Berner Fachhochschule
Technik und Informatik

Realisierung eines Chat Systems mit Java

Kurs	SRU2-BTI7051-15/16
Gruppe	Gruppe 1
	Aulbach Adrian, Egger Samuel, Fernandez Bruno
Status	Abgeschlossen

Änderungsverzeichnis

Datum	Version	Änderung	Autor
19.01.2016	0.1	Ausgangslage, Anforderungen und Systemarchitektur (Server)	Egger Samuel
20.01.2016	0.2	Benutzeranleitung	Fernandez Bruno
20.01.2016	0.3	Systemübersicht	Fernandez Bruno
21.01.2016	0.4	Realisierung	Egger Samuel
21.01.2016	1.0	Systemarchitektur (Client)	Aulbach Adrian

Inhaltsverzeichnis

1 Ausgangslage

Im Rahmen dieser Projektarbeit soll eine Chat-Lösung mit Java realisiert werden. Da für diesen Auftrag keine konkreten Anforderungen seitens Auftraggeber festgelegt wurden, sollen die Rahmenbedingungen mit den nachfolgenden Anforderungen grob abgesteckt werden. Die Anforderungen sind so formuliert, dass die Implementation der Lösung genügend Spielraum lässt.

2 Anforderungen

1 Funktionale Anforderungen

1. Das Chat-System MUSS die Möglichkeit bieten, sich mit einem Benutzernamen anzumelden.
2. Das Chat-System MUSS die Möglichkeit bieten, allen angemeldeten Benutzern eine Nachricht zu verschicken.
3. Das Chat-System MUSS die Möglichkeit bieten, sich abzumelden (entgegen der Variante, dass das Programme einfach geschlossen wird, ohne die Verbindung sauber zu schliessen).
4. Das Chat-System SOLL jedem Client erlauben, anzuzeigen, wer alles online ist.

2 Qualitätsanforderungen

1. Das Chat-System darf nicht zusammenbrechen wenn ein Client zu einem beliebigen Zeitpunkt nicht mehr erreichbar ist / die Verbindung vom Client unterbrochen wird (Bsp. Netzwerkfehler).
2. Ein Client wird automatisch abgemeldet sobald er länger als 3 Minuten (temporäre Verbindungsunterbrüche) nicht mehr online / erreichbar ist.
3. Zum Anmelden ist keine Registrierung erforderlich. Bei jedem Anmeldevorgang soll ein beliebiger Benutzername gewählt werden können.

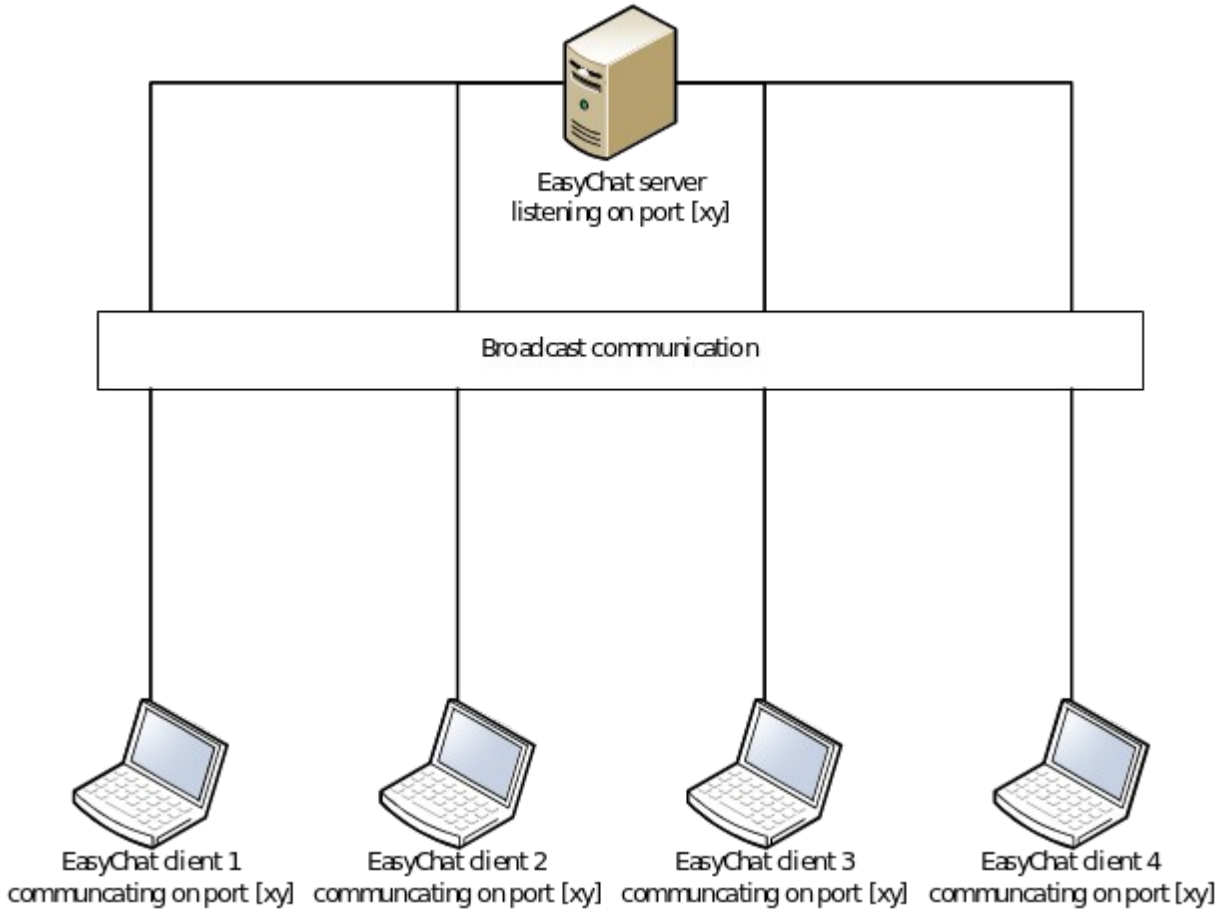
3 Konzept

In der EasyChat Architekturzeichnung sind die folgenden Komponenten ersichtlich:

- EasyChat Server
- EasyChat Clients

Die Menge der Clients in der Zeichnung wurde auf vier beschränkt. Der EasyChat Server erlaubt jedoch viel mehr Verbindungen zu als auf der Zeichnung ersichtlich sind. Dies dient also nur der Veranschaulichung.

