



26-05-2021

Yadda

Af Kenneth, Christian og Morten

Indhold

Indledning.....	2
Problemformulering	3
Metodeovervejelser	5
Research.....	5
Analyse	7
Konstruktion	10
Evaluering af proces og produkt.....	15
Konklusion.....	16
Referencer & Bilag	17

Indledning

I denne rapport vil vi komme nærmere ind på processen i det eksamensprojekt vi er blevet stillet i forbindelse med undervisningen. I projektet har vi fået til opgave at lave en webapplikation, der skal fungere som et socialt medie med muligheden for at kommunikere med hinanden vha. korte beskeder. På vores sociale medie skal det være muligt for en person at oprette en bruger, hvorefter man vha. en angiven email vil modtage en bekræftelsesmail til at godkende oprettelsen og give mulighed for at logge ind. Som altid når man opretter en bruger, vil vi selvfølgelig arbejde med hashede kodeord i vores database, og samtidig skal en bruger oprette sig med sit rigtige navn samt et brugernavn.

Når man er logget ind på det sociale medie, skal man som bruger have mulighed for at kunne "følge" andre brugere for at få fremhævet deres beskeder på sin tidslinje, og samtidig skal man kunne se hvilke brugere der vælger at følge en selv. Hvis man ikke følger nogle brugere, vil man få vist beskeder fra alle brugere sorteret efter nyeste øverst. Ved hver besked på sin tidslinje vil der også være tilknyttet en knap, så man kan besvare den specifikke besked. Sortering af beskeder på sin tidslinje skal også kunne ske vha. tags, der enten er prædefinerede eller som brugerne har oprettet. Også her skal visningen være sorteret i nyeste først.

Udover funktionalitet skal løsningen også gøres brugervenlig, så det er let for en bruger at finde rundt på siden og gøre brug af de funktionaliteter der er.

Formålet med projektet går ud på at vi som studerende kan anvende de færdigheder vi har tilegnet os i fagene, på en praktisk måde, i en problemstilling som ikke er kendt af os i forvejen.

I projektet vil vi primært gøre brug af Node.js og Express sammen med MongoDB, men også JavaScript og andre teknologier vi har lært eller gjort brug af i forbindelse med undervisningen. Vi vil også gøre brug af teori fra vores undervisning, og fra relevante bøger eller forums på nettet.

Vi vil i løbet af rapporten komme ind på hvilke metodeovervejelser vi har gjort os, på baggrund af den opgave vi har fået stillet. Derudover vil vi forklare hvordan vi har

researchet for at få løst opgaven, og i vores tilfælde vil denne del mest gå med at forklare det vi har lært i undervisningen, da det er udgangspunktet for den stillede opgave, og vores research har dermed baseret sig på emner relevant for dette.

I analysefasen vil vi forsøge at besvare de spørgsmål vi har stillet os selv i vores problemformulering, for at analysere på hvordan vi har prøvet at løse dem.

I konstruktionsfasen vil vi gå mere i dybden med opbygningen af nogle af de forskellige elementer som udgør webapplikationen, og vores primære fokus vil her være på Express og MongoDB, som er de læringselementer opgaven fokuserer på.

Til sidst i rapporten vil vi evaluere hele processen, og komme ind på, hvilke dele af opgaven der har været mest udfordrende, samt hvilke dele var lettere at håndtere. Derudover vil vi komme ind på om der er noget vi ville have gjort anderledes, eller prioriteret på en anden måde hvis vi skulle gøre det igen, for til sidst at afslutte med en konklusion.

Problemformulering

Vi vil i denne opgave udvikle en web-applikation, der fungerer som socialt medie. Applikationen skal have et registrering/login-system, som er det første brugeren bliver mødt af. Når man vil registrere sig som bruger, vil vi gøre brug af en valideringsmail som skal godkende den pågældende bruger og give adgang til at kunne logge ind. Hver bruger skal have deres egen personlige profil, hvor de skal kunne se hvem de følger og hvem der følger dem, samt mulighed for at importere et profilbillede.

Applikationens forside skal fungere som en tidslinje, hvor brugere kan se posts lavet af folk som de følger, samt et inputfelt, hvor de kan udforme deres egne posts. Tidslinjen ligger lodret i midten af skærmen og til siderne vil der være plads til annoncer. Brugeren skal også have mulighed for at lave en "tråd", altså tilføje beskeder under hvert post, som er kædet sammen. Derved kan brugere interagere sammen. Brugerne skal derfor kunne klikke på et post og komme ind på en

underside for dette, hvor svarene i tråden kan ses og skrives. De skal tilmed have et søgefelt, hvor de kan filtrere posts med tags, der svarer til inputtet i søgefeltet, og som en del af brugeroplevelsen have mulighed for at ændre applikationens farver via en knap.

