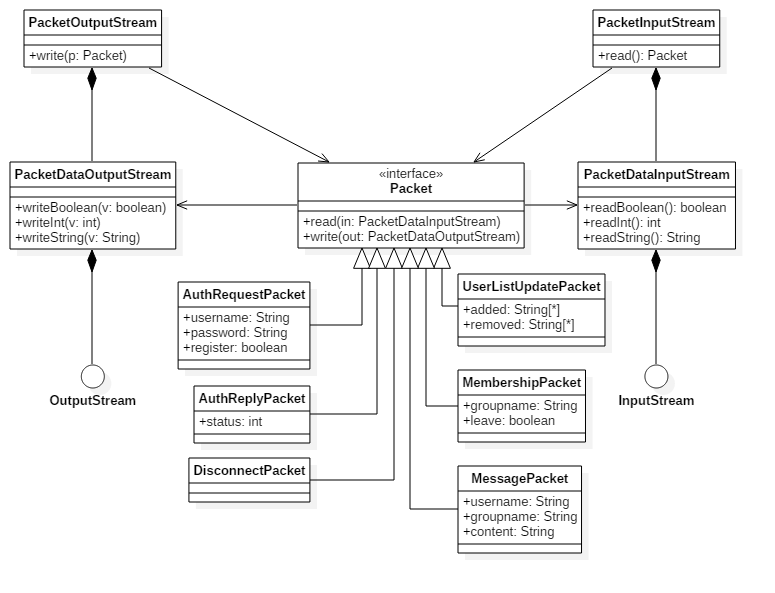
Übersicht der angemeldeten User

Übersicht der existierenden Gruppen.

Feld zum Versenden von Nachrichten

* 1. Implementierung der Geschäftslogik
* Die Paketschicht:

Java's Socket API bietet die Möglichkeit Daten mittels IOStreams über ein Netzwerk zu verschicken. IOStreams arbeiten mit einzelnen Bytes, die meisten Anwendungen - SagMa eingeschlossen - arbeiten allerdings auf einem höheren Level. Die Paketschicht stellt die Schnittstelle zwischen den Bytes die über das Netzwerk verschickt werden und anwendungsspezifischen DTOs (Data Transfer Objects), sogenannten Paketen, dar.

