2. SEMESTER - PROJEKT 2

Gruppe 4 - Artur, Anna, Dalena, Kasper & Marco

INDHOLDSFORTEGNELSE

INDLEDNING

I dette projekt, har vi lavet et quiz spil. For at kunne spille spillet, kræver det en bruger, som brugeren skal logge ind på. Spillet har forskellige kategorier, hvor vi har valgt kategorierne 'Film og TV' og 'Børn', som spilleren kan vælge.
Brugeren kan selv gå ind og tilføje spørgsmål, som andre bruger også kan quizze på.

PROBLEMFORMULERING

Hvordan kan man lave et quiz spil, der har et sikkert login og har en database samt API, hvor brugeren kan tilføje og slette elementer (spørgsmål og bruger)?

AFGRÆSNING

Vi har i denne opgave fokuseret på, at få et login system til at virke, sådan at brugeren kan logge ind og ud, samt oprette en ny bruger. Dertil skal der kunne vises quizzer fra databasen og kunne oprette en quiz. Det vil sige, at vi i gruppen vælger at nedprioritere en statistikside, hvor brugeren kan se, hvor mange quizzes der er vundet ud af hvor mange. Dertil er der også nedprioriteret at validere på de forskellige quizzes, når brugeren quizzer, da vi har fokuseret på at få de forskellige ting sat op med databasen som førsteprioritet.

METODER

Passport.js

For at gøre login mere sikkert, implementeres der en authentication, hvor vi har valgt passportjs som er en middleware til NodeJS. Passportjs har sessions er en tidsbegrænsning som starter når man logger på, og når tiden er udløbet, kræver

det at brugeren logger på igen. Dette er en måde at gøre det sikkert for brugerne da de skal indtaste deres login informationer for at kunne fortsætte.
(https://www.passportjs.org/ n.d.)

Bcrypt.js

Ligeledes har vi brugt bcrypt bibliotek for at sikre brugerens login. Ved brug af biblioteket kan vi hashe og salte brugerens password.
(https://www.npmjs.com/package/bcryptjs 2017)

Pico.css

For at sætte websitet op hurtigere uden at bruge for meget tid på css, har vi brugt pico.css som er en css framework. Dette framework er meget simpelt og kræver ikke en masse classes, da det primært bruger html tags til css.
(Larroche n.d.)

.ejs

Ejs står for "embedded JavaScript templating", og dette lader os skrive html med alminde JavaScript i vores views.
(Eernisse n.d.)

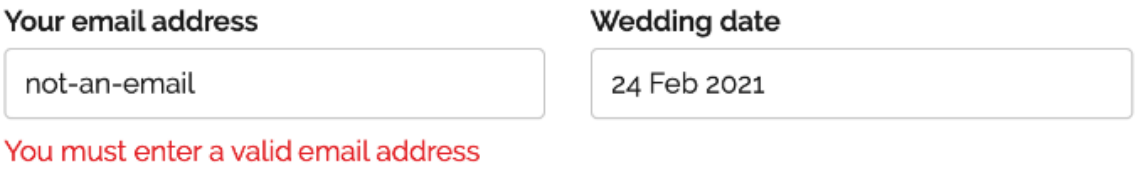
Datasikkerhed

Sanitizing og validation

Sanitizing af input er en måde at fjerne usikker symboler eller bogstaver, som hackeren kan benytte sig af, eksempelvis når en hacker forsøger at bruge SQL injection. Måden det fungere på, er når en brugeren indtaster et login eller

password, vil sanitizing derefter gå ind i inputfeltet og fjerne symboler eller tekster og på den måde bliver inputtet gjort sikkert. (Maury 2023)

Validering af input er en anden måde at sørge for, at input feltet er sikkert imod SQL injection. Eksempelvis kan man validere login informationer, for at skabe ekstra sikkerhed imod et angreb. Måden validering fungere på, er at begrænse, hvad brugeren kan indtaste, eksempelvis med nogle symboler, bogstaver og tal. Validering er også brugervenligt, da det hjælper brugerne med at se, hvad det er de gør forkert (Hoyt n.d.). (se billedet herunder)



