Projekt uge 11: Biblioteksystem

af **Rainbow Warriors** (Carina, Khava, Martin & Monika)

Indholdsfortegnelse:

Indledning

Til vores projekt i uge 11 skal vi udarbejde en webapp i form af et system, som man formentlig skal kunne bruge på et bibliotek. Det skal være muligt som bruger at gå ind og oprette sig selv med brugerid og password, og derudover skal man kunne registrere, låne, reservere og aflevere bøger ved brug af håndtering af data.

Det er et interessant og relevant projekt, som kommer til at involvere flere at de nye faglige kvalifikationer, vi har lært indtil videre i 2. semester, samt det, vi har lært i 1. semester.

Problemformulering

“Hvordan kan vi ved hjælp af bl.a. Node og Express pug lave en brugervenlig hjemmeside, hvor man kan oprette en bruger med tilhørende password og frit låne, reservere og aflevere bøger, samt en side, som udsteder bøder til dem, der ikke afleverer bøger i tide?”

Metodeovervejelser

Vores valg af metoder kommer mest af alt af, hvad vi har arbejdet med i undervisningen og den erfaring, vi har fået deraf. Fx har vi bl.a. overvejet, hvilken template engine, vi vil bruge, og noget frem til et valg ud fra, hvad vi før har arbejdet mest med - Pug.

Vi har selvfølgelig også benyttet os meget af Node til at teste vores kode på vores egen server i løbet af projektet, i stedet for at åbne det i browseren hele tiden. Express, som vi har brugt i forlængelse med Node, har dertil givet os en god struktur i vores filer, så de er meget lettere at komme til samt arbejde i inde i vores editors.

Som noget ret nyt har vi også lært at arbejde lidt med kryptering, herunder bl.a. Bcryptjs, som vi også har draget nytte af her i projektet. Mere om dette kommer vi ind på senere, men det er kort sagt et modul, man kan bruge til at hashe fx passwords. Det er en af de eneste metoder, vi hidtil har arbejdet med inden for kryptering, og for at være mest hurtige og effektive, vælger vi derfor bare at holde os til det, vi kender.

Derudover har vi oprettet et board på Trello og endnu en gang taget god brug af Google Drev til at skabe oversigt over opgaver og en tidsplan for os, da det har gjort hele arbejdsprocessen meget nemmere at overskue for os i løbet af ugen.

Research

Vi har måtte researche opsætningen af en web-app ved brug af node express og en tilhørende template. Hertil har vi researched metoden for opsætning efter best practice. Vi har researchet brugen af mongoDB, når vi skal bruge en no-sql database. MongoDB er godt dokumenteret, og der findes hjælpeværktøjer til at tilgå databasen, når man bruger Node. Derfor har vi også undersøgt mongoose.

Vi har kigget på Pug og dens tilhørende syntax, når man skriver en Pug-fil.
Pug er en template engine, som laver en skabelon af en HTML-side for os. Der findes flere forskellige template engines, men vi har valgt at bruge Pug, da vi ellers kun har erfaring med Dust, og det er vi knap så glade for at arbejde med, da den egentlig bare har skabt mere forvirring end gavn hos gruppen. Vi kan altså langt bedre lide at arbejde med Pug, så derfor har vi naturligvis truffet et valg ud fra det. Dokumentationen og layoutet for Pug er simpelthen bare bedre og mere overskueligt i vores øjne.

Pug er ligeledes lang mere veldokumenteret og brugt end Dust. Derfor er det nemmere for os at arbejde med. Med en hurtig søgning på google, får man feks mange sider med Pug og syntaks.

Analyse

Første punkt i ethvert projekt består stort set altid af planlægning og at få ideer til, hvordan resultatet af ens arbejde skal se ud i sidste ende. Vi har ikke været en undtagelse - dog har vi pga. projektets omfang og prioriteter ikke lavet skitser og wireframes til, hvordan vi ville have vores design til at se ud. Det har vi udviklet lidt efterhånden, imens vi kodede det hele.

