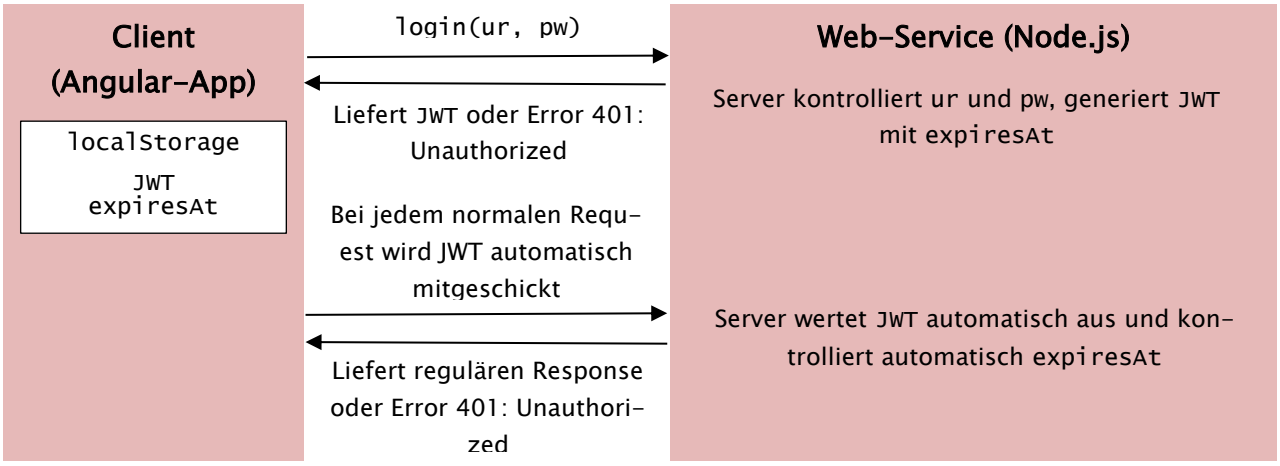
Exkurs: Authentifizierung in Node.js durch JSON Web Tokens (JWT)

Prinzip



1. Der Client (Angular-App) schickt an den Web-Service Benutzername und Passwort für den Login-Vorgang und erhält ein JWT welches in codierter Form eine Payload (z.B. Benutzername, Benutzernummer, Benutzergruppe, usw.) – diese ist frei wählbar – sowie die Lebenszeit des Tokens am Server enthält.
2. Der Client merkt sich im Browser-Speicher (`localStorage`) das JWT. Die enthaltenen Informationen können decodiert werden. Anhand dieses kann am Browser erkannt werden ob das Token am Server noch gültig ist und evtl. bereits vom Browser aus entschieden werden ob der Zugriff auf den Web-Service noch möglich ist.
3. Über einen `HttpInterceptor` wird in der Angular-App bestimmten Requests auf dem Web-Service das JWT beigefügt. Der Interceptor wird im `AppModule` konfiguriert.
4. Der Server wertet dann automatisch (!) dieses JWT aus und erlaubt oder verbietet den Zugriff. Wird beispielsweise kein JWT mitgeschickt dann wird der Fehler 401: `Unauthorized` geworfen. Auch bei einem JWT bei dem die Lebenszeit überschritten ist, wird derselbe Fehler geworfen. Bemerkenswerterweise kontrolliert hier der Server die Lebenszeit jedes JWT das irgendwann einmal von ihm generiert wurde. Am Server kann auch erkannt werden, von welchem Benutzer der Request stammt (siehe hinten).

Programmierung des Angular-Clients `JwtAuthService`

1. Folgende Pakete müssen installiert werden:

   ```
   npm install jwt-decode moment
   ```

   `jwt-decode` Zum Decodieren des JWTs

   `moment` Zur Verarbeitung von Zeit- und Sekundenangaben

2. Erstellen des Services `JwtAuthService`:

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { lastValueFrom } from 'rxjs';

// Zum Dekodieren des JWTs
import { jwtDecode } from 'jwt-decode';
// Um mit Zeitangaben rechnen zu können
import * as moment from 'moment';

const URL = 'http://localhost:8080';

// Definiere die Inhalte meiner Payload
interface MyPayload {
  username: string;
  exp: number
}
```

```
@Injectable({ providedIn: 'root' })
export class JwtAuthService {

  constructor(private httpClient: HttpClient) { }

  async login(username: string, password: string) {
    try {
      // Holen des JWT vom Server
      const result = await lastValueFrom(this.httpClient
        .post<{ jwt: string }>(`${URL}/movie/login`,
          { username: username, password: password }));
      // Schreibe codiertes JWT in LocalStorage
      localStorage.setItem('jwt', result.jwt);
    } catch (error) {
      // Wenn Benutzer sich nicht authentifizieren konnte
      this.logout();
      throw error;
    }
  }

  logout() {
    localStorage.removeItem('jwt');
  }

  isLoggedIn() {
    const jwt = localStorage.getItem('jwt');
    if (jwt == null) {
      return false;
    } else {
      // Dekodiere JWT um die Verfallsdatum (exp) zu ermitteln
      const jwtDecodet = jwtDecode<MyPayload>(jwt);
      return moment().isBefore(moment((jwtDecodet.exp ? jwtDecodet.exp : 0) * 1000));
    }
  }
}
```

HINWEIS: Über `jwtDecodet.username` kann auf den Benutzernamen des eingeloggten Benutzers zugegriffen werden.

3. Erstellen des Interceptors `AuthInterceptor`:

