Hochschule
Augsburg University of
Applied Sciences

Fakultät für
Informatik

Studienarbeit

Studienrichtung
Informatik (Master)

Bernd Fecht (954020)

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

Teamkollege: Julian Hillesheimer
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:
Bernd Fecht

Inhaltsverzeichnis

1 Einleitung **3**
1.1 Aufgabenstellung 3
1.2 Verwendete Technologien 3

2 Chatclient **5**
2.1 Komponente Log-in 6
2.2 Komponente Chat 7
2.3 Komponente Räume 10

3 Abschluss **11**

Literatur **12**

1 Einleitung

Im Rahmen dieser Studienarbeit soll ein Client für eine 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 Kapitel 2 wird die Implementierung des Chatclients anhand der zuvor geschilderten Technologie und Quelltextausschnitten erläutert. Im abschließenden Kapitel 3 wird der entwickelte Client den gestellten Anforderungen gegenübergestellt und ein Ausblick auf zukünftige Erweiterungen gegeben.

1.1 Aufgabenstellung

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

1. Log-in und Logout (Client)
2. Betreten und Verlassen von Chaträumen
3. Verfassen von Nachrichten in Chaträumen
4. Lesen von Nachrichten in Chaträumen

1.2 Verwendete Technologien

Für die Single-Page-Chatclient-Webanwendung wurden die nachfolgenden Technologien verwendet.

1. Angular
2. Bootstrap
3. RxJS
4. Socket.IO
5. TSLint

6. TypeScript Node
7. TypeScript
8. Zone.js

Angular ist eine Plattform, die es einfach macht, Anwendungen für das Web zu erstellen. [Ang]

Bootstrap ist ein Framework für die Front-End Entwicklung von Webanwendungen. [Npma]

RxJS ist eine Bibliothek für reaktive Programmierung. [Rxj]

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

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

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

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. [Npme]

2 Chatclient

Der Client verwendet die bereits beschriebenen Technologien Angular, Bootstrap, Socket.IO und Zone.js. Der Client für die Chat-Webanwendung besteht dabei, aus den Komponenten AppComponent, LoginComponent, ChatComponent und RoomComponent. Darüber hinaus bilden die Klassen Message, SocketEvents, User, JoinRoomRequest und Room die Datenmodelle und AuthenticationService, SocketService und RoomService die Dienste ab. Da der Chatclient mit Angular entwickelt wird, besitzt die Anwendung einen spezifischen Aufbau. Jede Komponente besteht aus mindestens drei Dateien. In der CSS-Datei einer Komponente wird die Darstellung festgelegt, in der dazugehörigen HTML-Datei die Struktur und die Dateien mit der Endung .ts enthalten den TypeScript Quelltext, der die Logik enthält. Eine Besonderheit bildet die Komponente AppComponent, da diese die Anwendung, alle darin enthaltenen Komponenten, Module und Pfade beschreibt. Wie im Quelltextausschnitt 2.1 dargestellt, ist diese Beschreibung in der Datei app.modules.ts enthalten. Dort befindet sich die Deklaration der zuvor genannten Komponenten, die importierten Module und darüber hinaus ist auch definiert, welcher Pfad, zu welcher Komponenten gehört.

```
const appRoutes: Routes = [
  {path: "", redirectTo: "/chat", pathMatch: "full" },
  {path: "login-page", component: LoginComponent},
  {path: "chat", component: ChatComponent},
];

@NgModule({
  declarations: [
    AppComponent,
    LoginComponent,
    ChatComponent,
    RoomComponent
  ],
  imports: [
    RouterModule.forRoot(
      appRoutes,
      {useHash: false, enableTracing: true}, // <--- debugging
          purposes only
    ),
    BrowserModule,
```

```
    FormsModule,
    HttpModule
  ],
  providers: [SocketService, AuthenticationService, RoomService],
  bootstrap: [AppComponent]
})
```