Hashing

Hashing omsætter data i en envejsprocess, typisk et password. I dette projekt bruges det også til password, hvor det bliver omsat til en string af forskellige værdier af en bestemt længde - alt efter hvilken algoritme man vælger. I databasen bliver den hashede værdi gemt i stedet for det reelle password, og derfor skal der, når brugeren logger ind, også tjekkes om inputtet passer på den hashede værdi i databasen. (Lake 2018)

Salting

Salting bruges ved hashing, hvor der tilføjes en ekstra tilfældig unik string til passwordet inden det bliver hashet. (Lake 2018)

RESEARCH

For at lave en quiz skal vi have nogle spørgsmål. Vi har derfor undersøgt, hvilke typer spørgsmål og kategorier der skal være. Brugerne kommer selv til at kunne tilføje deres egne spørgsmål. (Danny n.d.) , (Cullen n.d.)

KONSTRUKTION

Database

I denne opgave har vi valgt at opsætte databasen med MongoDB som bliver hostet ved Mongo Atlas. I databasen har vi en kollektion til users og en til quizzes.

Users

I kollektionen med users, består en user af et id, username, password, email, avatar og et array med brugerens quiz score (se billede herunder). Her skal id og password være privat, så andre bruger ikke kan få adgang til dette.
En user skal selv have mulighed for at opdatere, username, password og email, med en put request til databasen. Derudover skal der også være mulighed for at kunne slette sin egen bruger.

Quizzes

Kollektionen med quizzes består af objekter som indeholder den enkelte quiz. Denne quiz har et id, titel, kategori, og et array af spørgsmål. Dette array består af objekter, som indeholder et spørgsmål og et array med svarmuligheder. I arrayet med svarmuligheder er en svarmulighed et objekt, som har navnet som string og en boolean for om svaret er det korrekte til spørgsmålet eller ej.
Dette er også de ting som brugeren skal taste ind, når de skal oprette en ny quiz, som laver et post request til databasen (se struktur herunder).

Opsætning og struktur

Dette projekt udvikles med express.js og node.js. Vi opretter et projekt via command line, der opsætter express projektet for os med .ejs som templates til views. For at spare tid vælger vi at bruge et css framework, som der blot linkes til i vores head på de forskellige views.
Vi opsætter view til alle sider, der skal være på siden. Dette er login, signup, tilføj quiz, brugerside, selve quiz siden og en statistikside. Ved login og signup blive der blot vist input felter, som brugeren skal udfylde for enten at logge ind eller oprette sig som bruger. På de andre sider ses der en navigation, hvor brugeren kan navigere imellem de forskellige quizzes, samt komme til at se statistikker og de quizzes man har gennemført eller til en brugerside, hvor man kan redigere sine info. Dertil kan man selvfølgelig også logge ud af sin konto.
På hver af siderne findes selvfølgelig også det dertilhørende information eller inputfelter.
Vi opsætter routers for de forskellige sider for at håndtere forespørgslerne fra brugeren. Vi håndtere get, post og delete forespørgslerne samlet fra routers.js for de forskellige undersider. For selve quizzen har vi createQuiz.js, loadQuiz.js og deleteQuiz.js. I createQuiz.js require vi quizSchema for at kunne tilgå de forskellige egenskaber i objektet. For at kunne tilskrive et nyt id til hver enkelt ny quiz der bliver lavet tager den det sidste nye objekt i Arrayet i databasen, ved hjælp af slice(-1) og øger id'et, som ses i lastIdCount variable, hvor vi tilskriver

quizid + 1. Når vi så sætter en ny quiz op, sætter vi counteren til quizId, for at tilskrive hver enkelt ny quiz med et øget id.
For at loade quiz dataen ind fra databasen i loadQuiz.js, laver vi en variabel som holder informationen fra schemaet.
I deleteQuiz.js håndterer vi sletning af en quiz, ved at finde id'et for quizzen.

