



**Hochschule
Augsburg** University of
Applied Sciences

Fakultät für
Informatik

Studienarbeit

Studienrichtung
Informatik (Master)

Julian Hillesheimer (954018)

Single-Page-Chat-Webanwendung mit TypeScript und AngularJS - Chatserver

Teamkollege: Bernd Fecht
Prüfer: Prof. Dr. Phillip Heidegger
Abgabe der Arbeit am: 12.07.2017

Hochschule für angewandte
Wissenschaften Augsburg
University of Applied Sciences

An der Hochschule 1
D-86161 Augsburg

Telefon +49 821 55 86-0
Fax +49 821 55 86-3222
www.hs-augsburg.de
info@hs-augsburg.de

Fakultät für Informatik
Telefon: +49 821 5586-3450
Fax: +49 821 5586-3499

Verfasser der Studienarbeit:
Julian Hillesheimer

Inhaltsverzeichnis

1	Einleitung	3
1.1	Aufgabenstellung	3
1.2	Verwendete Technologien	3
2	Chatserver	5
2.1	Architektur	5
2.2	Server	5
2.2.1	Client-Server Kommunikation	6
3	Abschluss	10
	Literatur	11

1 Einleitung

Im Rahmen dieser Studienarbeit soll eine einfache Single-Page-Chat-Webanwendung mit TypeScript und AngularJS entwickelt werden. In den beiden nachfolgenden Abschnitten werden hierfür zunächst die Anforderungen definiert und die dafür verwendeten Technologien vorgestellt. In dem nachfolgenden Kapitel wird die Implementierung unter Verwendung der zuvor beschriebenen Technologien geschildert, dabei wird auch auf den internen Aufbau der Anwendung eingegangen. In einem abschließenden Kapitel wird geprüft, ob die zuvor spezifizierten Anforderungen umgesetzt werden konnten.

1.1 Aufgabenstellung

Für die zu entwickelnde Single-Page-Chat-Webanwendung wurden die nachfolgenden Anforderungen spezifiziert.

1. Login und Logout (Server)
2. Chaträume bereitstellen
3. Persistenz (Benutzer und Chat-Räume)

1.2 Verwendete Technologien

Nachfolgend werden die Technologien aufgeführt, welche für die Entwicklung der Single-Page-Chat-Webanwendung verwendet werden, und anschließend beschrieben.

1. Express.js
2. nodemon
3. RxJS
4. Socket.IO
5. TSLint

6. TypeScript Node

7. TypeScript

8. Zone.js

Express.js ist ein schnelles, offenes, unkompliziertes Web-Framework für Node.js. [Exp]

Nodemon ist ein Werkzeug, welches den Quelltext nach Änderungen überwacht und automatisch einen Neustart des Servers durchführt. [Nod]

RxJS ist eine Bibliothek für reaktive Programmierung. [Rxj]

Socket.IO ist eine JavaScript Bibliothek, welche bidirektionale und ereignisgetriebene Kommunikation in Echtzeit ermöglicht. [Npma]

TSLint ist ein statisches Analysewerkzeug, das TypeScript-Quelltext auf Lesbarkeit, Wartbarkeit und funktionale Fehler untersucht. [Npmb]

TypeScript Node ist eine TypeScript Ausführungsumgebung für Node. [Npmc]

TypeScript ist eine typisierte Sprache, die zu reinem JavaScript übersetzt werden kann. [Typ]

Zone.js stellt für JavaScript Ausführungskontexte zur Verfügung, welche über asynchrone Aufgaben hinweg bestehen bleiben und als Zonen bezeichnet werden. [Npmd]

2 Chatserver

Bevor die Implementierung des Chatserver erläutert wird, soll zunächst die Architektur der Single-Page-Chat-Webanwendung skizziert werden.