Hvordan sender vi en validerings mail til en nyoprettet bruger?

Hvordan gør vi at brugeren kun kan se "yaddas" fra dem de følger?

Hvordan viser vi brugeren hvem der følger ham/hende, og hvem brugeren følger?

Hvordan sikrer vi at brugeroplevelsen forbliver god på tværs af hele applikationen?

Hvilken hashing-algoritme er optimal til hashing af brugernes passwords?

Metodeovervejelser

Inden projektstart og i løbet af vores projekt, har vi gjort os flere metodeovervejelser. Bl.a. som en del af opgavebeskrivelsen, skulle vi undersøge en anden krypteringsmetode til brug ved vores kodeord. Derefter skulle vi tage stilling til om vi ville bruge Bcrypt løsningen(Kelektiv 2021), som vi har brugt i tidligere projekter eller om vi ville bruge den anden løsning vi sammenlignede med. I tilfældet her, valgte vi at sammenligne med en løsning som hedder SimpleCrypt(ShowClix 2013). Som i de tidligere projekter har vi valgt at fortsætte med at bruge Bcrypt, da det er en løsning vi har erfaring med og på den måde kan vi afsætte mere tid til at lave den reelle løsning til projektet. Forskellen på de to vil vi komme nærmere ind på i research afsnittet.

Da vi i projektet også er blevet stillet udfordringen, at vi skal bruge email validering når en bruger opretter sig selv, har vi også her skulle finde på en måde at gøre dette på. I vores tilfælde har vi forsøgt os med et Node modul der hedder Nodemailer(Andris Reinman 2010), som gør det muligt at sende en email vha. en funktion i sin kode til en anden mail.

Desuden har vi valgt at bruge Pug som template engine, da det også er noget vi har erfaring med fra tidligere projekter og har arbejdet med i undervisningen.

Research

I vores researchfase har vi primært gjort brug af vores tidligere projekter, for at se om der var funktionaliteter vi ville have brug for igen, og dermed kunne modificere dem til at passe til vores projekt. Da vi i undervisningen har fået at vide at vi ikke skal gentage os selv, eller genopfinde den dybe tallerken, har vi ikke set nogen grund til fx at skulle bygge en login funktionalitet fra bunden igen. Så de elementer vi har kunne gøre brug af igen, har vi taget med over i det nye projekt og justeret dem så de passer til det nye projekt. Samtidig har vi fastholdt samme mappestruktur som tidligere, for at gøre fremtidig fejlsøgning mere overskueligt.

Som nævnt i metodeovervejelser har vi også lavet research omkring email validering og krypteringsmetoder, for at finde ud af hvordan vi lettest ville kunne gøre brug af disse funktionaliteter.

Til kryptering af brugernes password bruger vi en hashingalgoritme. I dette projekt bruger vi bcrypt, ligesom vi har gjort til alle andre projekter. Bcrypt er et modul, der bruges som hashingfunktion til netop passwords. Bcrypt installerede vi i projektmappen via npm install, og vi har required det via en const i vores dokument ved navn handlerAccounts.js med følgende kodelykke:

```
3  const bcrypt = require("bcryptjs");
```

Vi bruger den indbyggede require-funktion til at inkludere bcrypt-modulet. Hashing af brugernes password foregår via scriptet i vores handlerAccounts-fil, som er forbundet med skemaet i accounts-filen. Vi rammer skemaets password via følgende kodelykke:

```
88  let hash = await bcrypt.hash(password, 10);
89  let newUser = new Account({
90    username: username,
91    firstname: firstname,
92    lastname: lastname,
93    email: email,
94    password: hash,
95    confirmationCode: token, //Email verification
96  });
```

I vores database, Accounts, hvor brugere opbevares, kommer brugeres passwords så til at have følgende syntaks:

```
"password" : "$2a$10$1Hyvu8JwtZxkJN1OHxRPn.StNjd1ackUXNm/8LwFhG9Wdx6ftCIoC",
```

