ANHANG

17 Sourcecode

17.1 Main.ts

```
import jwt from 'jsonwebtoken';
import dotenv from 'dotenv';
import mongodb from "mongodb";
import bcrypt from 'bcryptjs';
import express from 'express';
import { Role } from './Role.js';
import { User } from './User.js';

dotenv.config();

const DB_URL = `mongodb://${process.env.DB_USER}:${process.env.DB_PASS-
WORD}@${process.env.DB_HOST}:${process.env.DB_PORT}?authMechanism=${pro-
cess.env.DB_AUTHMECHANISM}&authSource=${process.env.DB_DATABASE}`;
const DB_CLIENT = new mongodb.MongoClient(DB_URL)
const DB_CONNECTION = DB_CLIENT.connect()

var app = express();
let port: number = parseInt(process.env.PORT);

app.get("/signup", (req, res, next) => {
    let userName = <string>req?.query?.UserName;
    let userEmail = <string>req?.query?.Email;
    let userPassword = <string>req?.query?.Password;
    let userRepeatPassword = <string>req?.query?.RepeatPassword;

    if(userName == null || userName == ""){
        res.status(422).json({ message: "Error: Field \"UserName\" is re-
quired." });
    }
    else if(userEmail == null || userEmail == "" || !validateEmail(userE-
mail)){
        res.status(422).json({ message: "Error: Field \"Email\" is required
and must be valid." });
    }
    else if(userPassword == null || userPassword.length < 6){
        res.status(422).json({ message: "Error: Field \"Password\" must be at
least 6 characters long." });
    }
    else if(userPassword != userRepeatPassword){
        res.status(422).json({ message: "Error: Field \"RepeatPassword\" must
match Field \"Password\"." });
    }
    else{
        DB_CONNECTION.then(() => {
```

```
            const authDB = DB_CLIENT.db("auth");
            const usersColl = authDB.collection('users')

            usersColl.count( { name: escape(userName) }, (error: any, count:
number) => {
                if(error){
                    res.status(500).json({ message: "Error: Could not create
new user. Please try again later." });
                }
                else if(count != 0){
                    res.status(422).json({ message: "Error: A user with this
name already exists. Please choose a different name." });
                }
                else{
                    usersColl.count( { email: escape(userEmail) }, (error:
any, count: number) => {
                        if(error){
                            res.status(500).json({ message: "Error: Could not
create new user. Please try again later." });
                        }
                        else if(count != 0){
                            res.status(422).json({ message: "Error: A user
with this Email already exists. Please use a different email." });
                        }
                        else{
                            bcrypt.hash(userPassword, parseInt(pro-
cess.env.SALT), (err: any, hash: any) => {
                                if(err){
                                    res.status(500).json({ message: "Error:
Could not create new user. Please try again later." });
                                }
                                else{
                                    DB_CONNECTION.then(() => {
                                        const authDB = DB_CLIENT.db("auth");
                                        const rolesColl = authDB.collec-
tion('roles')

                                        rolesColl.findOne({name: "Default"},
(err: any, result: any) => {
                                            if(err){
                                                res.status(500).json({ mes-
sage: "Error: Could not create new user. Please try again later." });
                                            }
                                            else if(result == undefined || re-
sult == null || result == ""){
                                                res.status(500).json({ mes-
sage: "Error: Could not create new user. Please try again later." });
                                            }
                                            else{
```

```
                                                var role = new Role(result);
                                                if(role.id == null ||
role.name == null || role.permissions == null){
                                                    res.status(500).json({
message: "Error: Could not create new user. Please try again later." });
                                                }
                                                else{
                                                    var user = new User(es-
cape(userName), escape(userEmail), hash, role.permissions)
                                                    DB_CONNECTION.then(() => {
                                                        const valueToInsert =
{ name: user.name, email: user.email, hash: user.hash, permissions: user.per-
missions }
                                                        const authDB = DB_CLI-
ENT.db("auth");
                                                        const usersColl =
authDB.collection('users')
                                                        usersColl.inser-
tOne(valueToInsert, (err: any, result: any) => {
                                                            if(err){
                                                                res.sta-
tus(500).json({ message: "Error: Could not create new user. Please try again
later." });
                                                            }
                                                            else{
                                                                let token =
jwt.sign({ user }, process.env.JWTSECRET, { algorithm: 'HS256', expiresIn:
'1h' });
                                                                res.sta-
tus(200).json({ message: "Successfully created account and logged in.", token
});
                                                            }
                                                        })
                                                    });
                                                }
                                            }
                                        });
                                    }
                                );
                            }
                        );
                    }
                );
            });
        });
    }
});
```

```
app.get("/login", (req, res, next) => {
    let userName = <string>req?.query?.UserName;
    let userEmail = <string>req?.query?.Email;
    let userPassword = <string>req?.query?.Password;

    if((userName == null || userName == "") && (userEmail == null || userEmail
== "")){
        res.status(422).json({ message: "Error: At least one of fields
\"UserName\" and \"Email\" is required." });
    }
    else if(userPassword == null || userPassword == ""){
        res.status(422).json({ message: "Error: Field \"Password\" is re-
quired." });
    }
    else if(userName != null && userName != ""){
        //Log in with username
        DB_CONNECTION.then(() => {
            const authDB = DB_CLIENT.db("auth");
            const usersColl = authDB.collection('users')

            usersColl.findOne( { name: escape(userName) }, (error: any, re-
sult: any) => {
                if(error){
                    res.status(500).json({ message: "Error: Could not log in.
Please try again later." });
                }
                else if(result == null || result == undefined || result == ""
