Fachhochschule Nordwestschweiz
Hochschule für Wirtschaft

AACSB ACCREDITED

Projektarbeit Software Engineering - Projekt 2

Extending the Chatserver

Arpad Broglie
David Thomi
Teodor Glisic

Olten, 29. November 2025

Olten

arpad.broglie@students.fhnw.ch
david.thomi@students.fhnw.ch
teodor.glisic@students.fhnw.ch
www.fhnw.ch/wirtschaft

Inhaltsverzeichnis

1 Einleitung

Dieses Projekt wurde im Rahmen der Vorlesung Software Engineering durchgeführt, mit dem Ziel, das Verständnis für Netzwerkkommunikation zu fördern. Unsere Projektgruppe hat sich für die Option 1 entschieden, den Server zu erweitern. Der grösste Teil der Arbeit, war die Chatroom-Funktionalität zu implementieren.

Anleitung: Chat-Server mit IntelliJ IDEA

1.1 1. Voraussetzungen

Folgende Komponenten müssen bereitgestellt sein:

- IntelliJ IDEA (Community oder Ultimate Edition).
- Java Development Kit (JDK): Mindestens Version 17 wird empfohlen.
- JSON-Bibliothek: Die Datei org.json (z. B. json-20240303.jar) muss heruntergeladen und lokal gespeichert werden (z. B. vom Maven Repository).

1.2 2. Projekt aus GitHub klonen

1. IntelliJ IDEA öffnen.
2. Auf "Get from VCS" (oder File > New > Project from Version Control) klicken.
3. Im Feld "URL" die Repository-Adresse eingeben: https://github.com/teodorglisic/sel_project_chatserver.git
4. Ein lokales Verzeichnis wählen und auf "Clone" klicken.
5. Das Projekt wird nun heruntergeladen und geöffnet.

1.3 3. Externe Bibliothek (org.json) einbinden

Da der Code die Bibliothek org.json benötigt, diese aber nicht automatisch geladen wird, muss sie den Projekt-Abhängigkeiten hinzugefügt werden:

1. Im Menü auf File > Project Structure (oder Strg+Alt+Shift+S) klicken.
2. Links im Bereich "Project Settings" den Punkt "Libraries" auswählen.
3. Auf das "+"-Symbol klicken und "Java" wählen.
4. Zum Speicherort der heruntergeladenen json-*.jar Datei navigieren, diese auswählen und mit OK bestätigen.
5. Im folgenden Dialog das Modul auswählen (meistens nur eines verfügbar) und mit OK bestätigen.
6. Das "Project Structure"-Fenster mit Apply und OK schließen.

1.4 4. Quellordner konfigurieren (falls nötig)

IntelliJ muss erkennen, wo der Quellcode liegt. Da das Paket chatroom direkt im Hauptverzeichnis liegt:

1. Sicherstellen, dass der Ordner chatroom im Projektbaum sichtbar ist.
2. Falls Java-Dateien rot markiert sind (weil Klassen nicht gefunden werden), muss eventuell das Hauptverzeichnis als "Sources Root" markiert werden:
3. Rechtsklick auf das Hauptverzeichnis des Projekts.

4. Mark Directory as > Sources Root.

1.5 5. Server starten

Der Server kann nun direkt über die IDE gestartet werden:

1. Im Projektbaum zur Datei chatroom/server/Server.java navigieren und diese öffnen.
2. Neben der main-Methode (Zeile 19) auf den grünen Play-Button (Pfeil) klicken.
3. "Run 'Server.main()'" auswählen.

In der Konsole unten sollte nun die Meldung erscheinen: Port is 50001 (oder ähnlich), was bestätigt, dass der Server läuft.