Vi har først tænkt over hvert enkelte konkrete funktionalitet, som webappen skal have, og skrevet det ned for at skabe et bedre overblik for os selv. Det er en app med relevante muligheder og funktioner, som vi kender fra hverdagen, hvilket har gjort det spændende projekt at arbejde med. Vi har altså kort sagt planlagt ud fra noget pseudo-kode og fokuseret på, at få vores app til at gøre det simpelt at låne, reservere og registrere bøger, og i den process også gemme data i vores Mongo database.

Konstruktion

Vi har fået en projektmappe med til projektet, hvor strukturen er noget, som vi så har bygget løbende videre på i takt med, at vi har haft brug for at have flere filer. Strukturen kommer fra Express, som et en slags template til filer, så man nemt og enkelt kan lave en fin træstruktur til sine filer med Node.

Vi er startet med at lave skemaer og handlers til vores bøger, så vi allerede næsten fra starten af har kunne skrive bøger ind i vores database. Derefter har vi benyttet lidt den samme fremgangsmåde til at lave et skema og en handler til brugere, der vil oprette sig på hjemmesiden. I begge tilfælde har vi også lavet en form ved hjælp af Pug, hvor der er input felter til de informationer, der skal bruges.

Mongoose Schema

Mongoose er et værktøj til at tilgå MongoDB, som vi i dette tilfælde har benyttet til at holde på al vores data fra vores inputs. Mongoose er en slags udvidet version til MongoDB og bruges hovedsageligt i form af skemaer. Hvert skema kortlægges til en samling af MongoDB og definerer derefter formen på dokumenterne i samlingen.

Nedenfor ses et eksempel på et skema, vi har lavet ved hjælp af Mongoose. Her bruger vi skemaet til at indeholde info til bøger.

```
const mongoose = require('mongoose');

const bookcopiesSchema = mongoose.Schema ({
  bookid: Number,
  id: Number
});

module.exports = mongoose.model("Bookcopies", bookcopiesSchema, 'bookcopies')
```

Det data, vi i dette projekt arbejder med, involverer primært registrering af bøger og brugere. Derfor er det nødvendigt at gemme fx login data med et brugernavn og et password, så man altid kan komme ind på siden for at se på bøger, låne, aflevere, m.m. og ikke mindst informationer og bøgernes titler, forfattere, forlag og meget mere. Det har til tider været en stor udfordring at få vores database til at opfange netop de data, vi vil have den til, især efter implementering af vores login-system.

Her ses et lille udpluk fra en af vores mange øjeblikke i vores database, hvor tilfældige brugere er oprettet til vores egne test formål:

```
"__v" : 0,
"cpr" : 984545,
"currentpenalties" : 8,
"email" : "dfghj",
"firstname" : "ghbjh",
"lastname" : "jkkj",
"middlename" : "hjbb",
"newsletter" : "jnj",
"password" : "$2a$10$q9p9b1OePCZ1.Y3AqnI3hevC/QTYIHSgTjHCNcVBbabZrYSzoKHsi"


"_id" : ObjectId("6051f5475f48f84fe0bdde76"),
"id" : 5858,
"__v" : 0,
"cpr" : 454545,
"currentpenalties" : 500,
"email" : "carinausbeck@live.dk",
"firstname" : "Carina",
"lastname" : "Andresen",
"middlename" : "Usbeck",
"newsletter" : "n",
"password" : "$2a$10$1y7iOw9HjD6WgWnjJ6Qkc.Oy2DhvMvZgEWo6RobhEZOrQouL2Hk/K"
```


Registrering af bøger

Vi har lavet inputfelter til registrering af bøger i et *view* som vi har kaldt bookform. I vores router har vi lavet en get request til at vise siden og vi har lavet en post til at skrive data fra input felterne ind i databasen. Dette sker i filen handleBooks. Først etableres forbindelsen med mongoDB ved hjælp af mongoose. Derefter defineres objekter for author, published og disse bliver lagt ind i et nyt objekt som laves med metoden *new* da det er sådan strukturen er lavet i databasen. Derefter bruger vi mongoose metoden *create* sammen med skemaet, samt objektet som parameter og det skriver objektet ind i databasen. På samme vis har vi her en konstruktion som skriver antallet af eksemplarer af den pågældende bog ind i bookcopies collection.

```
let book = new Book({
    title: req.body.title,
    authors: author,
    copyright: req.body.copyright,
    edition: req.body.edition,
    published: published
});

 Book.create(book, function(error, savedDocument)
```