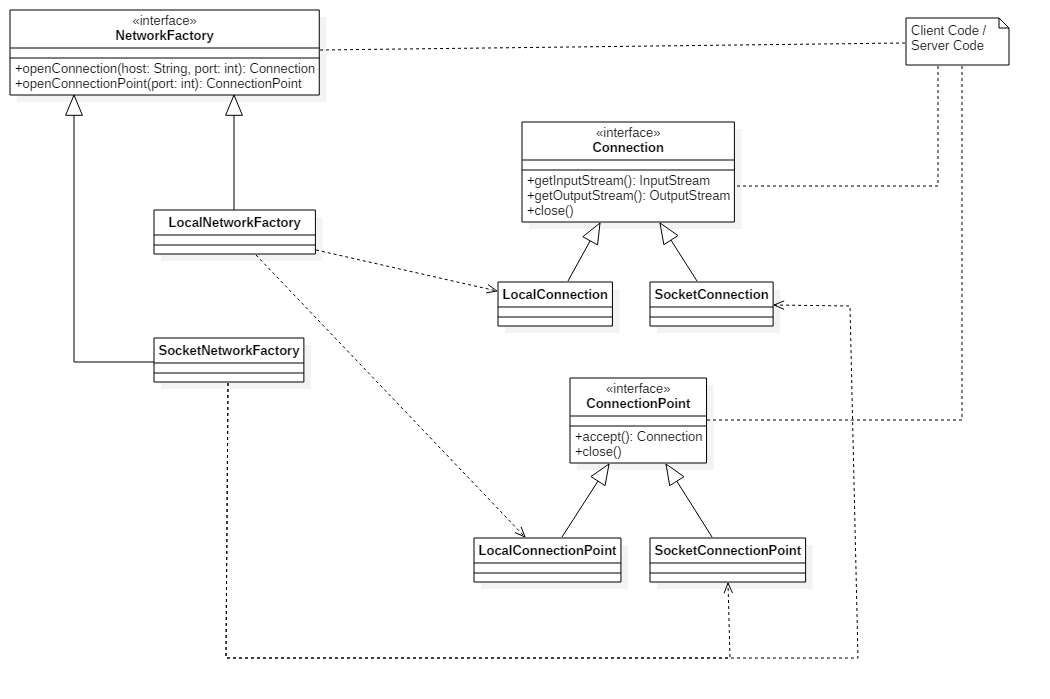


Im Zentrum der Paketschicht steht das Packet-Interface. Die Anwendung definiert verschiedene Paketklassen, die das Packet-Interface implementieren. Dazu muss eine read() und eine write()-Methode definiert werden, die einen PacketDataInputStream bzw. PacketDataOutputStream erhalten. Die Daten-Streams sind Wrapper für Input- bzw. OutputStreams, die es erlauben alle primitiven Datentypen von Java sowie Strings und String-Arrays zu serialisieren (im UML-Diagramm wurden beispielhaft drei Methoden angegeben, die anderen wurden der Kürze halber nicht aufgeführt). Die Daten-Streams machen das Implementieren des Packet-Interfaces trivial.

In der Geschäftslogik der Anwendung können nun verschiedene Paketklassen instanziiert und mit Hilfe des PacketOutputStreams verschickt werden. Auf der Empfängerseite können diese per PacketInputStream empfangen werden. Die Paket-Streams verwenden Reflection und die read() und write()-Methoden des Packet-Interfaces, um sicherzustellen, dass der PacketInputStream eine exakte Kopie des Objekts zurückgibt, welches der PacketOutputStream serialisiert hat.

* Die Netzwerkschicht

Während der Entwicklung ist das Problem aufgetreten, dass ein Arbeitsrechner mit hohen Sicherheitseinstellungen das Java Socket API blockiert hat und dadurch das Ausführen und Testen der Anwendung nicht mehr möglich war. Um dieses Problem zu umgehen wurde die Netzwerkschicht aufgebaut, die von der tatsächlich verwendeten Netzwerktechnologie abstrahiert und eine lokale Implementation entwickelt, die das Java Socket API nicht benötigt.



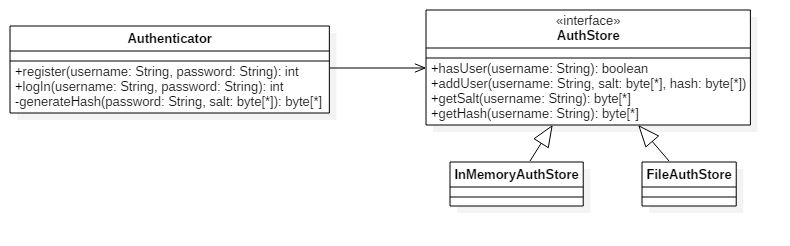
Die Netzwerkschicht folgt dem Entwurfsmuster der abstrakten Fabrik. Der Chatserver möchte auf Verbindungen warten. Diese Funktionalität wird durch das Interface ConnectionPoint bereitgestellt. Wenn eine neue Verbindung bereit ist, wird ein Objekt erzeugt, dass das Connection-Interface implementiert. Dieses erlaubt die Kommunikation über zwei IOStreams. Der Chatclient benutzt ebenfalls das Connection-Interface um mit dem Chatserver zu kommunizieren. Um die entsprechenden Connection- und ConnectionPoint-Objekte zu erzeugen, rufen sowohl der Client als auch der Server Methoden auf einer statischen Variable vom Typ NetworkFactory auf. NetworkFactory ist selbst wieder ein Interface, das von den Klassen LocalNetworkFactory und SocketNetworkFactory implementiert wird. Diese Klassen erzeugen jeweils Objekte vom Typ LocalConnection bzw. LocalConnectionPoint und SocketConnection bzw. SocketConnectionPoint.