Opsætning af schema

User

```
const passportLocalMongoose = require('passport-local-mongoose');
const mongoose = require("mongoose");

const userSchema = new mongoose.Schema( definition: {
    username: { type: String, required: true, unique: true },
    password: { type: String, required: true, minlength: 5 },
    email: { type: String, required: true, unique: true },
    avatarUrl: {type: String, required:false},
    quizScore: { type: Array }
}, options: {collection: 'Users'});

userSchema.plugin(passportLocalMongoose)

module.exports = User = mongoose.model( name: "user", userSchema);
```

User schema består af forskellige informationer om brugeren, såsom username, password, email, avatarUrl og quizScore. Username og email skal være unikke, som bliver tjekket senere i userCreate route. quizScore er et array, som skal indeholde points af forskellige quizzes. Når userSchema bliver defineret, tilføjer vi også et ekstra argument, som beskriver kollektionen i databasen, hvor bruger informationen skal sendes. I dette tilfælde, så sender vi vores user til 'Users' kollektion. I skemaet bliver der også brugt passport plugin: 'passport-local-mongoose', der gør det muligt at koble brugeren med autentificering værktøj, dvs. passport.

Quiz

Først har vi en variable med require mongoose, som laver en instance af mongoose der er en npm pakke. Mongoose hjælper med at gøre det lettere at tilgå til databasen. Derefter laves der en variable med navnet quizSchema, som opretter en ny mongoose schema med et objekt i. Objektet indeholder nogle egenskaber: answerArray som har typen array, question som har typen objekt, category som har typen string, createdBy som har typen string og createdDate som har typen Date. Det skal bruges når vi opretter en ny quiz. Schema'et bliver en skabelon til, hvad quizzet skal indeholde. Sidst har den en collection som kaldes Quizzes, som gør det sikkert at vi får fat i den rigtige database, da vi har en til user og en til quiz. Schemaet bliver eksporteret og får navnet 'quiz', hvor vi kobler quizSchema til. Vi kan nu lave quiz, hvor vores schema bliver skabelonen. Schemaet bruger vi i loadQuiz og createQuiz under routes.

```
const quizSchema = new mongoose.Schema({
    type: Object,
    quizId: {
        type: Number,
    },
    category: {
        type: String,
    },
    questions: [{
        type: Object,
        quizName: {
            type: String,
            required: true
        },
        answerArray: [
        {
            type: Object,
            answer: {
                type: String,
                required: true
            },
            answerIsCorrect: {
                type: String,
            }
        }
        ]
    }]
}, {collection: "Quizzes"});
```