Grunden til at det er implementeret er, at brugernes passwords nu ikke er opbevaret i den plain tekst, som de selv har angivet, da de registrerede sig på vores platform. Det gør naturligvis hacking betydeligt sværere. Strings som den ovenstående består af 60 karakterer, som kan opdeles i fire dele. "\$2a\$" er en hashingalgoritme-identifier, hvis formål blot er, at angive at vi bruger bcrypt. "10" er en cost-factor, som vi har angivet vores hash-variabel på linje 88 i handlerAccounts.js. Den tredje del; "1Hyvu8JwtZxkJN1OHxRPn." er en 22-karakter lang salt, der er indbygget i bcrypt. Salting er en funktionalitet, der tilføjer et yderligere lag sikkerhed til passwords

udover selve hashingen. Det er en række tilfældige karakterer, som tilføjes til brugerens password før den gemmes i databasen. Den fjerde og sidste del af det ovenstående password; "StNjd1ackUXNm/8LwFhG9Wdx6ftCloC" er den hashede værdi af brugerens password. Den saltede og hashede værdi står altså sammenlagt. Bcrypt's hashede værdi er 31 karakterer lang og er en 192-bit længde, mens dets salting er på 128-bits.

Et alternativ til bcrypt, som også kan bruges i expressprojekter er simplecrypt. Det kan ligeledes installeres med npm install og requires på samme måde som ovenstående. Simplecrypt har dog dets forskelle i forhold til bcrypt, da simplecrypt bruges til kryptering i stedet for hashing. Det kan altså ændre passwords til en lang, uigenkendelig string ligesom bcrypt, men da den krypterer passwords i stedet for at hashe, så vil der være en nøgle, der kan dekryptere brugernes passwords igen. Det har to simple, indbyggede funktioner til kryptering. En til kryptering af plain tekst til ciphertekst, og en til dekryptering af ciphertekst til plain tekst. Deres navne er henholdsvis "encrypt" og "decrypt". Simplecrypt's kryptering anvender som udgangspunkt AES 192, der er samme længde som bcrypt's hashede værdi, så på det punkt er de meget lige. Simplecrypt har tilmed en funktion, .salt, som kan tilføje salts til krypteringen. Den tilføjer som udgangspunkt en tilfældig salt, men det kan skræddersyes til at passe ens præcise behov, hvilket er en fordel. Vi fortsætter dog med bcrypt, da vi har bedst kendskab til denne, samt fordi dokumentationen til bcrypt er væsentligt mere dybdegående, hvilket er meget behjælpeligt, når man skal lave denne type projekter.

Analyse

I vores problemformulering har vi spurgt om hvordan vi sikrer os at en bruger kun kan se de beskeder som tilhører de personer brugeren følger, hvordan vi viser brugeren hvem der følger ham/hende og hvem brugeren selv følger. Derudover har vi spurgt, hvordan vi sikrer os at brugeroplevelsen forbliver god på tværs af hele applikationen.

For at brugeren kun kan se beskeder fra de personer de følger, har vi gjort brug af PUG's each funktion 2 gange. Én gang for hver yadda og igen én gang for hver follower. Brugeren vil derfor kun se beskeder fra dem han følger da vi gør brug af en if funktion i vores PUG fil, som sammenligner hver yadda og followers "username" og udskriver kun dem der er ens.

```
each yadda in yaddas
  each follower in followers
    if follower.following === yadda.username
```

Inde på profilsiden har vi lavet en form for menu, hvor brugere vil kunne se hvem de følger, samt hvilke andre brugere, der følger dem selv. Selve menuen er udformet i profiles.pug-filen, på nedenstående billede.

```
30   div(id="tabs-bar")
31     button(id="followersBtn" class="tablink tabButton-active") Followers
32     button(id="followingBtn" class="tablink") Following
33   div(id="follow")
34     div(id="followers" class="tab-content")
35
36     each follower in following
37       div(class="tab-inner")
38         div(class="followUser")
39           a(href="/profiles/" + follower.following)
40           img(src="/getImage/" + follower.following alt="missing image of " + user.username)
41           a(href="/profiles/" + follower.following)
42           h3 #{follower.following}
43   div(id="following" class="tab-content")
44
45   each follower in followers
46     div(class="tab-inner")
47       div(class="followUser")
48         a(href="/profiles/" + follower.following)
49         img(src="/getImage/" + follower.following alt="missing image of " + user.username)
50         a(href="/profiles/" + follower.following)
51         h3 #{follower.following}
52         if user.username === currentUser.username
53           a(href="/unfollow/" + follower._id) Unfollow
```

Vi har delt menuen op i to dele; Following og Followers. I hver halvdel bruger vi vores followers-kollektion til at finde ud af, hvem den enkelte bruger følger og hvem der følger brugeren. Vi har dertil nogle get-requests som efterspørger den data vi mangler, for at kunne printe brugernes data ud i menuen. Gennem vores kode har vi derfor gjort sådan, at enhver bruger kan se hvem præcist de følger og hvem der følger dem.

