Christoffer Kalmar, Erik Dam, Nickolaj Sørensen
Semester 2, covid-projekt. Webudvikling 2021

Javascript & Sikkerhed - Covid-projekt
2. Semester

Projekt a2 - Corona

Indledning

I vores gruppe som består af Christoffer, Nickolaj og Erik skulle vi finde ud af hvordan vi ville skabe et To Do system der kunne hente og uploade data igennem MongoDB. Vi ville også undersøge hvordan vi kunne skabe en funktion der ville virke online sådanne at man fx. ved knap-tryk kunne vise eller hente vores collection data som JSON og/eller XML format. Yderligere så var det vigtigt for os at alle brugerrettigheder fungerer som ønsket.

1 - Lave en to-do liste som man kan gå ind og edit, archiver eller delete.
2 - Lave et admin system som kunne overse, alle brugernes to-do lister.

Vi vil derfor skabe en app ud fra de kompetencer vi har indenfor koden og med hjælp af den tidliger opgave vi har lavet. Vi vil derfor lægge den online via en database hvor vi kan hente informationen ned fra.

Problemformulering

Hvorvidt kan vi udnytte Mongoose schema til at oprette en live version af en “to do” liste med login mulighed, og hvordan får vi schema displayed til en rekvirent/administrator med “to do” data fra samtlige brugere? Samt givet både bruger og administrator forskellige rettigheder på sitet/appen.

Research

Når det kom til research så var vi over det hele. Vi brugte dkexit, google og youtube. primært brugte vi youtube da vi har fundet en youtube kanal som laver step by step guides, hvis nu der er noget meget bestemt vi leder efter så har hans kanal været til meget hjælp udover det så har vi brugt en del dkexit til at kunne finde frem til en del af problemerne vi har haft. Når hverken youtube kanalen eller dkexit har det vi mangler så er det så google vi tager i brug og der har vi heldigvis været heldige med at finde en del som vi har manglet. Men størstedelen af researchen fremgår via Dkexit og Youtube.

Analyse/Konstruktion

1. Server

Vi brugte de samme samme opsætning som vores sidste projekt 2. Derfor er opsætningen meget identisk. Vi benytter os derfor af samme funktioner og moduler som vi require gennem diverse const øverst i vores server.js fil.
Bl.a hvad der her er nævneværdigt af nyt indhold er her vores admin Setup, som kaldes længere nede på server.js.

```
const {adminSetup, done} = require('./config/terminalcomands')
```

Hertil kommer vores routes som er de samme som før, dog tilføjet routes til ToDo og Admin undersiderne.
Som det næste der kommer i server.js har vi app.set efterfulgt af app.use hvor to-do og admin routers er tilføjet.

Nu kommer dog en interessant ny tilføjelse: for at kunne etablere vores første administrator bruger havde vi brug for at benytte og implementere kode fra en cmd prompt. Dette gjorde vi ved hjælp af Heroku's egen "run console". Vi fulgte en guide for at kunne tilføje nogle process.stdin.funktioner så dette var muligt. Switch funktionen tjekker så om inputtet indeholder admin setup, hvorefter den så kører admin setup funktionen som vi selv har sat op.

```
switch (text.trim()) {
  case 'quit':
    done();
  break;
  case 'admin-setup':
    adminSetup();
  break;
  default:
```

Funktionen responder så med et spørgsmål om hvilken e-mail der er gemt i databasen der skal have ændret rollen fra 'basic' til 'admin'. Indtaster man så en email på en eksisterende bruger i vores "userreqs" collection på vores database, blir denne så omdannet til user i vores 'user' collection med rollen admin.

Et levn i vores setup er dog at vi i denne omgang ikke har benyttet os af Niels' eksempel, og vedligeholdt den gamle connection til databasen, som ikke er dynamisk men derimod kører konstant.

2. Routes

1. todo.js

Her kører vi vores forskellige requires som bl.a indeholder vores ensureAuthenticated, authRole, ROLE som alle er funktioner og moduler vi henter fra vores config folder, som omtales senere i rapporten.
Vi indhenter her vores ensureAuthenticated funktion via router.get og sætter en async funktion som parameter som gør at funktionen bliver asynkron.

```
router.get('/', ensureAuthenticated, async function(req, res) {

  //query bliver user.id
  let query = {owner: req.user._id}

  //database collection todo søger vi så efter samme id.
  ToDo.find(query, function(err, obj) {

      var todoData = obj

      ToDoArchive.find(query, function(err, data) {
          res.render('to-do', {
              todoArchived: data,
              todos: todoData
          });
      });
  });
```

Vi laver en variabel ved navn query som vi gir objektet der indeholder property owner med value req.user._id som stammer fra vores bruger database og så den bestemte brugers id.

Vi tager så vores Todo collection og bruger .find metoden på vores bruger (igennem quiry) og tilføjer (err, obj) for at sætte de to objekter i parametre til hinanden i vores funktion.