1.6 6. Verbindung testen

- IntelliJ verfügt über ein integriertes Terminal sowie einen HTTP-Client.
- Über das Terminal in IntelliJ: Unten auf den Reiter "Terminal" klicken und folgenden Befehl eingeben:

Code	**Erwartete Ausgabe**
curl http://127.0.0.1:50001/ping	{"ping":true}

2 Nutzung des Codes

2.1 Erstellung des Servers

Die Serveranwendung wurde in Java implementiert. Für die Datenübertragung wurde JSON verwendet. Für die Handhabung von HTTP-Anfragen wurde die vorhandene Handler Klassenhierarche erweitert.

2.2 Beispiele der Kommunikation mit dem Server

Um den Server zu testen, haben wir mit Postman gearbeitet. Die nachfolgende Tabelle beschreibt eine typische Kommunikation mit dem Server, um einen Chatroom zu nutzen.

Client sendet	**Server antwortet**

POST http://localhost:50001/user/register
{
"username":"AJB1234", "password":"123"
}
	{"username":"AJB1234"}

POST http://localhost:50001/user/login
{
"username":"AJB1234", "password":"123"
}
	{"token":"1463242A4CCFF2CEB252DD212F5FE8F9"}

POST http://localhost:50001/user/register
{
"username":"David", "password":"1234"
}
	{"username":"David"}

POST http://localhost:50001/user/login
{
"username":"David", "password":"1234"
}
	{"token":"B322087DC5CBB5932DF4D8A0FD2D5A9D"}

POST http://localhost:50001/chatroom/create
{
"username":"David", "token":"B322087DC5CBB5932DF4D8A0FD2D5A9D", "chatroomName": "AJDAVID"
}
	{"chatroomName":"AJDAVID"}

Client sendet	Server antwortet

POST http://localhost:50001/chatroom/join
{
"token":"1463242A4CCFF2CEB252DD212F5FE8F9", "chatroomName": "AJDAVID"
}
	{"joined":true}

POST http://localhost:50001/chatroom/send
{
"token":"1463242A4CCFF2CEB252DD212F5FE8F9", "chatroomName": "AJDAVID", "message":"Is anyone here?"
}
	{"sent":true}

POST http://localhost:50001/chat/poll
{
"token":"B322087DC5CBB5932DF4D8A0FD2D5A9D"
}
	{"messages":[{"chatroom":"AJDAVID","message":"Is anyone here?","username":"AJB1234"}]}

POST http://localhost:50001/chat/send
{
"token":"1463242A4CCFF2CEB252DD212F5FE8F9", "username":"David", "message":"Hello David this is a private message."
}
	{"send":true}

POST http://localhost:50001/chat/poll
{
"token":"B322087DC5CBB5932DF4D8A0FD2D5A9D"
}
	{"messages":[{"chatroom":"AJDAVID","message":"Hellooo","username":"AJB1234"},{"message":"Hello David this is a private message.","username":"AJB1234"}]}

POST http://localhost:50001/chatroom/users
{
"chatroomName":"AJDAVID"
}
	{"users":["David","AJB1234"]}

Client sendet	Server antwortet

POST http://localhost:50001/chatroom/leave
{
"to-ken":"1463242A4CCFF2CEB252DD212F5FE8F9", "chatroomName":"AJDAVID"
}
	{"left":true}

POST http://localhost:50001/chatroom/delete
{
"to-ken":"B322087DC5CBB5932DF4D8A0FD2D5A9D", "chatroomName":"AJDAVID"
}
	{"deleted":true}

GET http://localhost:50001/chatrooms	{"chatrooms":[]}

3 Codedesign

3.1 Erklärung Chatroom-Klasse

Die Klasse Chatroom repräsentiert einen Gruppenchat und fungiert als die zentrale Verwaltungsinstanz aller aktiven Chaträume. Der Server verwendet keine externe Datenbank, weshalb alle während der Laufzeit des Servers erstellten Chaträume in einer statischen Liste gespeichert werden.