2.1 Architektur

Die Architektur der Single-Page-Chat-Webanwendung, dargestellt in der Abbildung 2.1, teilt sich in die beiden Bereiche Server und Client auf. Der Server stellt die Sockets bereit, über welche die Daten zwischen dem Server- und dem Clientprogramm ausgetauscht werden und worüber die Chat-Kommunikation erfolgt. Des Weiteren besitzt der Serverbereich noch einen Datenhandler, um auf gespeicherte Daten in einer JSON-Datei, wie beispielsweise für die Authentifikation zugreifen zu können, und die Klasse Room, damit die Chat-Kommunikation auch über unterschiedliche Chaträume verschickt werden kann. Der Clientbereich besteht aus den Komponenten App, Chat, Login und Room. Der Clientbereich enthält darüber hinaus noch einen Client für Sockets, Modelle und Dienste. Der Socketclient ermöglicht die Nutzung der vom Server bereitgestellten Sockets. Login dient der Authentifizierung von Benutzern, die die Chat-Webanwendung nutzen möchten. Die Komponente Chat dient dem Nachrichtenaustausch. Die Komponente Room ermöglicht dem Benutzer die Auswahl eines Chatraums. In den Modellen sind die Benutzer, der Antrag für das Betreten eines Raums, die Nachrichten, die Räume und die Ereignisse der Sockets abgebildet. Die letzte Komponente stellt Dienste für die Authentifizierung, die Nutzung der Sockets und dem Betreten von Räumen bereit.

2.2 Server

Der Chatserver verwendet die zuvor beschriebenen Technologien Express und Socket.IO und hat des Weiteren eine Referenz auf den Datenhandler. Der Server, abgebildet durch die Klasse `server.ts`, wurde nach dem Entwurfsmuster Singleton entwickelt. Dadurch wird sichergestellt, dass zu jedem Zeitpunkt nur eine einzige Serverinstanz existiert. In der Methode `init()` wird das Ver-

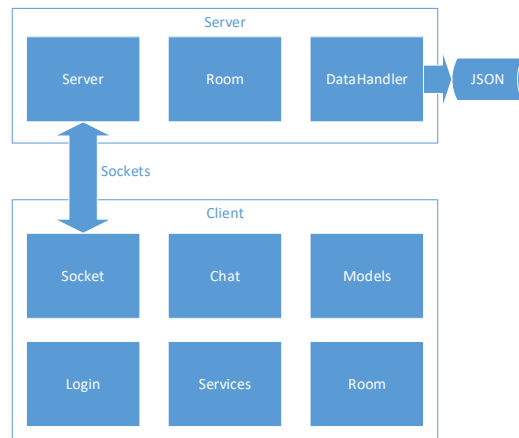


Abbildung 2.1: Architektur der Chat-Webanwendung

halten des Servers initialisiert. Für jeden Socket, der über eine ID identifiziert werden kann, wird bei dem Ereignis LOGIN überprüft, ob der Benutzer, der sich authentifizieren möchte, vorhanden und gültig ist. Bei dem Ereignis MESSAGE wird die Nachricht über den Socket übertragen und kann von jedem Client, der diesen Socket verwendet, empfangen werden. Bei dem Ereignis GETROOMS liefert der Server alle verfügbaren Chaträume über den Socket an den Client aus. Bei JOINROOM fügt der Server den anfragenden Benutzer dem Raum hinzu und aktualisiert den Raum, indem die aktualisierten Rauminformationen über den Socket übertragen werden. Der Server nutzt die Klasse DataHandler für das Ereignis LOGIN. Über diese Klasse können die Benutzer, welche in der JSON-Datei user.json enthalten sind, ausgelesen werden. Die Methode checkForUser(user: User) überprüft dabei, ob die übergebenen Daten des Benutzers, welcher sich authentifizieren möchte, in der JSON-Datei enthalten sind. Die JSON-Datei besteht aus einer Liste von Benutzereinträgen, die die Attribute Benutzername und Passwort enthalten.