Der Versand der Nachrichten geschieht als Broadcast. Dies bedeutet, dass jede verschickte Nachricht auch an jeden angemeldeten Client verschickt wird.



Der Client wird so konzipiert, dass er möglichst einfach in einem privaten Netzwerk integriert werden kann. Aus diesem Grund wird in dieser Implementierung auf gewisse Überprüfungen verzichtet. Es wird beispielsweise nicht überprüft ob der eingegebene Username bereits verwendet wird oder nicht. Eine Login Validierung ist auch nicht vorgesehen. Es soll sich jeder User, am besten mit dem Eigenen Namen oder persönlichen Nick anmelden können.

Die angemeldet Clients müssen nicht zwingend ersichtlich sein. Nur beim Schreiben einer Nachricht muss ersichtlich sein, wer sie geschrieben hat. Der Server kann auf einem x-beliebigen System im Netzwerk laufen. Auch der Port soll frei wählbar sein. Diese Informationen müssen jedoch jedem Client bekannt sein damit sie sich am Server anmelden können.

Ziel dieses Projektes soll sein, ein kostengünstiger Chat-Client für Teams zu realisieren, die beispielsweise in einem Grossraumbüro arbeiten und sich kurz und knackig Informationen wie Links, Infos, usw. weiter leiten wollen.

4 Systemarchitektur

3 Server-Client Kommunikation

Damit Client und Server miteinander kommunizieren können, war es notwendig, ein Protokoll respektive einen Vertrag zwischen Server und Client zu definieren. Der Datenaustausch erfolgt via TCP/IP über eine UTF-8 codierte JSON[1] Objekte. Damit sowohl für den Client als auch den Server klar ist, wann eine Nachricht vollständig empfangen wurde, wird nach jedem JSON Objekt ein Zero-Byte gesendet.

Das folgende Sequenzdiagramm soll den logischen Ablauf der Kommunikation zwischen Client und Server verdeutlichen:

