ChatServer Dokumentation

Projekt: Erweiterung des Servers mit Chatroom Funktionen & Eine Chat UI erstellen

Fach: Software Engineering (SEL)
Dozent: Bradley Richards
Studenten: Lehkdup Shöntsang und Jakob Koller
GitHub Projekt: https://github.com/Lehkdup/ChatServerClient

Code Design

Die ChatServer Applikation gliedert sich in zwei Typen von Klassen: Objekte (z. B. Account, Client, Chatroom) und Handler (ChatHandler, PingHandler, UserHandler usw.). Die Handler steuern die exponierten Endpunkte und delegieren die Aufgaben an die entsprechenden Objekte.
Beispielsweise besteht eine enge Kopplung zwischen der Account-Klasse und dem UserHandler. Ebenso kontrolliert der ChatHandler weitgehend die Chatroom- und Client-Klassen. Die Klassen CleanupThread und Server bilden die Grundlage für das Erstellen von Threads, die für die Lastverteilung zuständig sind, sowie für den HttpServer, der Nachrichten sendet und empfängt.
Die abstrakte Klasse Handler wird von den Klassen ChatHandler, PingHandler und UserHandler abgeleitet. In der Basisklasse werden grundlegende Methoden implementiert, die für die Verarbeitung von GET- und POST-Nachrichten sowie das Parsen von JSON erforderlich sind. Diese Methoden können von den abgeleiteten Klassen überschrieben werden.
Die Klasse HandlerResponse dient zur Generierung von Antworten auf API-Endpunkte.

Endpunkte

GET /chatroom

Dieser Endpunkt gibt eine Liste aller verfügbaren Chatrooms zurück. Die Methode getAllChatrooms() in der ChatHandler-Klasse nimmt den Request entgegen, ruft alle Chatrooms ab und konvertiert sie in ein JSON-Array.

POST /chatroom/create

Mit diesem Endpunkt kann ein neuer Chatroom erstellt werden. Die ChatHandler-Klasse erstellt ein neues Chatroom-Objekt, das folgende Parameter benötigt:

- Name des Chatrooms
- Eine Liste von Clients (Usernamen)
- Den Ersteller des Chatrooms.

Innerhalb des Chatroom-Konstruktors wird die chatroomId automatisch inkrementiert. Der Endpunkt gibt die neu generierte ID zurück, mit der der Chatroom in späteren Abfragen identifiziert werden kann.

POST /chatroom/join

Ein eingeloggter User kann über diesen Endpunkt einem Chatroom beitreten. Der Request muss die chatroomId sowie den Login-Token des Users enthalten. Die API gibt die aktualisierte Liste aller Nutzer im Chatroom zurück.

POST /chatroom/leave

Dieser Endpunkt ermöglicht es einem Nutzer, einen Chatroom zu verlassen. Dafür müssen die chatroomId und der Login-Token des Users angegeben werden. Bei erfolgreicher Durchführung der Operation gibt die API folgendes JSON zurück:
{"leftChatroom": true}

POST /chatroom/delete

Über diesen Endpunkt kann ein Chatroom gelöscht werden. Dazu sind folgende Voraussetzungen erforderlich:

- Die chatroomId des zu löschenden Chatrooms.
- Der Auftraggeber muss den Login-Token angeben und der Ersteller des Chatrooms sein.

Wenn diese Bedingungen erfüllt sind, gibt die API folgendes JSON zurück: {"chatroomDeleted": true}

POST /chatroom/users

Dieser Endpunkt liefert eine Liste aller Nutzer in einem Chatroom. Der anfragende User muss einen gültigen Login-Token mit dem Request senden. Die API gibt eine Liste der Usernamen im Chatroom zurück.

Wichtigste Funktionalitäten

POST /chat/send

Dieser Endpunkt wurde so konzipiert, dass er sowohl für private als auch für Chatroom-Nachrichten genutzt werden kann. Unsere Entscheidung basierte auf der Annahme, dass das Versenden einer Nachricht unabhängig vom Ziel (privater User oder Chatroom) ähnlich abläuft. Die Unterscheidung erfolgt wie folgt:

- Wird ein Username mitgesendet, handelt es sich um eine private Nachricht.
- Wird eine chatroomId angegeben, wird die Nachricht als Chatroom-Nachricht behandelt.

Die Weiche basiert darauf, ob die chatroomId null ist oder eine gültige ID enthält. Die Client- und Chatroom-Klassen besitzen jeweils separate send()-Methoden. Die Methode in der Client-Klasse blieb unverändert, während die Methode in der Chatroom-Klasse alle Nachrichten an die chatroomMessages-Liste der Clients im Chatroom verteilt.

POST /chat/poll

Die Logik des Endpunkts wurde erweitert, um sowohl private als auch Chatroom-Nachrichten abzurufen. Die Client-Klasse wurde entsprechend angepasst und enthält nun zwei Listen:

- messages: für private Nachrichten.
- chatroomMessages: für Chatroom-Nachrichten.

Die Chatroom-Klasse füllt die chatroomMessages-Liste, aus der die Nachrichten beim Abrufen iteriert werden. Der Endpunkt gibt schliesslich alle Nachrichten in einem JSON-Array zurück.