Hashing med Bcryptjs

Til at kryptere brugeres passwords med hashing har vi benyttet modulet Bcryptjs. Det kan være lidt kringlet at arbejde med, men efter en længere process med sved på panden, er det endelig kommet til at virke for os. Vi har en *handleUsers* fil, som

håndterer brugerne, ved at oprette et *user* objekt, hvorfra man får inputs fra brugeren. Derefter går Bcryptjs (hvis const vi har kaldt *bcrypt* her) så ind og hasher vores indtastede password, *pwd*. Se nedenstående screenshots for reference.

```
'use strict';
const bcrypt = require('bcryptjs');
const fs = require('fs').promises;
const handleuser = require('handleUser');
const saltRounds = 10;
let myPlaintextPassword = req.body.password;
// const UFILE = __dirname + '/../data/users.json';

bcrypt.hash(myPlaintextPassword, saltRounds, function(err, hash) {
    console.log(hash);
});
```

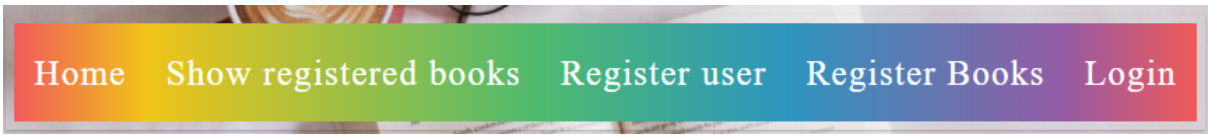
```
exports.postUsers = async function (req) {
    let chk = { id: req.body.id };  // check object for existence
    let user = new User({                    // create object in db-format
        id: req.body.id,
        password: req.body.password,
        cpr: req.body.cpr,
        currentpenalties: req.body.currentpenalties,
        email: req.body.email,
        firstname: req.body.firstname,
        middlename: req.body.middlename,
        lastname: req.body.lastname,
        newsletter: req.body.newsletter
    });
    let pwd = await bcrypt.hash(req.body.password, 10);
    console.log(pwd);
    req.body.password = pwd;
    console.log(req.body.password);
    try {
        let cs = await mon.upsert("localhost", "library", User, user, chk); // Tager fat i mongoose db
        return;
    } catch (e) {
        console.log(e);
    }
}
```

Login/logout

Når man ikke er logget ind på vores webapp, så har man ikke så mange interaktionsmuligheder. Hvis man derimod logger ind, får man flere muligheder i vores menu - og så har man også lov til bl.a. at reservere og låne bøger. Vi har haft en masse problemer med vores login, idet der var problemer med at aflæse kode og brugerid, som vi har lavet om flere gange i processen.

Ved hjælp af Node.js Express applikation har vi oprettet en layoutvisning, der er knyttet til enhver anden visning som inkluderet. Layoutet indeholder en overskrift og menu, hvor:

- brugeren kan tilmelde sig eller logge ind med brugernavn og adgangskode.
- efter bemyndiger vil brugeren til at få adgang til menuen.



Vi har indstillet den krævede attribut for begge knapper. Hvis nogen forsøger at login uden at have adgang med bruger-id og adgangskode, vises en fejl.

```
router.post('/', async function(req, res, next) {
await login.getLogin(req)
 .then( function (rc) {
  if (!rc)
   res.render('index', { title: 'Login', tf: "Login failed", returnCode: rc });
  else
   res.render('index', { title: 'Login', tf: "Logged in successfully",
   authenticated: req.session && req.session.authenticated, returnCode: rc });
   });
});
```

Der er to hovedfunktioner til authenticated:

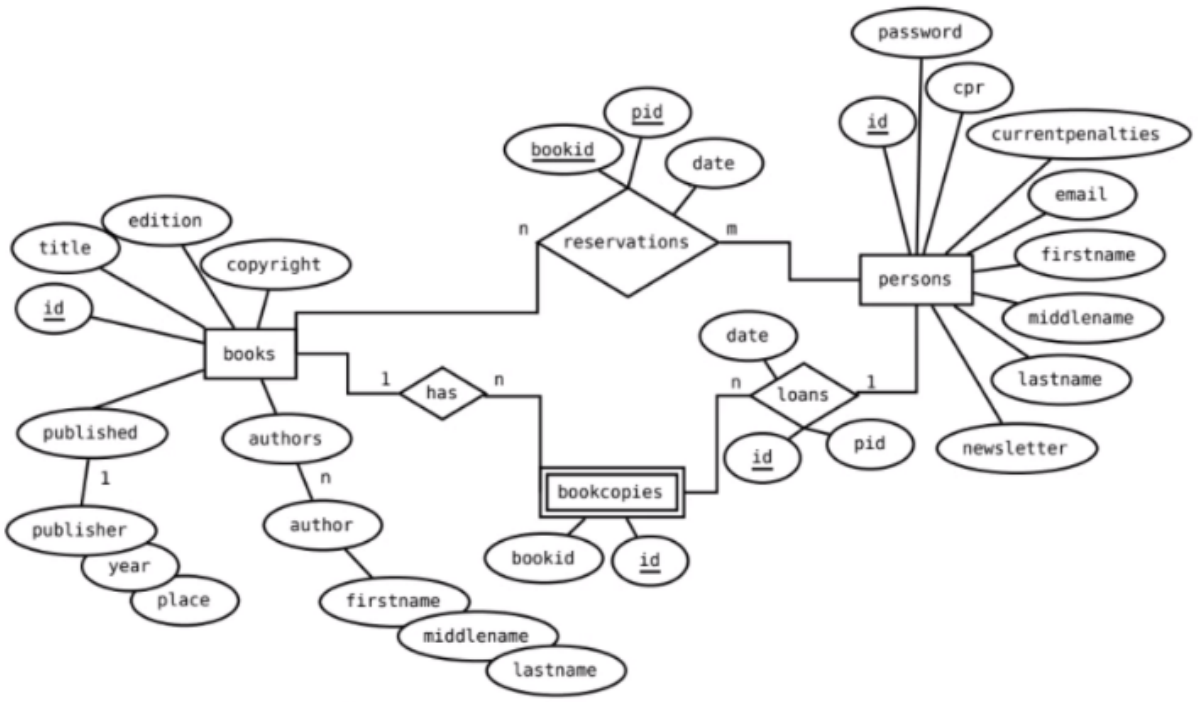
- opret ny bruger i database
- logind:
  - find brugernavn af anmodningen i databasen, hvis den findes
  - sammenligne adgangskode med adgangskode i database ved hjælp af bcrypt, hvis det er korrekt
  - returner brugerinformation

Vi har brugt Nodes oprindelige bcrypt til at hash-adgangskoder for registrerede brugere som en grundlæggende sikkerhedsfunktion.

Loans / lån

Noget af det sidste, vi har nået i projektet, er at udvikle vores lånesystem. Først har vi overvejet, hvordan det skal struktureres. Det er gjort ved at referere til nedenstående model, som Niels har vist os, hvor loans/lån tager et *id* fra *bookcopies*, hvorefter den tager et timestamp og et id fra den person der er logget ind, hvorefter den skriver disse over i databasens loans collection. Dette skulle foregå i en funktion som er routet med en *post request*.
Vi kom frem til at vi kunne have både lån og reservation af bøger i det samme view da en normal brugersituation ville være at man vil reservere bøger hvis der ikke er

nogen hjemme og hvis der er bøger tilgængelig for lån ville det ikke være nødvendigt at reservere den.