Vi konstaterer så vores todoData i en variabel som vores obj parametre og vi gør det for at få svar på user.id, når vi har dette user.id kan vi gå ind og bruge samme metode på ToDoArchive med id'et for at finde todos der er blevet arkiveret og render så disse sammen på 'to-do' siden.

router.post funktionen er intrettet i 3 cases under et switch statement hvor vi får tilsendt vores informationer vi har indtastet som bliver returneret via Ajax og via switchen konstaterer funktionen så om den skal 'create', 'archive' eller 'update' og udfører derefter den kode der passer til denne case.
I tilfældet af create: Opretter den et nyt dokument i vores todo collection.
I tilfældet af archive: Her tager vi så dokumentet og laver req.body. på namekey som title, text og user._id og gemmer dette midlertidigt via en let variabel.

```
let toDoArcive = new ToDoArchive(toDoObj);
```

så udføres ToDo.findOneAndDelete og insætter bruger id for at få denne fjernet og indeni den funktion udfører vi en toDoArchive.save for at gemme dette dokument.

I tilfældet af update: Opdaterer vi den to do, med de nye data, som brugeren har indtastet og dette gøres via en .findOneAndUpdate funktion hvor toDoObj indsættes.

2. admin.js
Her starter vi samme setup med router.get og bruger vores ensureAuthenticated sammen med authRole for at konstatere om brugeren er administrator.

Her bruger vi en switch statement med to cases: 'pendingUser' og 'userRole' så hvis man trykker på knappen decline så udføres igen .findOneAndDelete på email og derved slettes brugeren. Hvis man trykker accept så finder den så brugeren i user request collection og vender tilbage med de bruger informationer og så bruges en callback funktion der returnerer og opretter en ny bruger, og sletter så fra fra userReq og gemmer i user.

Så kan vi også ændre rollen på brugeren ved hjælp af .findOneAndUpdate hvor vi så bruger samme tilgang som før, hvor vi finder brugeren på deres email v.ha. email: req.body.email og skifter rollen via input i knapperne i admin.ejs.

3. Config

auth.js
under vores ensureAuthentication funktion har vi tilføjet authRole : function(role) for at tilføje om en bruger skal have administrator rettigheder eller ej.
Først og fremmest responder vi status 401, hvis man ikke har den rigtige rolle. Dette gøres ved hjælp af (req.user.role !== role) og det er fordi at !== er en boolean der tjekker om rollen er aktiv. role kommer fra roles.js

roles.js

Her laver vi et objekt ved navn ROLE med value key pairs ADMIN: ‘admin’ og BASIC: ‘basic’ og laver en module.export på ROLE for at bruge disse på andre sider.

terminalcomands.js
Denne bruges udelukkende til at skabe den første administrator bruger på sitet. (eller flere hvis man absolut vil lave dem via cmd prompt). Udover funktionen adminSetup som her er kreeret for at streame information fra sitet gennem cmd prompt, er her intet interessant. Måden hvorpå vi sender informationer er via input: process.stdin og output: process.stdout.

4. Models

Under vores models er vores diverse database schema sat op. Forskellen er dog at vi her har brugt samme skema til til to forskellige formål: fx. vores ToDoSchema som bruges som både vores ToDo collection og vores arkiverede ToDoArchive.
Det samme har vi så gjort ved vores user.js hvor vi bruger samme UserSchema til både vores indeværende brugere samt user requests.

5. Views

Vi skal displaye vores admin funktioner brugerventligt på en side, og det gøres via admin.ejs. Her vises en sektion hvor vi gør brug af et Javapoint Expression Tag for at vise vores users ‘forEach’. disse wrappes i css for at se lidt kønnere ud.
derudover har vi to forms med input til pendingUser samt userRole som er brugt længere oppe i rapporten i vores funktioner.

Evaluering af proces

Da vi havde fået vores ny projekt beskrivelse af vide så gik vi igen igang med at skulle få lavet noget research, vi satte os derfor sammen ind i vores discord kanal som er vores primær kommunikations værktøj for at dele filer, research, links og diverse ting. Vi skulle derfor finde ud af hvordan vi ville tage fat i den her opgave og da vi i sidste projekt fandt ud af at det hele virkede meget bedre uden trello så valgte vi derfor at bruge Scrum igen, men den her gang havde vi ikke en bestemt scrum-master i stedet så valgte vi bare at stole på hinanden og alle tre kom med