Chatroom.java

Das Chatroom-Objekt repräsentiert einen einzelnen Chatroom mit allen erforderlichen Attributen. Eine statische Liste innerhalb der Klasse hält alle existierenden Chatrooms.
Die Klasse enthält verschiedene Methoden, um die Logik hinter den Endpunkten zu steuern. Dieses Setup ähnelt dem der Client-Klasse, bei der jeder Client einen eingeloggten User repräsentiert. Dadurch entstehen Synergien im Zusammenspiel zwischen den beiden Klassen.

Chat Oberfläche

Der Chat UI befindet sich in einem anderen Repo damit die Ausführung einfacher ist. Github Repo: https://github.com/Lehkdup/Chat-UI

Auf der mit Vue JS erstellten Chat Oberfläche kann man mit anderen registrierten Nutzern chatten. Ebenfalls ist es möglich die Server Adresse beim Einloggen zu wechseln. Aufgrund der Limitation vom Server werden alle Chats in einem Chat Fenster angezeigt.

Nutzung

Um das Programm zu starten, muss man sich im Terminal im Projektverzeichnis befinden. Danach müssten folgende Commands ausgeführt werden:

1. Einmalig «npm install»
2. Zum starten des Servers «npm run dev»

Wichtig zu beachten:

Im Port/URL-Feld muss «http://» eingegeben werden, zum Beispiel „http://javaprojects.ch:50001“. Danach erhält man entweder einen grünen Umriss oder die Anzeige „Online“, um den Serverstatus zu verifizieren.

Zunächst müssen die Benutzer registriert werden, die getestet werden sollen. Erst danach kann man sich mit den erstellten Benutzern einloggen. Falls ein Benutzer sich nach dem Einloggen registriert, muss der eingeloggte Benutzer die Seite neu laden, damit der neue Benutzer korrekt registriert wird. (Alle Tests wurden mit dem Server «avaprojects.ch:50001» durchgeführt.)

Projektstruktur

Die **Projektstruktur** der Anwendung basiert auf einer modularen Architektur, die sich in mehrere Kategorien von Ordnern und Dateien gliedert. Diese fördern die Wiederverwendbarkeit und Wartbarkeit der Codebasis.

Der **src/**-Ordner enthält alle zentralen Module und ist in verschiedene Unterverzeichnisse aufgeteilt:

- **components/**: Enthält die Vue-Komponenten wie Login.vue und ChatLayout.vue. Diese Komponenten repräsentieren die UI-Elemente der Anwendung.
- **stores/**: Beinhaltet den auth.tsStore für die Verwaltung der Authentifizierungsdaten.

- **services/**: Umfasst Service-Module wie api.ts, die mit Hilfe von Axios HTTP-Anfragen an die API stellen.
- **router/**: Definiert in index.ts die Navigation und Routen der Anwendung.
- **App.vue**: Die Hauptkomponente, die das Gerüst der Anwendung bereitstellt und andere Komponenten einbindet.
- **main.ts**: Der Einstiegspunkt der Anwendung, in dem die Vue-Instanz erstellt und die Hauptkonfiguration geladen wird.

Hauptkomponenten und deren Zusammenspiel

1. **App.vue**
   Diese zentrale Komponente verwendet <router-view>, um die jeweils aktive Route darzustellen, und steuert Übergänge zwischen den Komponenten.
2. **Login.vue**
   Implementiert ein Formular zur Benutzeranmeldung. Nach erfolgreicher Authentifizierung werden das Token und der Benutzername im auth-Store gespeichert, und der Benutzer wird zum Chat weitergeleitet.
3. **Signup.vue**
   Ermöglicht neuen Benutzern die Registrierung. Die Komponente validiert eingegebene Passwörter und speichert nach erfolgreicher Registrierung das Token im Store.
4. **ChatLayout.vue**
   Diese Komponente dient zur Verwaltung von Einzel- und Gruppenchats (wurde nicht vollständig implementiert). Sie zeigt Nachrichten an und ermöglicht das Senden neuer Nachrichten.
5. **stores/auth.ts**
   Ein zentraler Store zur Verwaltung des Authentifizierungstokens und Benutzernamens. Er stellt Methoden bereit, um Authentifizierungsdaten zu setzen oder zu löschen.
6. **services/api.ts**
   Definiert eine Axios-Instanz und bietet Services für die Authentifizierung, Chats und Benutzerverwaltung. Dies sorgt für eine klare Trennung der API-Logik.
7. **router/index.ts**
   Legt die Routen der Anwendung fest, z. B. /login für die Anmeldeseite oder /chat für den Chatbereich.

Wichtige Implementierungsdetails

- **Authentifizierung**:
  Der auth-Store stellt sicher, dass der Authentifizierungsstatus zentral verwaltet wird, wodurch alle Komponenten darauf zugreifen können.
- **State Management mit Pinia**:
  Pinia wird als zentrale State-Management-Library genutzt, um den Zustand der Anwendung effizient zu synchronisieren.
- **API-Kommunikation mit Axios**:
  Das Modul api.ts ermöglicht mit einer vorkonfigurierten Axios-Instanz eine saubere Trennung der API-Endpunkte und bietet Services für die Kernfunktionen.
- **Routing und Navigation Guards**:
  Der Vue Router sorgt für die Navigation zwischen den Routen. Navigation Guards gewährleisten, dass nur authentifizierte Benutzer geschützte Bereiche betreten können.
- **Polling für Nachrichten**:
  Die Chat-Komponenten verwenden einen Polling-Mechanismus, um regelmässig neue Nachrichten vom Server abzurufen und so eine Echtzeitkommunikation zu ermöglichen.