Derefter har vi kigget på noget af det kode, vi allerede har lavet, for at genbruge det, der giver mening - for at gøre det lettere for os selv og for at spare tid. Heriblandt har vi så primært benyttet kodestykker fra vores handleBooks funktion, som ligesom navnet siger, håndterer alle vores indtastede bøger. Det er også her, vores database går ind og finder data om bøgerne.
Når vi fremviser de bøger som er tilgængelige for udlån henter vi dem fra databasen ved hjælp af en *get request* med en mongoose forbindelse. Til udlån ville vi ligeledes hente antallet af eksemplarer af en given bog fra bookcopies. For at vi viser brugeren den rigtige status på antallet af eksemplarer skal der foretages et match mellem de ids der ligger i loans med dem fra bookcopies og trække dem fra når den viser antallet.

Evaluering af process

Vi har i denne omgang været gode til at hele tiden referere til vores Trello i løbet af hele projektugen, og har også haft en god arbejdsstruktur, som bl.a. har budt os at sidde det meste af dagen på Teams sammen for at overkomme de problemer, vi er rendt ind i løbende. Det gør vi, fordi problemer tit kan løses lidt hurtigere, hvis man har flere hænder og øjne på sagen. Det har været en hård process, men vi har fået en masse hands-on erfaring ud af det i sidste ende - især med arbejde inden for databaser.

Konklusion

Vi har nu fået en fungerende web-app som er et lille lokalt bibliotek. Den fungerer ved, at vi har en database, hvor vi har nogle collections med dokumenter. Vi kan så tilgå vores database i vores web-app ved hjælp af noget middleware, som er lavet til at kommunikere mellem Nodejs og MongoDB. Det er vores mongoose vi her bruger. Vi har derfor, fået funktioner i vores app, så man som bruger kan registrere sig som user med tilhørende userID og password. Passwordet bliver hashet, så man ikke kan tilgå det som plaintext. Vores brugere kan efter registrering logge ind på web-appen, hvor flere valgmuligheder i menuen også bliver vist. De kan registrere nye bøger, og de kan få vist de registrerede bøger. Når de er færdige på web-appen, så kan de igen logge ud, hvor de på den måde vil få færre muligheder igen.

Perspektivering

Det har været et spændende projekt for os at have i hænderne, da opgaverne skildrer det, som vi vil komme til at lave i en “virkelig” situation. Vi har lært en hel del om, hvordan man bygger en web-app med tilhørende database i Node Express.

Vi har ikke nået en fuldstændig færdig biblioteks-app, da tiden har været knap, og der har været meget at undersøge og prøve af undervejs. I den færdige app ville vi gerne have flere funktionaliteter.

I den fuldstændige version skal brugeren kunne låne bøger fra biblioteket. Her skal funktionaliteterne kunne tilgå database collectionerne bookcopies for at se hvor mange kopier der findes, loans for at se om der ligger nogle deri eller om de skal tilføjes, persons for at se hvem der låner. Når en bruger så har lånt en bog, skal der sættes en timestamp, og en timer på 30 dage skal i gang. Efter de 30 dage skal der påføres en bøde.

Til gengæld har vi nu en hel del erfaring med Node express, Pug, MongoDB og Mongoose. Det er både teoretisk men mest af alt hands-on erfaring, vi tager med os videre, som er helt uvurderlig.

Referencer

An Introduction to Mongoose for MongoDB and Node.Js (n.d.) available from <https://code.tutsplus.com/articles/an-introduction-to-mongoose-for-mongodb-and-nodejs--cms-29527> [15 March 2021]

Introduction to Mongoose for MongoDB (2018) available from <https://www.freecodecamp.org/news/introduction-to-mongoose-for-mongodb-d2a7aa593c57/> [15 March 2021]

Mongoose v5.12.0: Schemas (n.d.) available from <https://mongoosejs.com/docs/guide.html> [16 March 2021]

Node.JS | Mongoose (n.d.) available from <https://metanit.com/web/nodejs/6.7.php> [15 March 2021]

Views · Master · Niels Müller Larsen / NativeExpressUdb (n.d.) available from <https://gitlab.com/arosano/nativeexpressudb/-/tree/master/views> [15 March 2021]

Web Programming II, Node.Js (n.d.) available from <http://dkexit.eu/webdev/site/apas02.html#assNode8M> [15 March 2021]