|| result._id == null || result._id == undefined || result._id == "" || re-
sult.name == null || result.name == undefined || result.name == "" || re-
sult.email == null || result.email == undefined || result.email == "" || re-
sult.hash == null || result.hash == undefined || result.hash == "" || re-
sult.permissions == null || result.permissions == undefined){
                    res.status(422).json({ message: "Error: Incorrect username
or password." });
                }
                else{
                    var user = new User(result.name, result.email, re-
sult.hash, result.permissions);
                    user.id = result._id;
                    bcrypt.compare(userPassword, user.hash, function(err: any,
bcResult: boolean){
                        if(err){
                            res.status(500).json({ message: "Error: Could not
log in. Please try again later." });
                        }
                        else{
                            if(bcResult){
```

```
                                let token = jwt.sign({ user }, pro-
cess.env.JWTSECRET, { algorithm: 'HS256', expiresIn: '1h' });
                                res.status(200).json({ message: "Successfully
logged in.", token });
                            }
                            else{
                                res.status(422).json({ message: "Error: Incor-
rect username or password." });
                            }
                        }
                    });
                }
            });
        });
    }
    else{
        if(userEmail == null || userEmail == "" || !validateEmail(userEmail)){
            res.status(422).json({ message: "Error: Field \"Email\" is not
valid." });
        }
        else{
            //Log in with email
            DB_CONNECTION.then(() => {
                const authDB = DB_CLIENT.db("auth");
                const usersColl = authDB.collection('users')

                usersColl.findOne( { email: escape(userEmail) }, (error: any,
result: any) => {
                    if(error){
                        res.status(500).json({ message: "Error: Could not log
in. Please try again later." });
                    }
                    else if(result == null || result == undefined || result ==
"" || result._id == null || result._id == undefined || result._id == "" || re-
sult.name == null || result.name == undefined || result.name == "" || re-
sult.email == null || result.email == undefined || result.email == "" || re-
sult.hash == null || result.hash == undefined || result.hash == "" || re-
sult.permissions == null || result.permissions == undefined){
                        res.status(422).json({ message: "Error: Incorrect
email or password." });
                    }
                    else{
                        var user = new User(result.name, result.email, re-
sult.hash, result.permissions);
                        user.id = result._id;
                        bcrypt.compare(userPassword, user.hash, function(err:
any, bcResult: boolean){
                            if(err){
```

```
                                res.status(500).json({ message: "Error: Could
not log in. Please try again later." });
                            }
                            else{
                                if(bcResult){
                                    let token = jwt.sign({ user }, pro-
cess.env.JWTSECRET, { algorithm: 'HS256', expiresIn: '1h' });
                                    res.status(200).json({ message: "Success-
fully logged in.", token });
                                }
                                else{
                                    res.status(422).json({ message: "Error:
Incorrect email or password." });
                                }
                            }
                        });
                    }
                });
            });
        }
    }
});

app.get("/checkpermission", (req, res, next) => {
    let token = <string>req?.query?.SessionToken;
    let permission = <string>req?.query?.Permission;

    if(permission == null || permission == ""){
        res.status(422).json({ message: "Error: Field \"Permission\" is re-
quired." });
    }
    else{
        let jwtValid = isTokenValid(token);
        if(!jwtValid.success){
            res.status(422).json({ message: "Error: Field \"SessionToken\" is
invalid. The current session might have expired." });
        }
        else{
            DB_CONNECTION.then(() => {
                const authDB = DB_CLIENT.db("auth");
                const usersColl = authDB.collection('users')
                usersColl.findOne( { name: jwtValid.user.name }, (error: any,
result: any) => {
                    if(error){
                        res.status(500).json({ message: "Error: Could not
check permission. Please try again later." });
                    }
```

```
                    else if(result == null || result == undefined || result ==
"" || result._id == null || result._id == undefined || result._id == "" || re-
sult.name == null || result.name == undefined || result.name == "" || re-
sult.email == null || result.email == undefined || result.email == "" || re-
sult.hash == null || result.hash == undefined || result.hash == "" || re-
sult.permissions == null || result.permissions == undefined){
                        res.status(500).json({ message: "Error: Could not
check permission. Please try again later." });
                    }
                    else{
                        var user = new User(result.name, result.email, re-
sult.hash, result.permissions);
                        user.id = result._id;

                        var hasPermission = false;
                        for (let p of user.permissions) {
                            if(p[permission] != null && p[permission]){
                                hasPermission = true;
                            }
                        }

                        res.status(200).json({ message: "Permission status of
permission \"" + permission + "\" is: " + hasPermission , hasPermission });
                    }
                });
            });
        }
    }
});

function escape(message: string){
    if(message == null && message == undefined){
        return "";
    }
    message = message.toString().replace(/</g, "&lt;").replace(/>/g,
"&gt;").replace(/"/g, "&#34;").replace(/'/g, "&#39;").replace(/`/g,
"&#96;").replace(/\(/g, "&#40;").replace(/\)/g, "&#41;").replace(/\//g,
"&#47;").replace(/\\/g, "&#92;").replace(/\[/g, "&#91;").replace(/\]/g,
"&#93;").replace(/\{/g, "&#123;").replace(/\}/g, "&#125;").replace(/\|/g,
"&#124;").replace(/\~/g, "&#126;");
    return message.trim();
}

function isTokenValid(token: string): any{
    var message = null;
    jwt.verify(token, process.env.JWTSECRET, (err: any, payload: any) => {
        if(err){
            message = { success: false };
```

```
        }else{
            message = { success: true, user: payload.user };
        }
    });
    return message;
}

function validateEmail(email: string)
{
    return /^(\w+([\.-]?\w+)*)@(\w+([\.-]?\w+)*(\.\w{2,4})+)$/.test(email);
}

app.listen(port, function(){
    console.log(`Server listening at *:${port}`);
});

process.on('SIGINT', signal => {
    console.log(`Process has been manually interrupted`);
    process.exit(0);
});

process.on('exit', exitCode => {
    console.log(`Process exited with code ${exitCode}`);
});
```