Je nachdem, ob die statische Variable auf eine LocalNetworkFactory oder eine SocketNetworkFactory zeigt, werden immer nur lokale Connection-Objekte oder nur socketbasierte Connection-Objekte erzeugt.

Die Input- und OutputStreams der lokalen Connection-Objekte sind direkt miteinander verknüpft. Deshalb wird die Socket API nicht benötigt, es können aber nur Verbindungen innerhalb der gleichen virtuellen Maschine von Java erstellt werden. Dies ist für den normalen Gebrauch natürlich nicht geeignet, zum Testen muss aber nur eine lokale NetworkFactory eingerichtet und dann Server und Client in der gleichen JVM gestartet werden.

* Das Authentifizierungssystem

Das Authentifizierungssystem überprüft und speichert Passwörter. Um die Sicherheit der Benutzer zu gewährleisten werden die Passwörter zusammen mit einem Salt gehasht und nur der Hashwert abgespeichert.



Die Klasse Authenticator stellt den Einstiegspunkt zu dem Authentifizierungssystem dar und ist für die Überprüfung der Passwörter inklusive Hashgenerierung verantwortlich. Die Speicherung der Hashwerte wird an das Interface AuthStore delegiert.

Authenticator stellt die Methoden register() und logIn() zur Verfügung, die jeweils einen Benutzernamen und ein Passwort annehmen und einen Statuscode als Ganzzahl zurückgeben (da dieser Statuscode serialisiert werden soll, wurden hier Ganzzahlkonstanten einem Enum vorgezogen).

Zur Hashwertbildung wurde die Bouncy Castle Crypto API eingebunden. Bouncy Castle ist eine bekannte Open Source Implementierung von Verschlüsselungsalgorithmen in Java und C#. Bei kryptografischen Algorithmen ist es vorzuziehen eine öffentliche Implementation einzubinden, anstatt den Algorithmus selber zu programmieren, da so sichergestellt ist, dass in der Implementation keine Schwachstellen vorhanden sind, da der Source Code von vielen verschiedenen Programmierern und Experten eingesehen und kontrolliert werden kann.

Es wird der SCrypt-Algorithmus verwendet, um die Hashwerte zu berechnen. Dieser Algorithmus wird zur Berechnung von Hashwerten aus Passwörtern empfohlen, da sowohl Rechenaufwand als auch Speicherverbrauch angepasst werden können. Je höher diese Anforderungen sind, desto schwieriger wird es eine Brute-Force-Attacke durchzuführen. Gleichzeitig wird ein Wörterbuchangriff durch die Verwendung von Salts verhindert. Die Parameter N, R und P für den SCrypt-Algorithmus wurden nach öffentlichen Empfehlungen gewählt. Eine schnelle Zeitmessung ergab, dass eine Hashwertberechnung ca. 100 ms dauert. Das ist für den gedachten Gebrauch als Chatserver, an dem sich nicht sehr oft angemeldet wird, absolut ausreichend. Für eine Brute-Force-Attacke stellt diese Zahl allerdings eine große Herausforderung dar, da auf einem Prozessor nur ca. 10 Versuche pro Sekunde durchgeführt werden können, während bei Passwörtern, die mit einer weniger aufwändigen Hashfunktion verschlüsselt wurden, oft tausende Versuche pro Sekunde durchgeführt werden können.

public class Authenticator {

private final static int SALT\_BYTES = 32;

private final static int HASH\_BYTES = 128;

private final static int SCRYPT\_N = 16\_384; // 2 ^ 14

private final static int SCRYPT\_R = 8;

private final static int SCRYPT\_P = 1;

private final AuthStore store = new FileAuthStore("sagma.auth");

public int register(String username, String password) {

if (store.hasUser(username))

return STATUS\_USERNAME\_TAKEN;

if (password.isEmpty())

return STATUS\_INVALID\_PASSWORD;