Wichtige Methoden:

- getChatroomByCreator(String token): Liefert alle Chaträume, die vom Nutzer mit diesem Token erstellt wurden.
- deleteChatroom(Chatroom chatroomToDelete): Entfernt einen Chatroom aus der globalen Liste.
- findChatroomByName(String chatroomName): Sucht in der globalen Liste nach einem bestimmten Chatroom.
- addMember(Client client) / removeMember(Client client): Fügt Nutzer der Liste members hinzu oder entfernt sie.
- listChatrooms(): Gibt alle derzeit existierenden Chaträume zurück.

3.2 Erklärung ChatroomHandler Klasse

Die ChatroomHandler Klasse fungiert als Schnittstelle zwischen den HTTP Anfragen des Clients und der Datenstruktur des Servers. Die Klasse erbt von der abstrakten Klasse Handler. Der Chatroom-Handler nutzt die Funktion Switch-Case-Statement um die HTTP GET und POST Anfragen zu verarbeiten.

Endpoints	Erklärung
/chatrooms (GET)	Listet alle Chatrooms
/chatroom/create (POST)	Erstellt einen Chatroom
/chatroom/join (POST)	Einem Chatroom beitreten
/chatroom/leave (POST)	Einen Chatroom verlassen
/chatroom/delete (POST)	Einen Chatroom löschen (nur für den Ersteller des Raumes möglich)
/chatroom/users (POST)	Die User eines Chatrooms auflisten
/chatroom/send (POST)	Eine Nachricht an einen Chatroom senden

3.3 Erklärung der ChatHandler Klasse

Die ChatHandler Klasse erbt ebenfalls von Handler und ist für die Kommunikation zwischen Users zuständig. Die wichtigsten Methoden sind:

- sendMessage(…) Zuständig für das Versenden einer Nachricht mithilfe Authentifizierung (Client.findbyToken), Empängersuche (Client.findByUsername) und die eigentliche Zustellung (recipient.send())

- receiveMessage(…) Wird über den Endpoint /chat/poll aufgerufen. Nach der Identifikation des Nutzers per Token werden über client.getMessages() alle in der Warteschlange des Nutzers gespeicherten Nachrichten an den Client zurückgesendet.

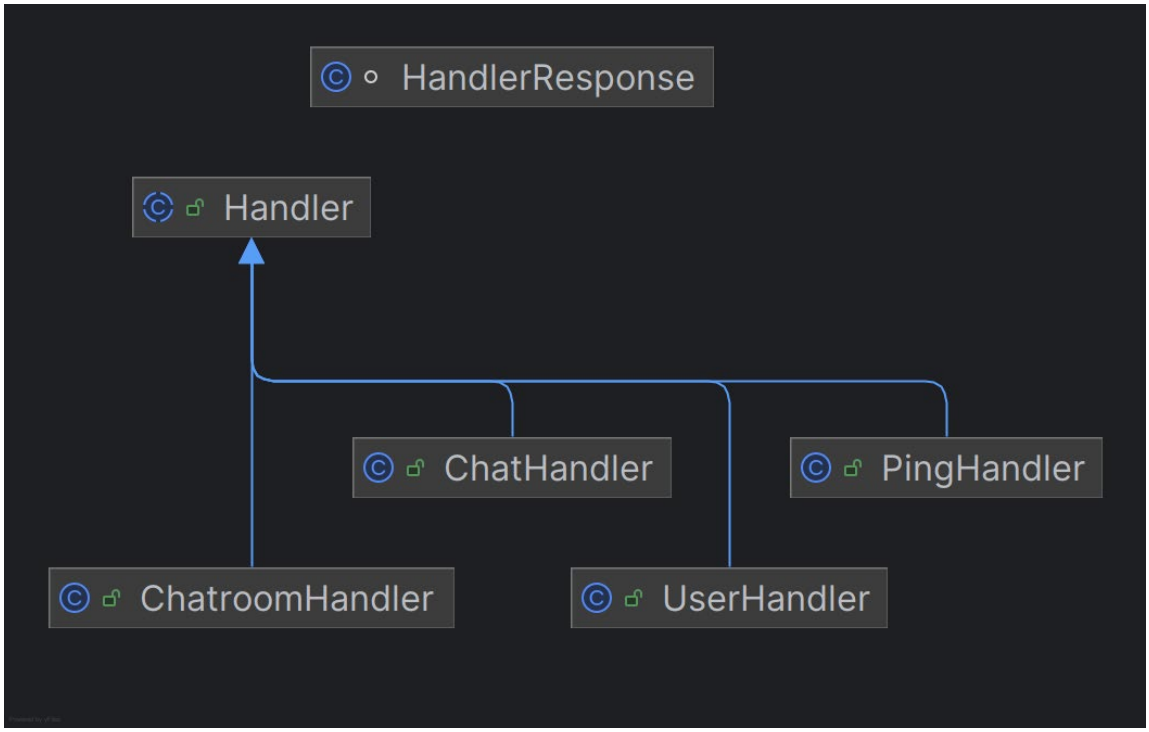
Endpoint	Erklärung
/chat/poll (POST)	Gibt private Nachrichten sowie solche aus Chatrooms zurück
/chat/send (POST)	Versenden von privaten Nachrichten