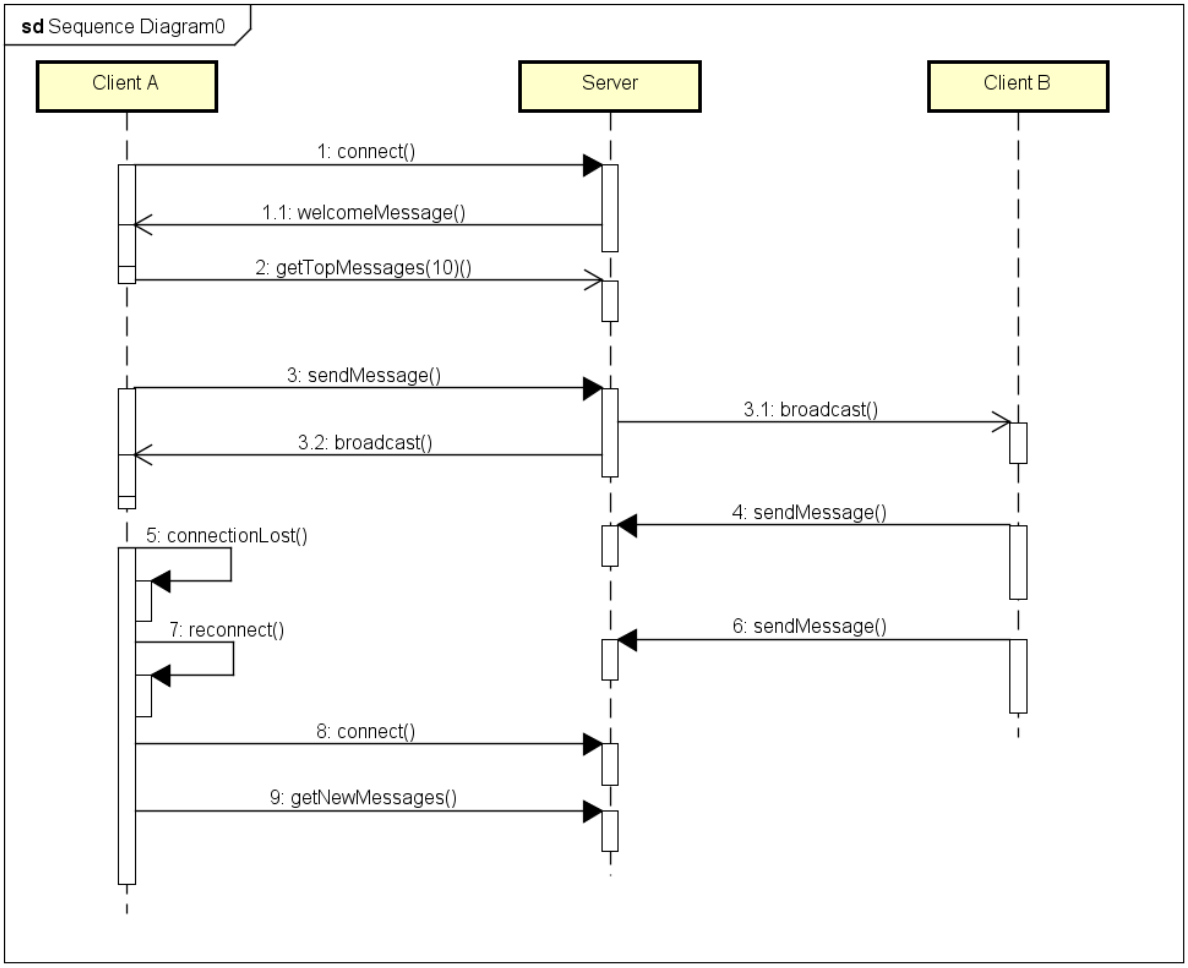


Abbildung 1: Kommunikation zwischen Client und Server

1. Client A verbindet sich mit dem Server nachdem sich bereits andere Clients (B) angemeldet hatten. Der Server sendet dem Client eine Willkommensnachricht: *{“message“:“Willkommen im EasyChat“,“sender“:“server“}*
2. Client A fordert die letzten 10 Nachrichten an, die bereits an alle anderen angemeldeten Clients versendet wurden: *{“top“:10}*
3. Client A sendet eine Nachricht, die an alle anderen Clients versendet werden soll: *{“message“:“<Nachricht>“,“sender“:“<Benutzername>“}*

1 Siehe http://www.json.org/

Der Server leitet diese Nachricht an alle Clients weiter:
{“message“:“<Nachricht>“,“sender“:“<Benutzername>“, “id“:<messageID>}

4. Client B sendet eine Nachricht.
5. Beim Client A passiert wegen eines Netzwerkfehlers ein Verbindungsunterbruch.
6. Client B sendet nochmals eine Nachricht.
7. Client A versucht, die Verbindung wieder herzustellen.
8. Verbindung zum Server wurde erfolgreich hergestellt.
9. Client A erkundigt sich beim Server nach allen verpassten Nachrichten seit seiner letzten empfangenen Nachricht. Der Client speichert dazu intern immer die ID der zuletzt empfangenen Nachricht ab.
   {“until“:<zuletzt empfangene messageID>}
   Der Server sendet maximal 100 Nachrichten um zu vermeiden, dass eine sehr grosse Anzahl an Nachrichten abgefragt wird.

4 Server Architektur

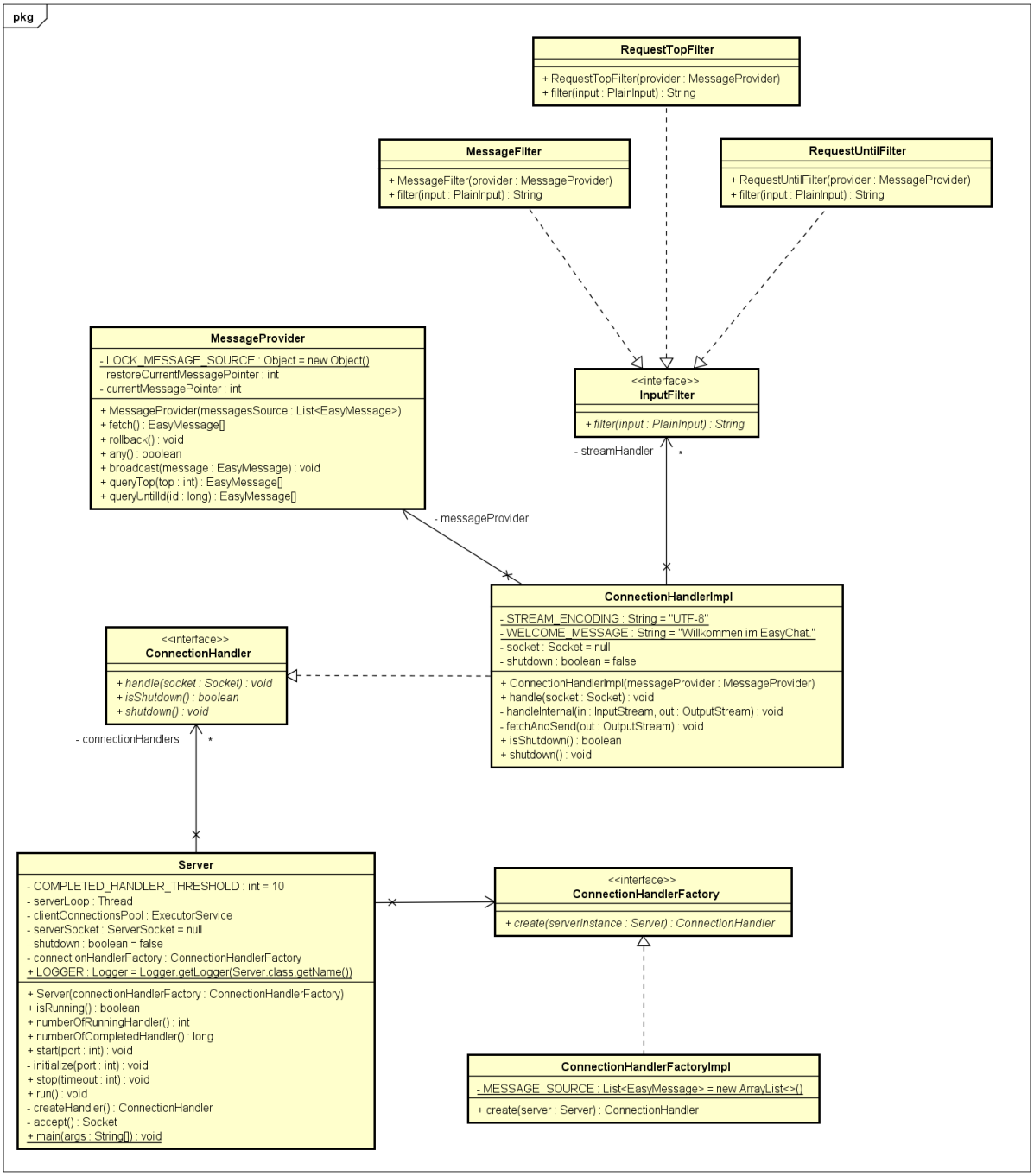


Abbildung 2: Klassendiagramm Server

1 Klasse Server

Die Klasse Server ist der Einstiegspunkt des gesamten Serverprojekts. Die Klasse implementiert die „Server Loop“, welche eingehende Verbindungsanfragen von Clients entgegen nimmt. Der Server hört standardmässig auf den Port 5200.