SecureRandom random = new SecureRandom();

byte[] salt = random.generateSeed(SALT\_BYTES);

byte[] hash = generateHash(password, salt);

store.addUser(username, salt, hash);

return STATUS\_OK;

}

public int logIn(String username, String password) {

if (!store.hasUser(username))

return STATUS\_INVALID\_CREDENTIALS;

byte[] salt = store.getSalt(username);

byte[] expected = store.getHash(username);

byte[] hash = generateHash(password, salt);

return Arrays.equals(hash, expected) ? STATUS\_OK : STATUS\_INVALID\_CREDENTIALS;

}

private byte[] generateHash(String password, byte[] salt) {

try {

byte[] pw = password.getBytes("UTF-8");

return SCrypt.generate(pw, salt, SCRYPT\_N, SCRYPT\_R, SCRYPT\_P, HASH\_BYTES);

} catch (UnsupportedEncodingException e) {

// UTF-8 must be supported

throw new RuntimeException("UTF-8 not supported.", e);

}

}

}

Das AuthStore-Interface bietet Methoden um Benutzern Salts und Hashwerte zuzuordnen und wieder abzufragen. Falls eine andere Speichermethode, z. B. eine Datenbank, hinzugefügt werden soll, muss nur dieses Interface implementiert werden. Die in-memory Implementierung ist trivial.

public interface AuthStore {

/\*\*

Checks whether a record for the given username exists.

\*

@param username not null

\*/

boolean hasUser(String username);

/\*\*

Adds a new record to the auth store. If a record with the given username already exists, it will be overwritten.

\*

@param username not null

@param salt not null

@param hash not null

\*/

void addUser(String username, byte[] salt, byte[] hash);

/\*\*

Returns the salt associated with the username.

\*

@param username not null

@return the salt for the user or null if no record was found

\*/

byte[] getSalt(String username);

/\*\*

Returns the hash associated with the username.

\*

@param username not null

@return the hash for the user or null if no record was found

\*/

byte[] getHash(String username);

}

public class InMemoryAuthStore implements AuthStore {

protected final Map<String, byte[]> salts = new HashMap<>();

protected final Map<String, byte[]> hashes = new HashMap<>();

@Override

public boolean hasUser(String username) {

return salts.containsKey(username);

}

@Override

public void addUser(String username, byte[] salt, byte[] hash) {

salts.put(username, salt);

hashes.put(username, hash);

}

@Override

public byte[] getSalt(String username) {

return salts.get(username);

}

@Override

public byte[] getHash(String username) {

return hashes.get(username);

}

}

Die Klasse FileAuthStore speichert die Hashwerte in einer Textdatei. Der Einfachheit halber wurde FileAuthStore als Unterklasse von InMemoryAuthStore angelegt, da die Funktionalität des Abfragens identisch ist. Nur im Konstruktor müssen die Werte aus der Datei geladen werden (falls vorhanden) und wenn ein neuer Wert eingefügt wird, dann sollen alle Werte in die Datei exportiert werden.

Die Byte-Arrays werden mittels zweier Hilfsmethoden als hexadezimaler Text abgespeichert.

public class FileAuthStore extends InMemoryAuthStore {

private final String filename;

public FileAuthStore(String filename) {

this.filename = filename;

try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {

String line;

while ((line = reader.readLine()) != null) {

if (line.isEmpty())

continue;

String[] parts = line.split("\t");

if (parts.length != 3) // ignore malformed lines

continue;

byte[] salt = parseHexadecimal(parts[1]);

byte[] hash = parseHexadecimal(parts[2]);

if (salt != null && hash != null)

super.addUser(parts[0], salt, hash);

}

} catch (IOException ignored) {}

}

@Override

public void addUser(String username, byte[] salt, byte[] hash) {

super.addUser(username, salt, hash);

try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename))) {

Set<String> users = hashes.keySet();

for (String user : users) {

writer.write(user);

writer.write("\t");

writer.write(toHexadecimal(getSalt(user)));

writer.write("\t");

writer.write(toHexadecimal(getHash(user)));

writer.newLine();