3.4 Erklärung Client Klasse

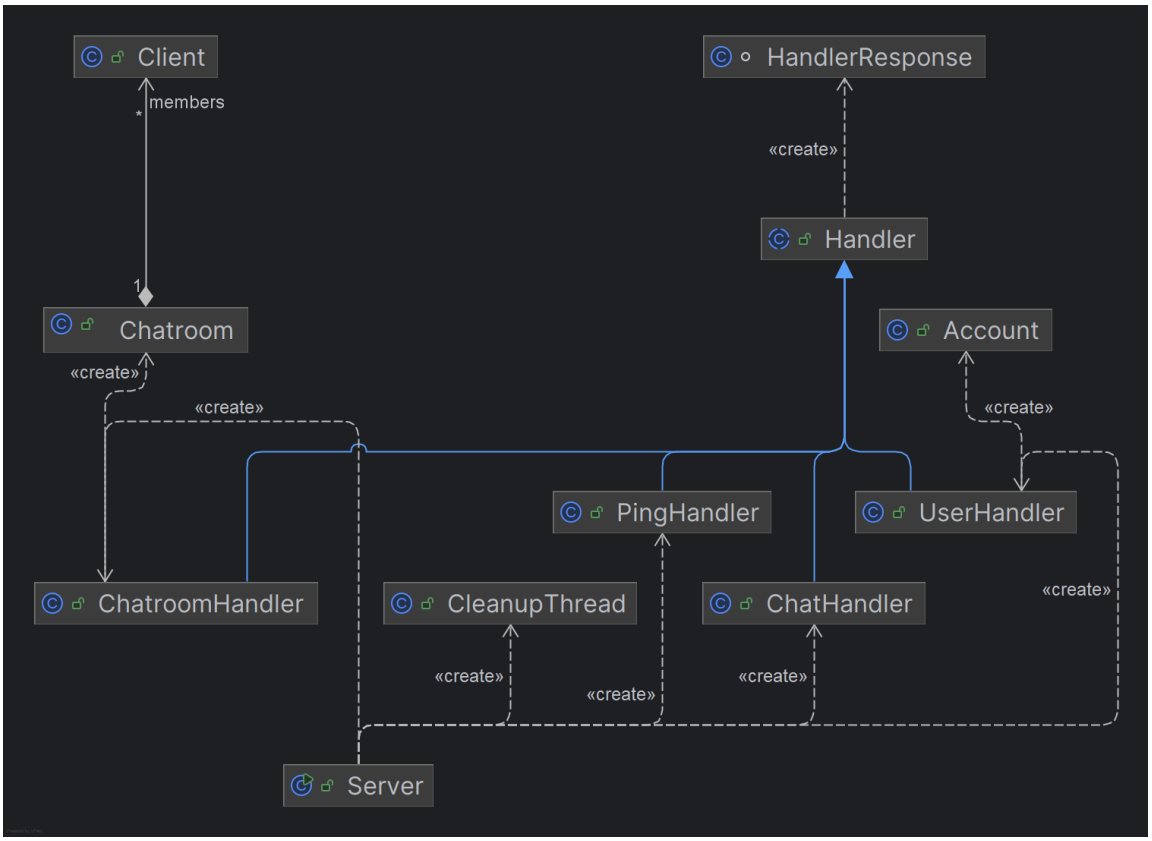
Diese Klasse ist ein Client aus der Perspektive des Servers. Ein User unseres Servers wird als Client mit username und token dargestellt.