```
import { HttpHandler, HttpInterceptor, HttpRequest } from "@angular/common/http";

export class AuthInterceptor implements HttpInterceptor {

  intercept(request: HttpRequest<any>, next: HttpHandler) {
    if (request.url !== 'http://localhost:8080/movie/login') {
      // Beim Login wird kein JWT mitgeschickt
      const authToken = localStorage.getItem('jwt');
      request = request.clone({ setHeaders: { Authorization: 'Bearer ' + authToken } });
    }
    return next.handle(request);
  }
}
```

4. Ändern des `AppModules`:

```
@NgModule({
  declarations: [ … ],

  imports: [ … ],

  providers: [
    {
      provide: HTTP_INTERCEPTORS,
      useClass: AuthInterceptor,
      multi: true
    }
  ],
  bootstrap: [ AppComponent ]

  entryComponents: [ … ]
})
export class AppModule { }
```

Programmieren des Servers

1. Folgende Pakete müssen installiert werden:

```
npm install express-jwt jsonwebtoken
```

express-jwt	Zur Entgegennahme des JWTs bei jedem Request an den Server

`jsonwebtoken` Zum Kodieren des JWTs

2. Aussehen von `index.js`:

```
const express = require('express');
const app = express();

const bodyParser = require('body-parser');
app.use(bodyParser.json());

…
const movieRouter = require('./movie/movie.router.js');
app.use('/movie', movieRouter);
app.get('/', (request, response) => response.redirect('/movie'));

// Standard Error-Handler
app.use((error, request, response, next) => {
  if (error.name === 'UnauthorizedError') {
    response.status(401).send(error);
  } else
    response.status(500).send(error);
});

app.listen(8080, () => console.log('Web-Service listen on port 8080'));
```

3. Aussehen von `movie.router.js`:

```
…
// Um Endpunkt absichern zu können
const { expressjwt } = require('express-jwt');

const PASSWORD = 'secret';
const ALGORITHM = 'HS256';

router.get('/',
  expressjwt({ secret: PASSWORD, algorithms: [ ALGORITHM ] }), listAction);
router.get('/:id', expressJwt({ … }), viewAction);
router.post('/', expressJwt({ … }), insertAction);
…
router.post('/login', loginAction);
```

4. Aussehen von `movie.controller.js`:

```
…
// Um JWT erstellen zu können
const jwt = require('jsonwebtoken');

// Sekunden der Lebenszeit eines Tokens
const EXPIRES_IN = 1000;
const PASSWORD = 'secret';
const ALGORITHM = 'HS256';

function loginAction(request, response) {

  const user = request.body;

  movieModel.getUser(user.username, user.password)
    .then(result => {
      const jwtToken = jwt.sign({ username: result.username }, PASSWORD,
        { expiresIn: EXPIRES_IN, algorithm: ALGORITHM });
      response.send({ jwt: jwtToken });
    })
    .catch(error => response.status(401).send('unauthorized'));
}
```

Beim erfolgreichen Login wird ein JWT erstellt, das Daten (die sog. Payload) aufnehmen kann. Im obigen Fall wird als Payload der Benutzername eingefügt. Diese Payload kann am Client wieder decodiert werden.

WICHTIG: Bei einem Request mit gültigem JWT wird vom Server der Request automatisch um das JSON-Objekt `auth` ergänzt, welches unter anderem die Payload des JWTs enthält – im obigen Fall `username`. Dadurch kann erkannt werden, welcher Benutzer den Request abgesetzt hat, z.B. `auth: { username: 'sepp', iat: 1706112513, exp: 1706113513 }`.

5. Aussehen von `movie.model.js`:

```
…
const crypto = require('crypto');

async function getUser(username, password) {
```

```
  if (!username || !password) {
    return Promise.reject('User not set');
  } else {
    try {
      const database = new Database(connectionProperties);

      const sql = `
        SELECT id, username, firstname, lastname
          FROM users
          WHERE username = ? AND passwordhash = ?;
      `;

      const passwordHash = crypto.createHash('sha256')
        .update(password)
        .digest('hex');

      const result = await database.queryClose(sql, [ username, passwordHash ]);

      return !result || result.length === 0 ?
        Promise.reject('User not found') : Promise.resolve(result[0]);

    } catch (error) {
      return Promise.reject('Database error');
    }
  }
}
```

Links

https://medium.com/better-programming/angular-http-interceptors-what-are-they-and-how-to-use-them-52e060321088

https://blog.angular-university.io/angular-jwt-authentication/

https://onthecode.co.uk/decode-json-web-tokens-jwt-angular/