2 Klasse ConnectionHandlerFactory (-Imp) und ConnectionHandler (-Impl)

Der Server übergibt jede angenommene Verbindung zu einem Client an einen separaten Thread, um die „Server-Loop“ nicht zu blockieren. Die Logik zum Kommunizieren mit dem

Client befindet sich in der Klasse ConnectionHandlerImpl, die das Interface ConnectionHandler implementiert. Der Server verwendet zum Instanziieren von einem neuen ConnectionHandler eine Factory (ConnectionHandlerFactory) um die Kopplung zwischen „Server-Loop“ und „Kommunikationsprotokoll“ möglichst zu entkoppeln.

3 Klasse MessageProvider

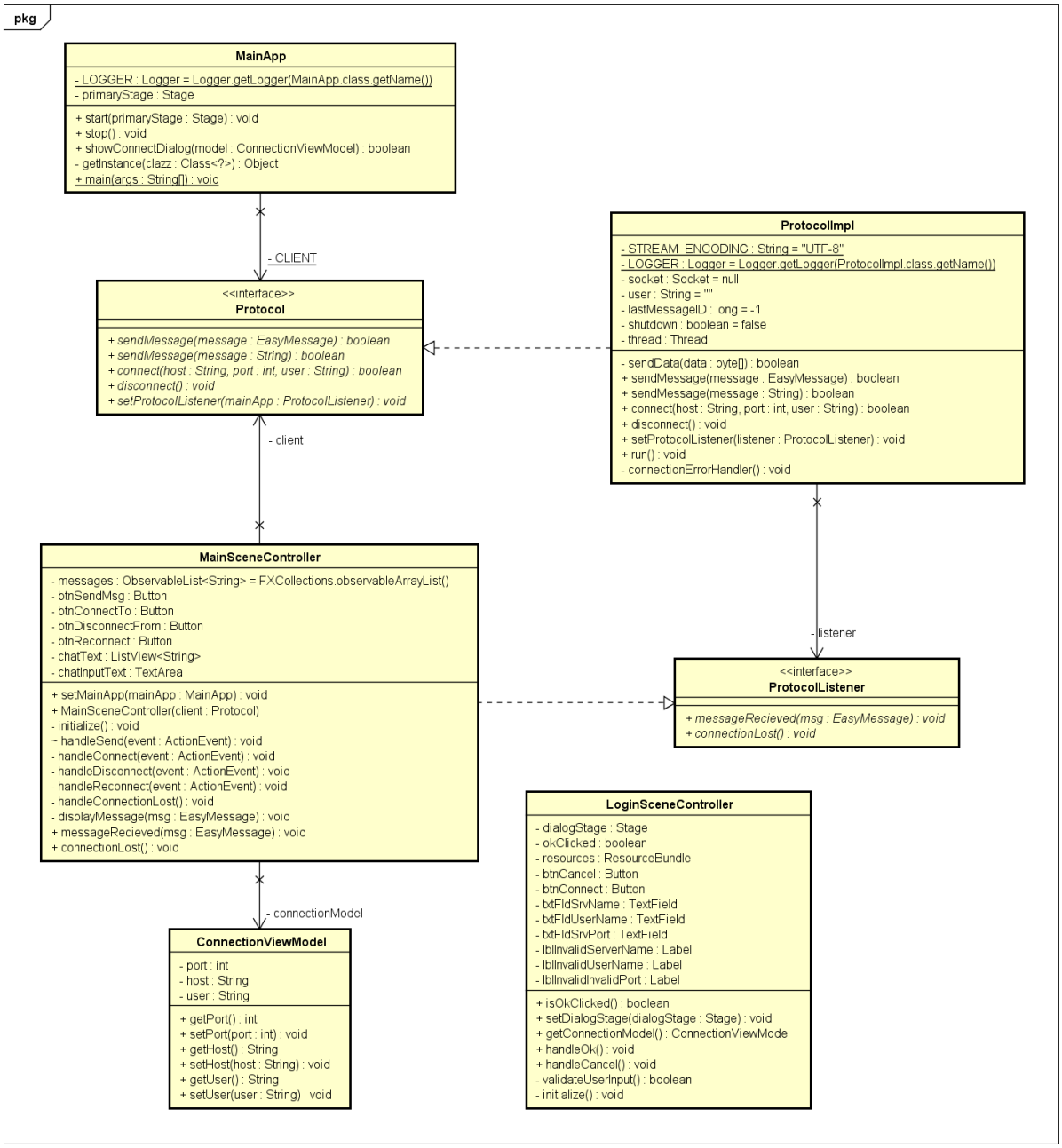
Die Klasse MessageProvider kapselt die globale Speicherquelle aller Nachrichten und erlaubt das Queuen von Nachrichten zum Versenden und die Abfrage der eingegangenen Nachrichten. Aktuell werden die Nachrichten nur im RAM gespeichert d.h. sobald der Server beendet wird, gehen alle Nachrichten verloren. Die Entkopplung erlaubt es aber, in einer zweiten Iteration ohne weitere Probleme eine persistente Speicherung der Nachrichten (z.B: in eine Datenbank) zu realisieren.