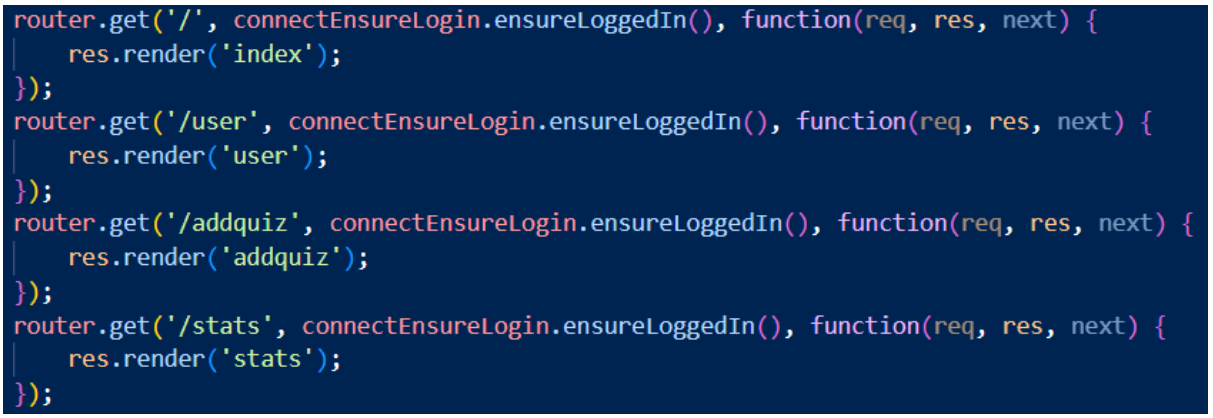
Datasikkerhed

Passportjs

Vi bruger passport.js som middlewear for autentificering af brugeren.
I routers.js for login, har vi gjort sådan at, hvis ikke den kan autentificerer brugeren, vil brugeren blive ledt tilbage til login siden og ellers ført videre til startsiden.

```
router.post(
    '/login',
    passport.authenticate('local', {
        failureRedirect: '/login',
        successRedirect: '/',
    })
);
```

For at sikre at brugeren er logget ind for at kunne navigere rundt på siden, har vi brugt connectEnsureLogin, så hvis ikke brugeren er logget ind kan de ikke tilgå siderne.



```
router.get('/', connectEnsureLogin.ensureLoggedIn(), function(req, res, next) {
    res.render('index');
});
router.get('/user', connectEnsureLogin.ensureLoggedIn(), function(req, res, next) {
    res.render('user');
});
router.get('/addquiz', connectEnsureLogin.ensureLoggedIn(), function(req, res, next) {
    res.render('addquiz');
});
router.get('/stats', connectEnsureLogin.ensureLoggedIn(), function(req, res, next) {
    res.render('stats');
});
```

i app.js har vi brugt serialiseUser og deserializeUser funktionerne, som gemmer og genopretter brugerens informationer i en session. SeriliazeUser kører hver gang brugeren logger ind og bestemmer, hvilken information der skal gemmes i session. DeserializeUser kører hver gang der sendes en request til serveren og henter informationen fra session for at gøre det tilgængeligt for brugeren.

```
// Passport Local Strategy
passport.use(User.createStrategy());

// To use with sessions
passport.serializeUser(User.serializeUser());
passport.deserializeUser(User.deserializeUser());
```

Som det ses i koden her håndterer vi session på applikationen, hvor vi f.eks. har sat sessionen til at være 1 time, velvidende om at session normalt skal være noget længere.

```
app.use(session({
    secret: process.env.SESSION_SECRET,
    resave: false,
    saveUninitialized: true,
    cookie: { maxAge: 60 * 60 * 1000 }
}));
```

Hashing og salting

Til at starte med brugte vi bcrypt's bibliotek til at hashe og salte brugernavn og password.
Når brugeren opretter sig med brugernavn og password, har vi i createUser.js gjort brug af biblioteket, hvor vi generere en salt, hasher passwordet og tilskriver saltet.

```
// hash user password
const salt = await bcrypt.genSalt(10);
user.password = await bcrypt.hash(password, salt);
await user.save();
res.json(user)
```

Når brugeren så logger ind efter at have oprettet sig, sammenligner vi i loginUser.js om passwordet stemmer overens ved brug af bcrypt biblioteket igen.

```
const isMatch = await bcrypt.compare(password, user.password);
```

Hashing og Salting med passport.js

Men vi fandt ud af at passport.js havde dens egen indbyggede måde at hashe og salte. Ved brug af .register i createUserPassport.js, kan vi bruge password som argument når brugeren opretter sig. Dette gør at passwordet automatisk bliver hashet og saltet i databasen.

```
User.register(new User({ email, username, avatarUrl, quizScore: [] }), password, function (err, user) {
    if (err) {
        res.json({ success: false, message: "Your account could not be saved. Error: " + err });
    }
    else {
        req.login(user, (err) => {
            if (err) {
                res.json({ success: false, message: err });
            }
            else {
                res.json({ success: true, message: "Your account has been saved" });
            }
        });
    }
})
```