Brugeroplevelsen har hele tiden været i centrum af projektet. Før vi begav os ud i kode, startede vi med at udarbejde en low-fidelity prototype i Adobe XD. Herigennem sikrede vi, at vores designvalg var gennemtænkte med henblik på brugeren. Vi lavede tilmed også farvetemaerne som noget af det første. Igennem vores

brugeroplevelsesdesign fulgte vi C.R.A.P.-reglerne, hvorved målet for os er at gøre indholdet overskueligt og selvsigende, mht. hvordan brugeren skal forstå og interagere med det.

I vores farvetemaer har vi anvendt to primære farver i hvert tema; hvid og lilla, og mørk grå og turkis. En del af grunden til at vi valgte netop disse er den kontrast der er mellem både temaerne og de to primære farver i hver. De gør det let for os at adskille indhold på hjemmesiden på en logisk måde for brugeren at forstå. Et eksempel er den nedenstående information, man kan finde inde på hver brugers profilside. Den lilla farve gør det nemt for øjet at skelne mellem overskrifterne og det data der står derunder. Der er mange lignende eksempler på tværs af hele siden, hvor dette er gennemgående.

Username:
Christian Ehlers


First Name:
Christian

Last Name:
Ehlers

Email:
christiankrushave@gmail.com


Enable dark mode:
☐

Repetitioner er et andet centralt element i vores opstilling af indhold. Vores tidslinje er et af de steder, hvor vi har gjort mest brug af det, da vores opslag naturligvis er identiske mht. opstilling, men bare har noget forskelligt indhold i form af username, tidspunkt, selve opslagets tekst med mere. Det ses på billedet herunder. Det er grundet repetitionen, at brugeren vil kunne se et mønster i indholdet og derved forstå hvilke dele af indholdet, der er kædet sammen, så de kan kende forskel på de individuelle opslag.



kennethm
test185
Reply

10:11 | 25 May 2021

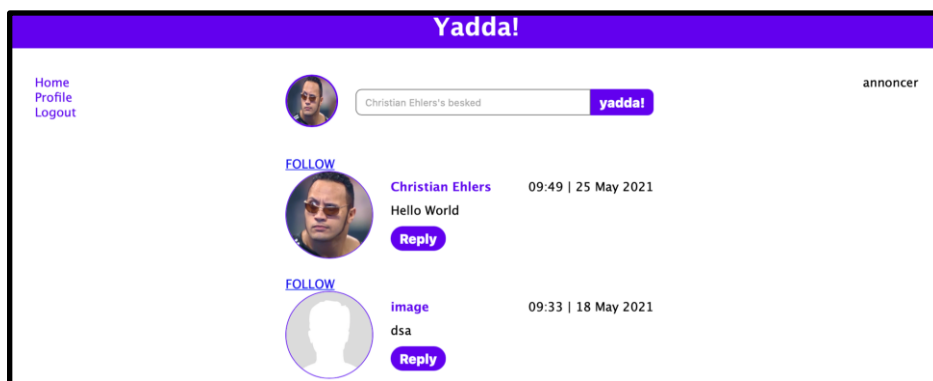


ADMIN
hejh
Reply

10:11 | 25 May 2021

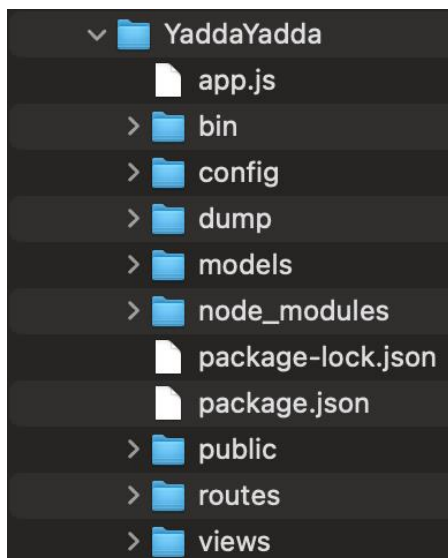
Som det kan ses på de to ovenstående billeder, så er alt vores indhold alignet med hinanden. Det hjælper med at mindske forvirring og forbedre overskueligheden i applikationen. Ligesom med repetitioner skaber alignment en visuel forbindelse mellem de forskellige dele af indholdet.