Listing 2.1: Erstellung des Chatclient mit Angular

2.1 Komponente Log-in

Die Komponente für das Log-in ist durch die Klasse LoginComponent abgebildet. Die Klasse hält eine Referenz auf ein leeres Objekt des Typs User. Die Attribute dieses Objekts sind per Binding an die beiden Eingabefelder, die in der HTML-Datei login.component.html für das Log-in enthalten sind, gebunden. Zudem ist der Button für das Log-in an die login() Methode dieser Klasse gebunden. Dies ist im Quelltextausschnitt 2.2 dargestellt. Erfolgt eine Eingabe, werden automatisch die Attribute dieses Objekts mit den entsprechenden Werten befüllt.

```
<input [(ngModel)]="user.username" id="email" type="text"
    class="form-control" name="Username" placeholder="
    Username" required="" autofocus="" />
<input [(ngModel)]="user.password" id="password" type="
    password" class="form-control" name="Password"
    placeholder="Password" required=""/>
<button (click)="login()" class="btn btn-lg btn-primary btn
    -block" name="Submit" value="Login" type="Submit">
    Login</button>
```

Listing 2.2: Binding zwischen Benutzer und Log-inmaske

Die Methode login() wertet die Attribute des Benutzerobjekts aus, indem dafür der Dienst AuthenticationService verwendet wird. Findet dieser die Benutzerdaten, wird, wie in Quelltextausschnitt 2.3 abgebildet, der Benutzer im lokalen Speicher des Browsers hinterlegt und über den Routingmechanismus von Angular wird zu der Ansicht des Chats gewechselt.

```
private login() {
  this.authService.login(this.user).then((res) => {
    if (res == true) {
      localStorage.setItem('currentUser', JSON.stringify(this.
          user));
      console.log("User successfully loged in.");
      // Wait for promise fulfillment
      this.router.navigate(["/chat"]).then();
    }
```

```
    else {
      this.errorMsg = "User not registered yet!";
    }
  })
}
```

Listing 2.3: Mechanismus für den Log-in eines Benutzers

AuthenticationService enthält auch eine Methode login(). Die im Quelltextausschnitt 2.4 dargestellt, über einen Socket ein LOGIN Ereignis mit den Benutzerdaten verschickt. Im zweiten Schritt wird die Abfrage der Antwort in einer Promise gekapselt, die Antwort, sobald diese eintrifft, zurückgibt. Dies ist notwendig, da es sich hier um asynchrone Abläufe handelt.

```
public login(user: User): Promise<any> {
  // Emit login to server
  this.socketService.emit(SocketEvents.LOGIN, user);
  // Wait for response
  return new Promise((resolve, reject) => {
    this.socketService.getResponse(SocketEvents.LOGIN)
      .subscribe((data) => {
        resolve(data);
    },
    err => {
      console.error("Failed to get a resonse from server at login
          . \n Error: " + err.message);
      reject(err);
    }
  )});
}
```

Listing 2.4: Dienst zur Authentifikation

2.2 Komponente Chat

Die Komponente Chat ist dafür verantwortlich, dass Chatnachrichten gesendet und bei den jeweiligen Empfängern dargestellt werden. Die Komponente wird durch die Klasse ChatComponent abgebildet. Die Komponente verwendet die beiden Dienste AuthenticationService und SocketService. Die Methode logout delegiert an die gleichnamige Methode des AuthenticationService. Diese Methode entfernt die Benutzerdaten des gegenwärtig angemeldeten Benutzers aus dem lokalen Speicher des Browsers und navigiert zu der Seite für das Log-in. Die statische Methode getTimeStamp berechnet die aktuelle Uhrzeit und liefert diese zurück. Die Klasse enthält eine Referenz auf ein Objekt des Typs Message. Dieses Objekt ist per Binding mit dem Inhalt des Eingabefelds des Chats verbunden. Der Button zum Abschicken verweist

wiederum auf die Methode sendMessage() der Klasse ChatComponent. Wie im Quelltextausschnitt 2.5 dargestellt, wird in dieser Methode der Inhalt des Nachrichtenobjekts als Daten für das Ereignis MESSAGE über den SocketService verschickt.

```
<div class="message_write">
  <textarea [(ngModel)]="message" class="form-control"
      placeholder="Schreibe eine Nachricht"></textarea>
  <div class="clearfix"></div>
  <div class="chat_bottom">
    <a (click)="sendMessage()" class="pull-right btn btn-success"
       >Senden</a>
  </div>
</div>

private sendMessage(): void {
  // Send only when message is typed
  if (this.message) {
    let msg: Message = new Message(this.message, this.authService
        .getCurrentUser().username);
    this.chatService.emit(SocketEvents.MESSAGE, msg);
    this.message = '';
  }
}
```