4 Klasse InputFilter

Die InputFilter werden verwendet, um unabhängig voneinander auf eingehende Client-Anfragen zu reagieren. Um Beispielsweise die Abfrage aller angemeldeten Benutzer zu ermöglichen, genügt es einen neuen InputFilter zu implementieren und diesen in der Klasse ConnectionHandlerImpl zu registrieren.

5 Client Architektur

Abbildung 3: Klassendiagramm Client



1 Klasse MainApp

Die Klasse MainApp ist der Einstiegspunkt für das FXML-GUI. Hier wird das GUI initialisiert.

2 Klasse MainSceneController

Die Klasse MainSceneController handelt die Interaktionen des Users im GUI ab. Sie übergibt Nachrichten des Users an die Methode sendMessage der Klasse ProtocolImpl und nimmt über die Methode messageRecieved Nachrichten von ProtocolImpl entgegen und zeigt diese im GUI an. Über die Methode connectionLost kann sie zudem von der Klasse ProtocolImpl benachrichtigt werden, wenn die Verbindung zum Server unterbrochen wurde, worauf dem User ein Button zum wiederherstellen der Verbindung angezeigt wird.

3 Klasse ProtocolImpl

In der Klasse ProtocolImpl ist die Kommunikation zwischen Client und Server implementiert. Sie nimmt Nachrichten aus dem MainSceneController entgegen und benachrichtigt diesen, wenn eine neue Nachricht vom Server eintrifft oder die Verbindung abbricht.

Von jeder eintreffenden Nachricht wird die ID gespeichert, so dass diese dem Server beim erneuten Verbindungsaufbau nach einem Verbindungsunterbruch mitgeteilt werden kann und der Server die in der Zwischenzeit versendeten Nachrichten an den Client senden kann. Wenn der Client geschlossen wird, geht die ID der letzten Nachricht verloren und vom Server werden nach einem Neustart nur die letzten 10 Nachrichten angefordert. Da wir die Daten als JSON formatieren, könnte jedoch der Client später um eine Methode erweitert werden, die beim Start die Nachrichtung der vorherigen Sitzung aus einer Datei liest und gleich abhandelt, wie wenn sie vom Server kämen. So wäre auch die ID der letzten erhaltenen Nachricht gesetzt, so dass der Server alle fehlenden Nachrichten senden könnte.

5 Projekt Kompilieren

Zum Kompilieren des Projekts sind folgende Voraussetzungen zu beachten:

- JRE und JDK 1.8 (http://www.oracle.com/technetwork/java/javase/downloads/index.html)
- Apache Maven 3.3.x (https://maven.apache.org/)

Eine Anleitung zum Installieren und Verwenden der jeweiligen Tools kann von der Herstellerseite entnommen werden. Jedes Projekt verfügt über eine pom.xml Datei im Hauptverzeichnis, welche von Maven benötigt wird.

Das Projekt besteht aus insgesamt drei einzelnen Projekten:

- EasyChatClient
    - JavaFX Benutzerschnittstelle
    - Klassen zum Kommunizieren mit dem Server
- EasyChatCommon
    - Enthält gemeinsame Klassen d.h. Klassen die sowohl vom Client als auch vom Server benötigt werden.
- EasyChatServer
    - Kommandozeilenapplikation zum Ausführen des Servers
    - Implementations des Chat Servers

Damit der Client respektive der Server kompiliert werden kann, muss zwingend das EasyChatCommon Projekt als erstes kompiliert werden.

6 Benutzeranleitung

1 Den Server Starten

Den Server auf der Kommandozeile mit

```
java -jar EasyChatServer-1.0-SNAPSHOT.jar [port]
```

ausführen. Die angabe des Ports ist optional, standardmässig wird der Port 5200 verwendet.

2 Mit dem Server verbinden

Damit der Client genutzt werden kann, muss zuerst eine Verbindung zu Server hergestellt werden. Hierfür wurde zuoberst im GUI ein „Connect“ Button (Nr. 1) platziert auf den geklickt werden muss.