I loginUserPassport.js, kan vi så bruge .authenticate, som kigger på om det passwordet brugeren logger ind med, stemmer overens med det i databasen.

```
module.exports = async (req, res) => {
    const { username, password } = req.body;

    if (!username) {
        res.json({ success: false, message: "Username was not given" })
    }
    else if (!password) {
        res.json({ success: false, message: "Password was not given" })
    }
    else {
        passport.authenticate("local",{ failureRedirect: '/login', successRedirect: '/addquiz' }, function (err, user, info)
            if (err) {
                res.json({ success: false, message: err });
            }
            else {
                if (!user) {
                    res.json({ success: false, message: "username or password incorrect" });
                }
                else {
                    console.log(info)
                    const token = jwt.sign({ userId: user._id, username: user.username }, process.env.JWT_SECRET, { expiresIn
                        if (err) throw err;
                        // res.json({ token });
                        res.redirect('/user')
                    });
```

PROCESEVALUERING

Som gruppe har vi fungeret godt sammen og har ikke haft nogle uoverensstemmelser. Vi har dog haft problemer og brugt lang tid på at få lavet, så brugeren selv kan lave quizzer. Vi havde problemer med vores quizSchema, men fandt ud af, at i og med vi bruger mongoose, er opsætningen af schemaet anderledes og mange timers problematik bundende ud i {}. Samtidig i processen blev databasen ved med at crashe efter alle de requests vi sendte afsted, hvilket gjorde at ventetiden for at kunne komme videre blev længere.

KONKLUSION

Vi har lavet projektet med NodeJS og sat det op med npx generate expressJS, hvor vi har brugt ejs som template. ejs filerne i vores view, viser siderne for selve quizzen, generere en quiz, login, signup, stats og brugerside. Vores database er sat op med MongoDB, som bliver hostet med Mongo Atlas. For at få dataen fra databasen har vi sat schemaer op for både quiz og bruger, som bliver importeret og brugt til håndtering af requests.
Til validering af user bliver brugt passport js, som også sørger for at hashe og salte password, når der bliver oprettet en bruger.

REFERENCER

Cullen, E. (n.d.) *100+ Fun General Knowledge Quiz Questions 2023* [online] available from <https://www.mentimeter.com/blog/audience-energizers/55-free-trivia-and-fun-quiz-question-templates> [15 March 2023]

Danny (n.d.) *40 Questions to Make You a Virtual Pub Quiz Master ‹ GO Blog | EF GO Blog* [online] available from <https://www.ef.com/wwen/blog/language/questions-virtual-pub-quiz/> [15 March 2023]

Eernisse, M. (n.d.) *EJS -- Embedded JavaScript Templates* [online] available from <https://ejs.co/#features> [15 March 2023]
Hoyt, B. (n.d.) *Don't Try to Sanitize Input. Escape Output.* [online] available from <https://benhoyt.com/writings/dont-sanitize-do-escape/> [15 March 2023]
https://www.npmjs.com/package/bcryptjs (2017) *Bcryptjs* [online] available from <https://www.npmjs.com/package/bcryptjs> [15 March 2023]
https://www.passportjs.org/ (n.d.) *Username & Password Tutorial* [online] available from <https://www.passportjs.org/tutorials/password/> [15 March 2023]
Lake, J. (2018) 'Encryption, Hashing, Salting: What's the Difference and How Do They Work'. [21 December 2018] available from <https://www.comparitech.com/blog/information-security/encryption-hashing-salting/> [15 March 2023]
Larroche, L. (n.d.) *Pico.Css · Minimal CSS Framework for Semantic HTML* [online] available from <https://picocss.com> [15 March 2023]
Maury, J. (2023) *How to Use Input Sanitization to Prevent Web Attacks* [online] available from <https://www.esecurityplanet.com/endpoint/prevent-web-attacks-using-input-sanitization/> [15 March 2023]