Listing 2.5: Versenden einer Chatnachricht

Für das Empfangen und die Darstellung von Chatnachrichten enthält die Klasse ein Array vom Typ Message, welches alle empfangen Nachrichten speichert. Die Klasse ChatComponent implementiert die Schnittstelle OnInit aus Angular. Damit wird gewährleistet, dass die Methode ngOnInit() genau einmal bei der Erstellung eines neuen Objekts aufgerufen wird. In dieser Methode wird definiert, wie Nachrichten empfangen und dargestellt werden. Wie im Quelltextausschnitt 2.6 abgebildet wird über den SocketService ein Observable geholt, wenn im lokalen Speicher des Browsers gültige Benutzerinformationen vorliegen. Indem man sich als Interessent bei diesem Beobachter einträgt, wird man informiert, sobald Nachrichten über den Socket des Beobachters übertragen wird. Erhält man eine Nachricht von dem Beobachter, wird über die zuvor beschriebene Methode getTimeStamp() die gegenwärtige Uhrzeit abgespeichert. Des Weiteren wird überprüft, ob die empfangene Nachricht dem eigenen Benutzer zugeordnet werden kann und dementsprechend wird ein Flag in der Nachricht gesetzt. Zuletzt wird die empfangene Nachricht, dem Array mit den bisher empfangen Nachrichten hinzugefügt.

```
ngOnInit(): void {
  if(!this.authService.checkCredentials()) {
    this.router.navigate(["login-page"]).then();
```

```
  }
  this.connection = this.chatService.getResponse(SocketEvents.
      MESSAGE).subscribe(message => {
    this.timeStamp = ChatComponent.getTimeStamp();
    let msg: Message = message;
    if (msg.username === this.authService.getCurrentUser().
        username) {
      msg.own = true;
    }
    this.messages.push(msg);
  })
}
```

Listing 2.6: Verarbeitung von eingehenden Nachrichten

Die Darstellung der Chatnachrichten erfolgt über eine Direktive aus Angular, welche direkt in der HTML-Datei für den Chat eingebettet ist. Wie im Quelltextausschnitt 2.7 ersichtlich wird, erstellt die Direktive *ngFor für jede Nachricht, welche im Array messages enthalten ist, einen Listeneintrag erstellt. Über weitere Direktiven, CSS und Binding wird die Nachricht rechtsbündig, die Uhrzeit linksbündig und der Avatar des Benutzers rechtsbündig dargestellt, wenn es eine Nachricht des eigenen Benutzers ist. Ansonsten sind die Ausrichtungen vertauscht.

```
<div class="chat_area">
  <ul id="chat-panels" class="list-unstyled">
    <li *ngFor="let msg of messages" class="left clearfix" [
        ngClass]="{'own_chat': msg.own}">
      <span class="chat-img1" [ngClass]="{'pull-right': msg.own,
          'pull-left': !msg.own}">
        <img src="{{placeholderUrl}}" alt="User Avatar" class="
            img-circle">
      </span>
      <div class="chat-body1 clearfix" [ngClass]="{'
          text_align_right': msg.own, 'text_align_left': !msg.own
          }">
        <p>{{msg.message}}</p>
        <div class="chat_time" [ngClass]="{'pull-left': msg.own,
            'pull-right': !msg.own}">{{timeStamp}}</div>
      </div>
    </li>
  </ul>
</div>
```