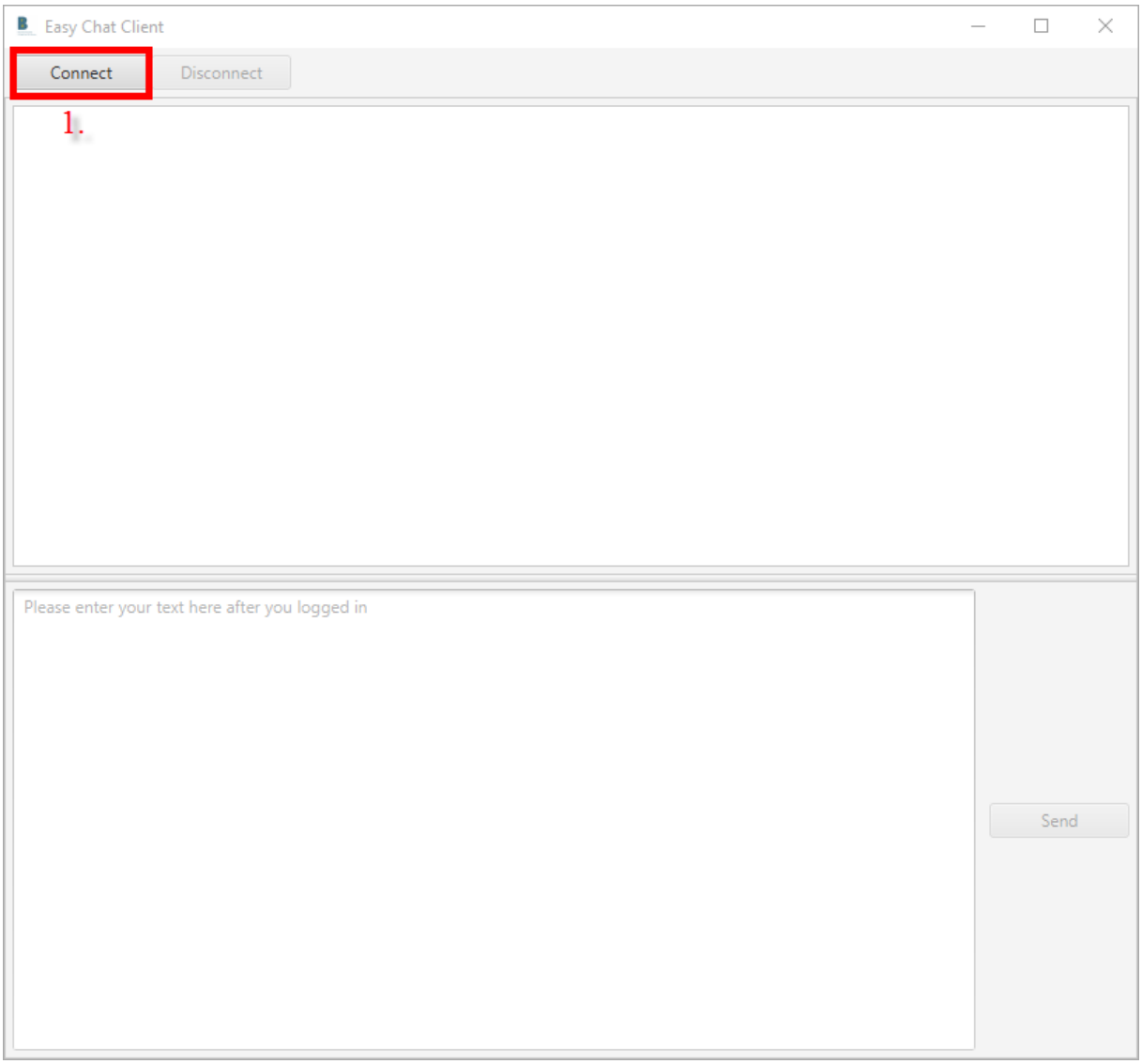


Abbildung 4: Chatfenster

Nachdem nun also auf dem Connect Button (Nr. 1) geklickt wurde, öffnet sich ein zweites Fenster indem man die Login Daten eingeben muss.

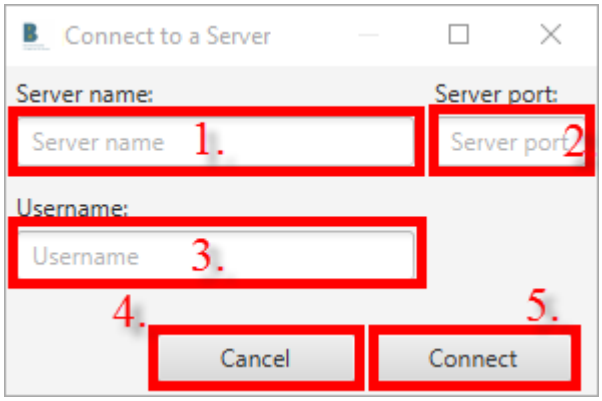


Abbildung 5: Anmeldedialog

Unter Server Name (Nr.1) muss der EasyChat Servername oder die IP Adresse eingegeben werden. Der Server Port kommt in das zweite Textfeld (Nr. 2). Diese Informationen müssen dem Anwender bekannt gegeben werden. Der User Name (Nr. 3) kann frei gewählt werden.

Achtung: Aktuell werden Namenskollisionen bei der Anmeldung mit einem bereits angemeldeten Namen nicht behandelt. Das kann zu unerwartetem Verhalten der Applikation führen.

Um sich nun mit dem Server zu verbinden, muss auf den Connect Button (Nr. 5) geklickt werden. Will man aus irgendeinem Grund das Login Fenster schliessen, kann auf den Cancel Button (Nr. 4) geklickt werden. Sollte sich der Client mit dem Server verbinden können, erscheint im Hauptfenster eine Meldung (Nr.1). Gleichzeitig werden die Buttons Disconnect (Nr. 2) und Send (Nr. 3) aktiviert.

Sollte sich der Client mit dem Server verbinden können, erscheint im Hauptfenster eine Meldung (Nr.1). Gleichzeitig werden die Buttons Disconnect (Nr. 2) und Send (Nr. 3) aktiviert.

Beim Login Fenster wird überprüft ob die Felder richtig befüllt wurden. Die Textfelder „Server Name“ und „Username“ dürfen beim Login nicht leer sein. Ansonsten wird die untenstehende Meldung geworfen. Beim „Server port“ wird zusätzlich noch überprüft, ob es sich bei der Eingabe und Zahlen handelt.