17.2 User.ts

```
import { Role } from './Role.js';

class User{
    public id: any
    public name: string
    public email: string
    public hash: (string | null)
    public permissions: []

    constructor(name: string, email: string, hash: (string | null), permis-
sions: []){
        this.name = name;
        this.email = email;
        this.hash = hash;
        this.permissions = permissions;
    }

    setPermissionFromRole(role: Role): void{
        this.permissions = role.permissions;
    }
    setOrAddPermission(permissionName: string, value: boolean): void{
        //TODO
    }
    removePermission(permissionName: string): void{
        //TODO
    }
}
export { User }
```

17.3 Role.ts

```
class Role{
    public id: any
    public name: string
    public permissions: []

    constructor(roleJson: any){
        if(roleJson != undefined && roleJson != null && roleJson._id != unde-
fined && roleJson._id != null && roleJson.name != undefined && roleJson.name
!= null && roleJson.permissions != undefined && roleJson.permissions != null){
            this.id = roleJson._id;
            this.name = roleJson.name;
            this.permissions = roleJson.permissions;
        }
    }

    setOrAddPermission(permissionName: string, value: boolean): void{
        //TODO
    }
    removePermission(permissionName: string): void{
        //TODO
    }
}
export { Role }
```

ANHANG 2

18 Dokumente

Es gibt keine Dokumente in diesem Projekt.