Listing 2.7: Darstellung der Chatnachrichten

2.3 Komponente Räume

Diese Komponenten wird durch die Klasse RoomComponent abgebildet. Durch die Methode ngOnInit() wird gewährleistet, dass jeder Benutzer, der die Chatseite betritt, automatisch ein Standardchatraum zugewiesen wird. Darüber hinaus werden in einem Drop-Down-Menü alle Räume aufgelistet. Der Benutzer kann darüber den Raum wechseln. Wie im Quelltextausschnitt abgebildet, funktioniert dies, indem jeder Eintrag, der über die Direktive *ngFor erstellt wurde, mit der Methode changeRoom() verknüpft wird. Diese Methode benutzt den Dienst RoomService um den gewählten Raum zu betreten. Der Dienst schickt ein Ereignis vom Typ JOINROOM über den Socket an den Server und liefert anschließend eine Promise zurück, welche die asynchrone Antwort des Servers kapselt. Die Antwort liefert den Raum zurück. Da dort auch der Raumname enthalten ist, läuft die Chatkommunikation fortan über den Socketkanal des Chatraums.

```
<ul class="dropdown-menu" aria-labelledby="dropdownMenu2">
  <li><a *ngFor="let room of rooms" (click)="changeRoom(room)">
      {{room.roomname}} </a></li>
</ul>

public changeRoom(room: Room) {
  this.roomService.joinRoom(this.authService.getCurrentUser(),
      this.currentRoom, room).then((res) => {
    if (res) {
      console.log("Changed room to: "+ room.roomname);
      this.currentRoom = res;
      this.users = this.currentRoom.usernames;
      this.chat.messages = [];
    }
    else
      console.log("Can't change user to new room!")
    }
  )
}
```

Listing 2.8: Auswahl eines Chatraums

3 Abschluss

Um den Chatclient zu überprüfen, wurde dieser zunächst über die Kommandozeile mit dem Befehl ng serve gestartet. Anschließend wurde der Client über die zugehörige URL im Browser aufgerufen. Bei falschen Benutzereingaben schlug das Log-in fehl. Mit Eingabe des Benutzernamens *user* und des Passworts *password* war die Anmeldung erfolgreich und eine Weiterleitung zum Chat fand statt. Dies bestätigt die Abbildung 3.1. Darüber hinaus ist dort zu sehen, dass eigene Nachrichten, sowie Nachrichten von anderen Benutzern empfangen werden konnten. Da die Nachrichten nicht über den Standardchatraum versendet wurden, zeigt die Abbildung auch, dass der Mechanismus der Chaträume funktioniert. Damit wurden die Anforderungen Persistenz von Benutzern, Verfassen und Lesen von Nachrichten in Chaträumen und Log-in erfolgreich überprüft.

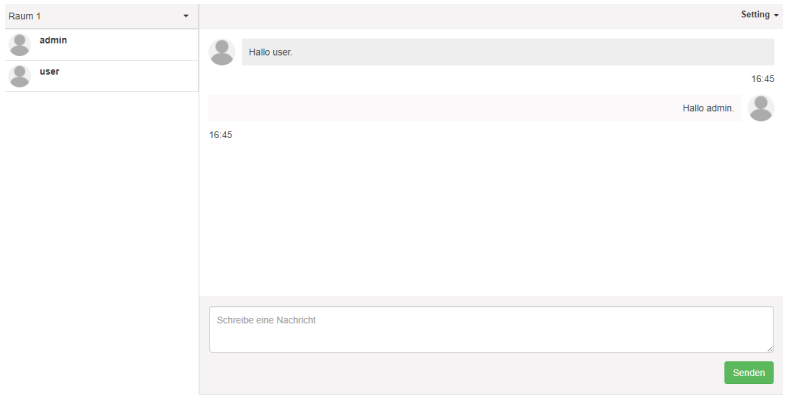

Abbildung 3.1: Überprüfung des Chatclients

Der Chatclient könnte zukünftig um die nachfolgenden Anforderungen erweitert und verbessert werden.

- Gruppierung von Nachrichten
- Darstellung der Benutzerrollen im Chatraum
- Darstellung der Anzahl der neuen Nachrichten
- Ändern der Benutzerinformationen (Name, Passwort, usw.)

Literatur

[Ang] *What is Anular?* 9. Juli 2017. URL: https://angular.io/docs.

[Npma] *bootstrap*. 10. Juli 2017. URL: https://www.npmjs.com/package/bootstrap.

[Npmb] *socket.io*. 9. Juli 2017. URL: https://www.npmjs.com/package/socket.io.

[Npmc] *tslint*. 9. Juli 2017. URL: https://www.npmjs.com/package/tslint.

[Npmd] *TypeScript Node*. 9. Juli 2017. URL: https://www.npmjs.com/package/ts-node.

[Npme] *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/.