Die wichtigsten Methoden dieser Klasse sind:
- add(String username, String token): Zur clients Liste hinzufügen.
- remove(String token): Aus der clients Liste entfernen.
- getMessages(): Alle Nachrichten dieses Clients abrufen.

3.5 Klassendiagramm Handlers



3.6 Abhängigkeiten der Klassen



3.7 Zusätzliche Hinweise

Dieses Kapitel dokumentiert spezifische technische Herausforderungen, die während der Implementierungs- und Testphase identifiziert wurden. Es werden die Lösungsansätze für die korrekte Verarbeitung von Sonderzeichen sowie die Implementierung eines Caching-Mechanismus zur Sicherstellung der Nachrichtenpersistenz bei Benutzerabmeldungen beschrieben.

3.7.1 Problem mit Spezialzeichen

Während der Testphase haben wir ein technisches Problem bei der Verarbeitung von Nachrichten mit Sonderzeichen (wie é, à, ö, ä, ü) festgestellt. Diese Zeichen wurden teilweise nicht korrekt übertragen oder serverseitig falsch interpretiert.

Bradley Richards hat den Basis-Code angepasst und dieser wurde so auch übernommen. Nach einem Test in Postman kann man nun diese Zeichen auch nutzen.

«Das Problem liegt darin, dass der Server dem HttpHandler die Länge der JSON-Daten mitteilen muss: Nicht die Anzahl Zeichen sondern die Anzahl Byte. Nicht-ASCII Zeichen sind länger als 1 Byte, was das Problem auslöst.» (Richards, 2025)

3.7.2 Problem mit Login/Logout und Persistenz der Nachrichten

Beim Testen der Standard-Use-Cases ist uns ein kritisches Verhalten im System aufgefallen, das zu Datenverlust führt.
Wir haben folgendes Szenario durchgespielt: Ein Benutzer (User A) sendet eine Nachricht an einen anderen, eingeloggten Benutzer (User B). User B empfängt diese Nachricht jedoch nicht sofort (ruft also nicht /chat/poll auf), sondern führt direkt danach den Logout-Befehl (/user/logout) aus.

Beobachtung: Beim erneuten Einloggen von User B ist die Nachricht nicht mehr auffindbar. Sie wurde weder zugestellt noch serverseitig gespeichert.

Technische Analyse: Unsere Untersuchung des Quellcodes hat gezeigt, dass dieses Verhalten architekturbedingt ist. Eingehende Nachrichten werden ausschließlich im Arbeitsspeicher (RAM) innerhalb des jeweiligen Client-Objekts in einer Liste (ArrayList) gespeichert. Der Logout-Prozess löscht die Referenz auf dieses Client-Objekt vollständig aus der Server-Verwaltung. In der Folge wird das Objekt inklusive der darin enthaltenen Nachrichtenliste vom Garbage Collector bereinigt. Da keine persistente Speicherung (z. B. Datenbank) existiert, werden nicht abgerufene Nachrichten im Moment des Logouts unwiderruflich gelöscht.

```
private void logoutUser(String token, HandlerResponse response) {
    Client.remove(token);
    response.jsonOut.put("logout", true);
}
```