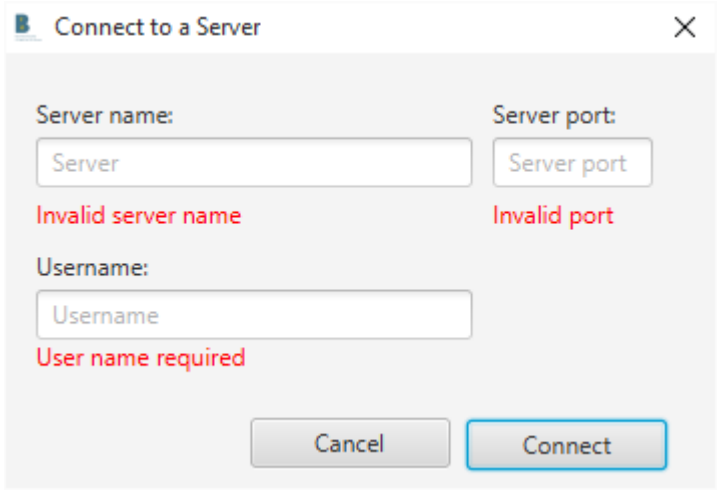


Abbildung 6: Anmeldedialog mit Fehler

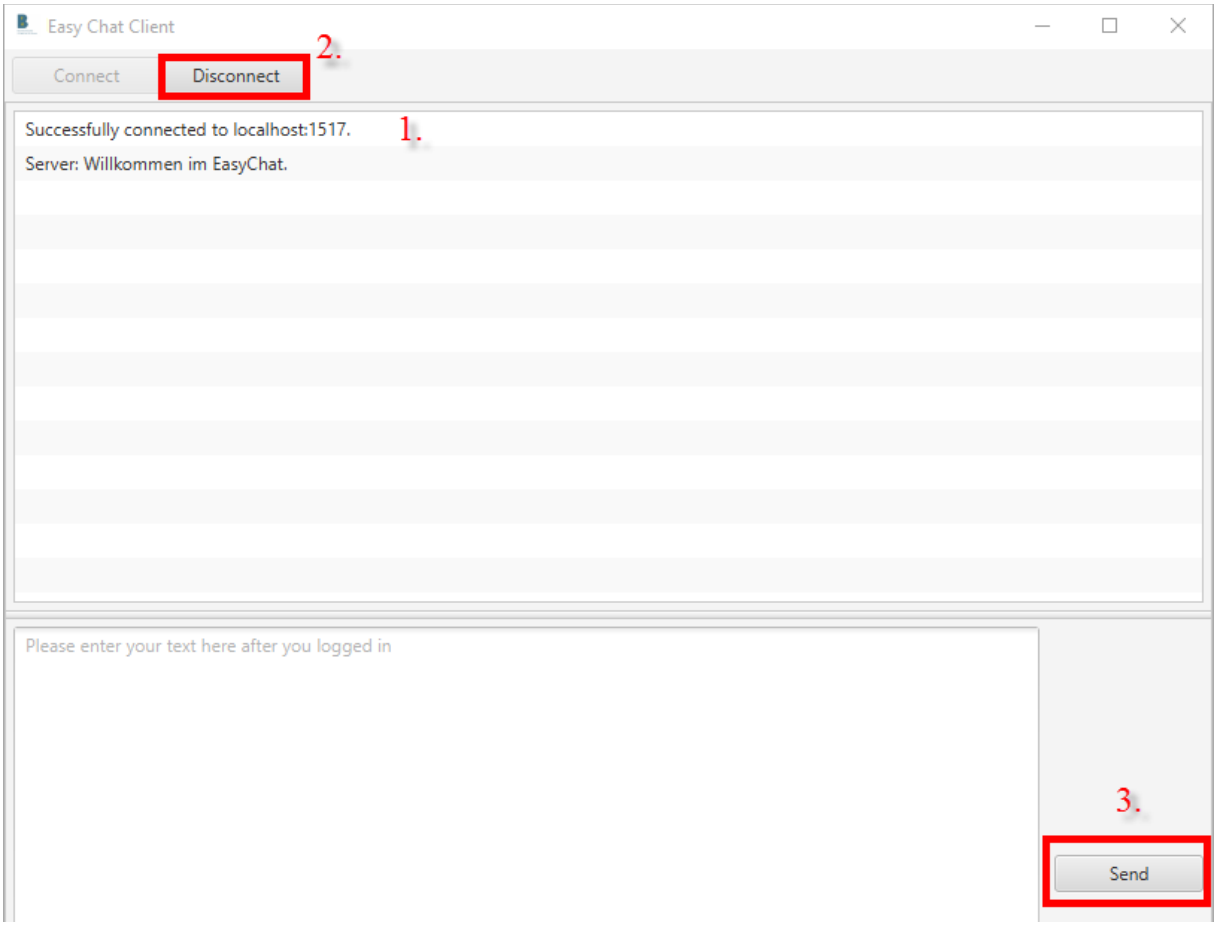


Abbildung 7: Chatfenster nach erfolgreichem Verbindungsaufbau

Wenn sich ein Client aus irgendeinem Grund nicht anmelden kann, erscheint die untenstehende Fehlermeldung (Nr. 1). Gründe für die erfolglose Verbindung können sein: falscher Hostname/IP, falscher Port, Server läuft nicht, Server kann nicht erreicht werden, usw.

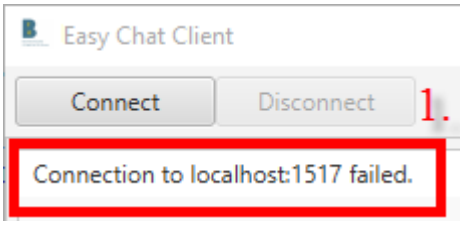


Abbildung 8: Meldung bei einem Verbindungsfehler

3 Nachrichten Senden

Nachdem man sich erfolgreich anmelden konnte, kann man sogleich mit dem Verfassen von Nachrichten beginnen. Im Textfeld (Nr. 2) muss die gewünschte Nachricht eingegeben werden. Damit diese dann verschickt wird, muss auf den Send Button (Nr.2) geklickt werden.