feedback om hvornår vi skulle holde møder og hvad vi skulle prøve at have færdig til de forskellige møder vi holdte henover ugen.
Mandagen mødte vi op om morgenen og snakkede om hvad vi havde fundet frem til, og hvis vi havde lavet noget, Nickolaj havde allerede der fundet en løsning på noget admin-kode som ville gøre resten af projektet en del nemmere for os. De resterende i gruppen downloade så det GIT for at prøve at arbejde videre med det for at prøve at få kodet noget ekstra til det næste møde som vi holdte om aftenen. Og det møde gik egentlig også fint udover at mange havde ikke nået det vi egentlig ville have nået så vi valgte derfor bare at scrappe de møde og så mødes tirsdag eftermiddag i stedet, hvor der var lidt mere fremskridt.
Med hensyn til kodningen så fandt vi også ud af i sidste projekt at de med at skulle lave live-share var lidt af et problem især fordi vi ikke allesammen var online på samme tid, og hvis bare ham der live-sharede var væk så kunne de 2 andre ikke arbejde videre. Derfor valgte vi bare at uploade til github som resten så kunne downloade. Det gjorde at vi ikke havde brug for at sidde sammen 24/7 for at kunne kode og gjorde sådan vi kunne sætte os hver for sig med musik.
All in all så gik det meget stille og roligt, der var ikke alt for meget stress over dette projekt selv om det alligevel var 2 projekter i træk. Vi var heldige med at vi kunne genbruge en stor del af koden fra det forrige projekt, og så egentlig bare bygge videre på det med de nye funktioner vi skulle tilføje via det her projekt.
Alle i gruppen holdte tiderne som vi havde aftalt til møder så det hele virkede som det skulle.

Afgrænsninger

Yderligere skal det være muligt for en rekvirent med adgangstoken at hidkalde samtlige data på databasen fra flere brugere i både json og XML format.

Ovenstående opgave nåede vi ikke. Der er brugt megen tid fra alle medlemmers side af for at give et forsøg, og intet er opnået. Vi har bl.a prøvet at bruge en simpel variabel og udføre en JSON.parse(ToDo) hvor ToDo er vores collection. Vi har dog konstateret at man skal forstå de individuelle collections som indeholdende

individuelle dokumenter, og det så er disse der skal hidkaldes.
Hvordan vi præcis skal gøre dette kunne vi ikke finde nogen løsning på. Vi prøvede os lidt hen ad vejen med at bruge det direkte bruger id der er synligt i collection i databasen, men dette virkede ikke. Dernæst prøvede vi at benytte os af req.user._id som heller ikke virkede. Selvom vi i det mindste havde håbet på den så ville være mulig i det mindste at hente fra en enkelt bruger.

Vi fandt dog en guide på MongoDB til at hente JSON samt CVR filer fra sin online MongoDB database.
https://docs.mongodb.com/database-tools/installation/installation-windows/ for at installere diverse tools til MongoDB. Ved hjælp af denne kunne man udføre en konkret mongoexport command på sin computer som var følgende:

mongoexport --uri mongodb+srv://ed:lollollol@cluster0.tiehk.mongodb.net/myFirstDatabase --collection todos --type json --out C:\Users\erikh\Desktop/todoliste

Commanden fungerer således at man ved hjælp af --URI kan opsætte en connection string URI format hvor man vælger mongodb+server hvor man bruger sit bruger id til mongoDB Atlas Cloud samt login forinden at clusteret man ønsker vises skal vælges. Efter --collection tilføjer man så den ønskede collection man vil have formateret, dernæst hvilken --type format man ønsker og afslutter med --out (indtast filplacering) for at hente filen ned. så på denne måde har vi faktisk "løst" opgaven i at hente JSON som så kan omdannes til XML, men denne funktion kunne vi ikke finde ud af at bygge ind i sitet så det kunne bruges så derfor arbejdede vi ikke videre i denne funktion. Da den i realiteten ikke er funktionsdygtig, for hvis nogen skulle bruge disse informationer så skulle vi gøre dette manuelt hver gang.

Konklusion

Vi har gennem arbejde med schema bemærket at mange elementer går igen, sådanne funktioner som req.body, .find(One/And/Delete etc.) er altså metoder er noget vi har brugt lidt på kryds og tværs af hinanden for at skabe vores ønskede output, hvilket vi må erkende har virket ret godt i virkeligheden.
Vores login side blev opdateret med en simpel "ROLE" indhold, som virkede bedre

og med meget mindre kode end forventet. Siderne er dermed lukket så kun Admin kan komme ind på Admin, og users kun på To Do undersiden.

Det var dog meget beklageligt vi ikke kunne finde en bedre løsning på at hente dataene ned fra collection formateret sådanne at vi kunne bruge dem live på sitet, da det var hovedmålet. Men så må vi håbe Niels lærer fra sig.

Bilag

Billag er inkluderet i selve rapporten og resterende billag er koden selv som findes på https://github.com/Mrown/todo

Referencer

https://www.youtube.com/watch?v=jI4K7L-LI58&ab_channel=WebDevSimplified
http://dkexit.eu/webdev/site/ch22s03.html
http://dkexit.eu/webdev/site/index.html
https://www.youtube.com/watch?t=245&v=-JcgwLIh0Z4&feature=youtu.be
https://github.com/Mrown/projekt2
https://docs.mongodb.com/database-tools/installation/installation-windows/