Beim erneuten Login wird ein komplett neuer Client (mit einer neuer ArrayList<Message>-Instanz) generiert:

```
private void loginUser(String username, String password, HandlerResponse response)
throws Exception {
    if (username.length() < 3 && password.length() < 3) {
        throw new Exception("Invalid username or password");
    } else {
        Account account = Account.exists(username);
        if (account == null || !account.checkPassword(password)) {
            throw new Exception("Invalid username or password");
        } else {
            String token = Account.getToken();
            Client.add(username, token);
            response.jsonOut.put("token", token);
        }
    }
}
```

```
public static void add(String username, String token) {
    synchronized (clients) {
       clients.add(new Client(username, token));
    }
}
```

Lösung: Um den Verlust von ungelesenen Nachrichten beim Logout zu verhindern, haben wir einen Caching-Mechanismus implementiert, der die Lebensdauer der Nachrichten von der flüchtigen Client-Session entkoppelt.

Die technische Umsetzung erfolgte in drei Schritten:

1. **Erweiterung der Account-Klasse:** Die Klasse Account, die im Gegensatz zum Client dauerhaft im Server-Speicher verbleibt, wurde um eine Liste cachedMessages erweitert. Diese dient als Zwischenspeicher ("Tresor") für Nachrichten, solange der Benutzer offline ist.
2. **Sicherung beim Logout (UserHandler):** Im UserHandler wurde der Logout-Prozess angepasst. Bevor das Client-Objekt gelöscht wird, werden alle noch nicht abgerufenen Nachrichten extrahiert und über setCachedMessages() im zugehörigen Account-Objekt gesichert.
3. **Wiederherstellung beim Login (UserHandler & Client):** Beim Login prüft der Handler nun, ob im Account gespeicherte Nachrichten vorliegen. Ist dies der Fall, wird der neue Client über einen speziellen Konstruktor (addCachedClient) initialisiert, der die gesicherten Nachrichten sofort wieder in den Posteingang der neuen Session lädt.

Durch diese Architektur bleiben Nachrichten auch ohne Datenbank erhalten, solange der Server läuft, selbst wenn der Benutzer sich zwischenzeitlich ab- und wieder anmeldet.

3.7.3 Automatisiertes Testskript zur Validierung

Um die Stabilität der API und die Wirksamkeit der implementierten Fehlerbehebungen sicherzustellen, wurde ein umfassendes Testskript in PowerShell (testAPIscript.ps1) entwickelt. Dieses Skript simuliert komplexe Benutzerinteraktionen und validiert die System-
antworten automatisch.

Es deckt insbesondere folgende kritische Bereiche ab:

- Persistenz-Prüfung: Das Skript simuliert den Nachrichtenversand an einen Benutzer, der sich unmittelbar danach ausloggt. Beim erneuten Login wird verifiziert, ob die Nachrichten korrekt aus dem Account-Cache wiederhergestellt wurden
- Raum-Logik (Negative Testing): Es wird geprüft, ob Benutzer nach dem Verlassen eines Chatraums (/chatroom/leave) tatsächlich keine weiteren Nachrichten mehr empfangen.

Hinweis zur Erstellung: Das Testskript baut direkt auf der Logik unseres Servers auf.
Für das effiziente Debugging und die Erstellung der PowerShell-Syntax wurden unterstützend die KI-Tools Gemini und ChatGPT verwendet. Das Skript generiert bei jeder Ausführung einen detaillierten Log-Bericht, der die JSON-Antworten des Servers für jeden Testschritt dokumentiert. Dies ermöglicht eine lückenlose Nachvollziehbarkeit der Testergebnisse.

4 Literaturverzeichnis

Richards, B. (27. November 2025). Kleiner Bug in Chat Server. Olten, Solothurn, Schweiz.