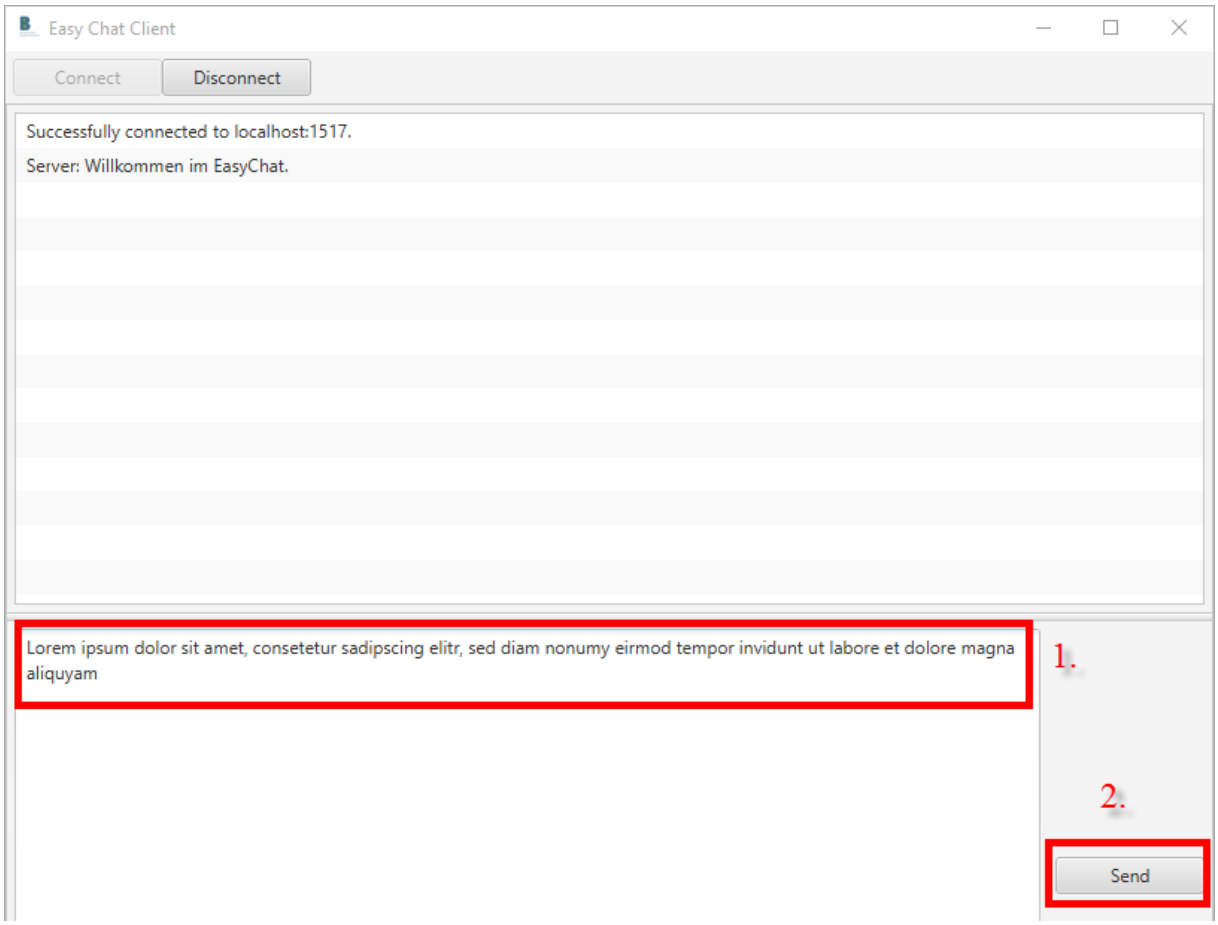


Abbildung 9: Nachricht versenden

Die versendete Nachricht wird im Chatverlauf (Nr. 1) im Format *[Me]: [Message]* angezeigt. Nachrichten von anderen Chatusern wird im Chatverlauf (Nr. 2) im Format *[Username]: [Message] angezeigt*

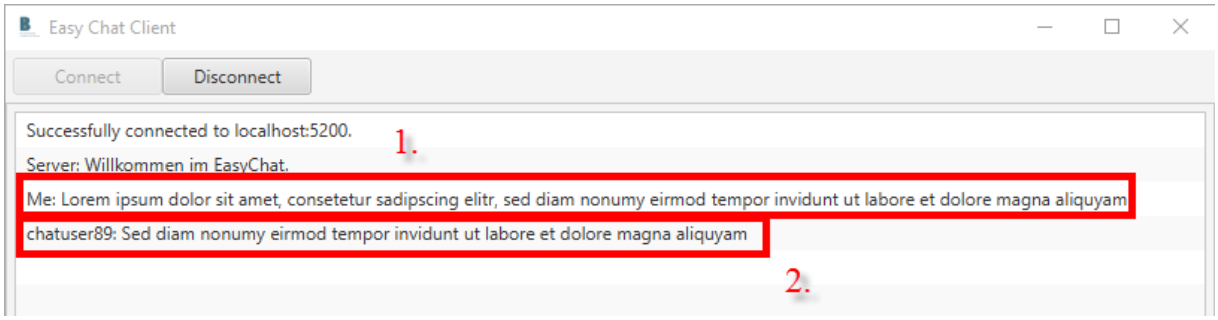


Abbildung 10: Anzeige einer versendeten Nachricht

4 Abmelden

Um sich von Server abzumelden, kann auf den Disconnect (Nr. 1) Button geklickt werden. Sobald dieser gedrückt wurde, wird der User von der Session getrennt, und es erscheint eine Meldung (Nr. 2) im Chatverlauf.

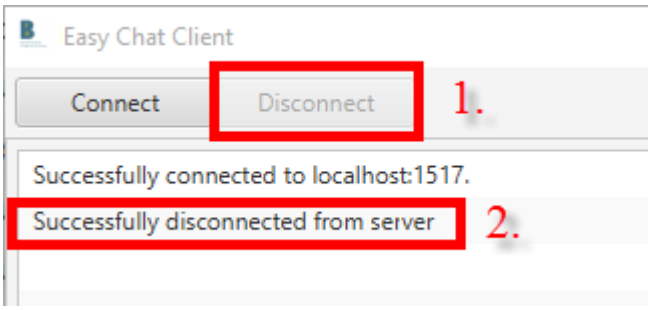


Abbildung 11: Abmelden vom Server

Jetzt besteht keine Verbindung mehr zum Server.