2.2.1 Client-Server Kommunikation

Wie im Quelltextausschnitt 2.1 zu sehen, wird in der Klasse server.ts ein Socket erstellt, über den die Kommunikation zwischen Server und Client durchgeführt werden kann. Für das Socket wird dabei der Standardnamensraum verwendet. Für die Kommunikation werden dabei die Ereignisse LOGIN und MESSAGE verwendet. Damit Benutzer Chaträume nutzen können, werden die Chaträume durch das Ereignis GETROOMS an alle Clients übertragen, die auf dem Socket auf Informationen lauschen. Der Mechanismus der Chaträume funktioniert dadurch, dass Nachrichten oder Informationen über

die Nachrichten über einen eigenen Kanal über den Socket übertragen werden. Der Kanal entspricht dabei dem Raumnamen. Somit erhalten nur die Clients, welche diesen Kanal beobachten, die Nachrichten und Informationen. Bei den Ereignissen JOINROOM und LOGOUT werden die Benutzerinformationen zu dem spezifizierten Raum hinzugefügt oder entfernt und anschließend über den zugehörigen Kanal an die entsprechenden Clients ausgeliefert.

```
public init() {
  this.app.get('/', function (req, res) {
    res.send("<h1>This is the socket.io server for the HillFe-Chat!</h1>")
  });
  this.io.on('connection', (socket) => {
    console.log("User connected with id: " + socket.id);
    socket.on('disconnect', () => {
      console.log('Guest-user disconnected');
    });
    socket.on(SocketEvents.GETROOMS, () => {
      this.io.to(socket.id).emit(SocketEvents.GETROOMS, this.rooms);
    });
    socket.on(SocketEvents.JOINROOM, (msg) => {
      let request: JoinRoomRequest = msg;
      let room: Room = this.addUserToRoom(request);
      if (room) {
        // make "db" save
        this.dataHandler.saveRooms(this.rooms);
        this.io.to(socket.id).emit(SocketEvents.JOINROOM, room);
        socket.join(request.newRoom.roomname);
        this.io.to(request.oldRoom.roomname).emit(SocketEvents.ROOMCHANGED, this.getRoom(request.oldRoom));
        // Emit for new room
        this.io.to(request.newRoom.roomname).emit(SocketEvents.ROOMCHANGED, this.getRoom(request.newRoom));
      }
    });
    socket.on(SocketEvents.MESSAGE, (msg) => {
      console.log('Server emitted: ' + JSON.stringify(msg));
      this.io.to(this.getRoomForUser(msg.username).roomname).emit(SocketEvents.MESSAGE, msg);
    });
    socket.on(SocketEvents.LOGIN, (msg) => {
      console.log("Server emitted login for user: " + JSON.stringify(msg));
      this.io.to(socket.id).emit(SocketEvents.LOGIN, this.dataHandler.checkForUser(msg));
    });
  });
}
```

```
socket.on(SocketEvents.LOGOUT, (msg) => {
  console.log("Server emitted logout for user: " + JSON.
    stringify(msg));
  let oldroom: Room = this.getRoomForUser(msg);
  this.removeUser(msg);
  this.io.to(oldroom.roomname).emit(SocketEvents.ROOMCHANGED
    , oldroom);
  this.dataHandler.saveRooms(this.rooms);
});
});
}
```

Listing 2.1: Server-Socket

Damit der Client mit dem Server kommunizieren kann, wurde die Klasse ClientSocket erstellt. Diese Klasse ist wie der Server nach dem Entwurfsmuster Singleton konzipiert. Damit der Client Zugriff auf den Server erhält, siehe Quelltextausschnitt 2.2, besitzt diese Klasse ebenfalls ein Socket, den diese über die URL des Servers erhält. Damit ist nun die Verbindung zwischen Server und Client über ein Socket etabliert.

```
export class ClientSocket {
  public readonly url = "http://localhost:4500";
  private static instance: ClientSocket;
  public static socket;

  private constructor () {
    ClientSocket.socket = io(this.url);
  }

  static getInstance(): ClientSocket {
    if (!ClientSocket.instance) {
      ClientSocket.instance = new ClientSocket();
    }
    return ClientSocket.instance;
  }
}
```

Listing 2.2: Client-Socket

Die Klasse SocketService stellt Methoden bereit, um diese Verbindung zu nutzen. Wie im Quelltextausschnitt 2.3 dargestellt, kann über die Methode emit() ein Ereignis mit zugehörigen Daten über den Socket an den Server übertragen werden, indem die gleichnamige Methode auf dem Socket aufgerufen wird.

```
public emit(event: SocketEvents, content: any): void {
  ClientSocket.socket.emit(event, content);
  console.log("Emitted event '%s' with content '%s'.", event, JSON
    .stringify(content));
}
```



```
}
```