}

} catch (IOException ignored) {}

}

private static final String HEX\_CHARS = "0123456789abcdef";

private byte[] parseHexadecimal(String text) {

if (!text.matches("(?:[0-9a-fA-F]{2})\*"))

return null;

text = text.toLowerCase();

byte[] values = new byte[text.length() / 2];

for (int i = 0; i < values.length; i++) {

char highChar = text.charAt(2 \* i + 0);

char lowChar = text.charAt(2 \* i + 1);

int high = HEX\_CHARS.indexOf(highChar);

int low = HEX\_CHARS.indexOf(lowChar);

values[i] = (byte) ((high << 4) | (low));

}

return values;

}

private String toHexadecimal(byte[] values) {

StringBuilder result = new StringBuilder(2 \* values.length);

for (byte byteValue : values) {

int intValue = Byte.toUnsignedInt(byteValue);

int high = intValue >> 4;

int low = intValue & 0xf;

result.append(HEX\_CHARS.charAt(high));

result.append(HEX\_CHARS.charAt(low));

}

return result.toString();

}

}

1. Fazit

Als Abschluss kann man sagen, dass das Projekt in Bezug auf Umfang und Komplexität für den Zeitraum angemessen war. Alle Musskriterien aus dem Pflichtenheft wurden erfolgreich in das Produkt integriert.

Das Projekt nahm etwas mehr Zeit in Anspruch als in der Berufsschule zur Verfügung stand, dies war aber zu Beginn des Projekts bereits ersichtlich. Entgegen unserer Erwartungen, war es uns in diesem Zeitraum jedoch nicht möglich die Wunschkriterien zu implementieren, da unser Fokus auf der Funktionalität der Musskriterien lag, nach dem Motto: „Lieber etwas weniger, dafür funktioniert es“.

Anhang

A1 : Pflichtenheft

SagMa - Pflichtenheft

1. Zielbestimmung

1.1 Musskriterien

Die Software soll den Austausch von Textnachrichten zwischen Individuen oder in Gruppen ermöglichen. Die Identifikation der Benutzer erfolgt über selbstgewählte Benutzernamen. Vor der Benutzung der Hauptfunktionalität muss sich ein Benutzer mit seinem Benutzernamen und Passwort identifizieren.

1.2 Wunschkriterien

Die Texteingabe und -darstellung kann so erweitert werden, dass einfache Formatierungen wie z. B. fetter oder kursiver Text und andere Textfarben möglich sind.

Das Versenden von Bildern und Dateien kann ermöglicht werden.

1.3 Abgrenzungskriterien

Der Chatverlauf soll nicht gespeichert werden.

Es soll nur ein Server verwendet werden. D. h. alle Benutzer, die miteinander kommunizieren wollen, müssen mit dem gleichen Server verbunden sein.

Eine Peer-to-Peer Kommunikation der Clients ist ebenfalls nicht vorgesehen.

2. Produktumgebung

Die Software wird in der Programmiersprache Java geschrieben.

Der Client benötigt ein Java Runtime Environment (JRE) in der Version 8 oder neuer.

Der Server benötigt ebenfalls eine JRE in der Version 8 oder neuer sowie Zugriff auf eine MySQL bzw. Microsoft SQL Server Datenbank.

3. Produktfunktionen

Man kann die IP-Adresse eines Servers angeben und sich mit Benutzername und Passwort anmelden.

Anschließend sieht man eine Liste mit vorhanden Gruppen und angemeldeten Benutzern. Wird eine Gruppe oder ein Benutzer ausgewählt, öffnet sich ein Chatfenster. Chatfenster von Gruppen zeigen alle Teilnehmer der Gruppe an.

4. Entwicklungsumgebung

Zur Entwicklung der Software wird Eclipse als Integrated Development Environment (IDE), JUnit als Testframework, EclEmma als Code Coverage Tool und Git & GitHub zur Versionsverwaltung verwendet.