Listing 2.3: Daten über Socket übertragen

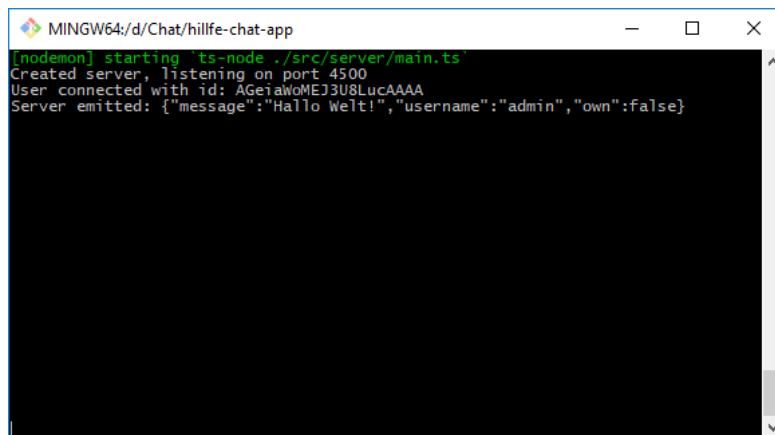
Die Methode `getResponse` kann genutzt werden, um auf eine Antwort, die über den Socket übertragen wird, zu warten. Hierfür wird, wie im Quelltextausschnitt 2.4 abgebildet, ein Beobachter verwendet. Dieser Beobachter informiert alle Interessenten, wenn ein zuvor spezifiziertes Ereignis auf dem Socket stattfindet, indem dabei die Daten des Ereignisses an die Interessenten zur Weiterverarbeitung weitergereicht werden.

```
public getResponse(event: SocketEvents): Observable<any> {  
  let observable = new Observable(observer => {  
    ClientSocket.socket.on(event, (data) => {  
      observer.next(data);  
    },  
    err => console.error(err.message)  
  );  
});  
return observable;  
}
```

Listing 2.4: Daten über Socket empfangen

3 Abschluss

Um abschließend den Server zu überprüfen, wurde dieser über die Kommandozeile mit dem Befehl `npm start` gestartet. Anhand von Logausgaben auf die Konsole, dargestellt in Abbildung 3.1, konnte sichergestellt werden, dass über den Socket des Servers Daten für das Login und das Versenden von Nachrichten erfolgreich übertragen werden konnten. Damit wurden die Anforderungen für das Login und das Senden von Nachrichten erfolgreich verifiziert.

A screenshot of a terminal window titled "MINGW64:/d/Chat/hilfe-chat-app". The terminal output shows the following lines:

```
[nodemon] starting 'ts-node ./src/server/main.ts'
Created server, listening on port 4500
User connected with id: AGeIawoMEJ3U8LucAAA
Server emitted: {"message":"Hallo Welt!","username":"admin","own":false}
```

Abbildung 3.1: Überprüfung des Servers

Der Chatserver könnte zukünftig noch um die Funktionen für eine Registrierung neuer Benutzer, einer Speicherung der Chatnachrichten, Verwaltung von Chaträumen und Rollen für Benutzer erweitert werden.

Literatur

- [Exp] *Express*. 9. Juli 2017. URL: <http://expressjs.com/de/>.
- [Nod] *nodemon reload, automatically*. 9. Juli 2017. URL: <https://nodemon.io/>.
- [Npma] *socket.io*. 9. Juli 2017. URL: <https://www.npmjs.com/package/socket.io>.
- [Npmb] *tslint*. 9. Juli 2017. URL: <https://www.npmjs.com/package/tslint>.
- [Npmc] *TypeScript Node*. 9. Juli 2017. URL: <https://www.npmjs.com/package/ts-node>.
- [Npmd] *zone.js*. 9. Juli 2017. URL: <https://www.npmjs.com/package/zone.js>.
- [Rxj] *The ReactiveX library for JavaScript*. 9. Juli 2017. URL: <http://reactivex.io/rxjs/>.
- [Typ] *TypeScript*. 9. Juli 2017. URL: <https://www.typescriptlang.org/>.