Proximity har været vigtigt i opbygningen af designet. Vi har blandt andet brugt det til at adskille menuen til venstre på siden, tidslinjen i center samt annoncepladsen til højre. De tre dele bør ses som tre separate grupper af indhold, hvorfor der er en stor mængde whitespace mellem dem, specielt hvis man sammenligner med proximity mellem hvert opslag i tidslinjen, som naturligvis hører sammen. Det er et godt værktøj til organisering af indhold, som vi har brugt netop hertil



Konstruktion

Vores projektmappe, YaddaYadda's konstruktion ses på det nedenstående billede:



På root-niveau i YaddaYadda-mappen ses tre filer; package.json, package-lock.json og app.js. Package.json er en obligatorisk fil, som bruges til at opbevare al metadata samt dependencies, som bruges i vores projekt. Det er eksempelvis bcrypt, pug, mongodb mm. Package-lock.json holder blandt andet styr på de præcise versioner af de installerede packages, der bruges i projektet. App.js indeholder en række requirements til packages samt errorhandlers til applikationen.

Bin-mappen indeholder kun én fil; www. Den bruges til at lave selve serveren for vores applikation, samt finde en port og lave errorhandlers i forbindelse med listening-errors med serveren.

Config-mappen indeholder to filer; config.js og nodemailer.js. Config.js indeholder blot et enkelt exporteret js-objekt, som indeholder oplysninger omkring vores “dummy” email-adresse. Vores dummy email bruges til automatisk at sende en mail som verificering til de brugere som opretter sig i vores system. Dvs. at hvis man som bruger forsøger at oprette sig med en ugyldig email, får man aldrig adgang til at kunne logge ind på vores sociale medie, da man ikke får mulighed for at aktivere sin bruger med den adgangstoken som mailen indeholder.

Filen er selvfølgelig indsat i vores git.ignore-fil, da objektet indeholder sensitiv information såsom fx kodeordet til vores dummy mail.

Nodemailer.js filen indeholder vores funktionalitet til at sende vores verificeringsmail, samtidig med at den fortæller hvilken mailtjeneste vi gør brug af, trækker vores

kodeord og mail ind fra config filen og opsætter hvordan mailen skal opstilles og hvilken tekst og indhold der skal være i den når en bruger modtager den.

```
1  const nodemailer = require('nodemailer');
2  const config = require('../config/config');
3  const user = config.user;
4  const pass = config.pass;
```

I linje 1-4 inkluderer vi nodemailer modulet, som vi har installeret gennem npm install, så vi får mulighed for at gøre brug af de indbyggede funktioner som nodemailer giver adgang til. Derefter inkluderer vi vores config fil, så vi får mulighed for at få adgang til mailen og kodeordet til vores dummy email, da vi skal bruge disse informationer for at sende en mail afsted til brugere.

```
6  let transporter = nodemailer.createTransport({
7    service: 'gmail',
8    auth: {
9      user: user,
10     pass: pass,
11   }
12 });
```

I linje 6-12 laver vi så det der i nodemailer hedder et transporter objekt, som gør det muligt at sende en mail. Her inkluderer vi som beskrevet tidligere hvilken mailtjeneste vi gør brug af, samt mailadressen og koden inkluderet fra config filen.

```
14 module.exports.sendConfirmationEmail = (name, email, confirmationCode) => {
15   console.log("Check");
16   transporter.sendMail({
17     from: user,
18     to: email,
19     subject: "Please confirm your Yadda account",
20     html: `<h4>Hello ${name}</h4>
21     <p>Thank you for creating a user on our Yadda social media.</p>
22     <p>Please confirm your email by clicking on the following link to be able to login to your account.</p>
23     <a href=http://localhost:3000/confirm/${confirmationCode}> Confirm your email adress</a>
24     <br>
25     <p>Best regards</p>
26     <p>Yadda M, C, K</p>
27     </div>`,
28   }).catch(err => console.log(err));
29 }
```

I linje 14-29 har vi så gjort brug af vores transporter objekt, hvor vi har inkluderet det i en funktion som gør brug af det navn som brugeren skriver i oprettelsen, den email som brugeren skriver ind og derefter den confirmationCode som vi har generet i vores handlerAccounts fil. Denne token bliver oprettet vha. et modul der hedder JsonWebToken(Auth0 2019), som generer en random string der bliver sendt med i den verificeringsmail som brugeren modtager.

```
45 const token = jwt.sign({ email: req.body.email }, config.secret); // Token for email verification
```

Som det kan ses af linje 19-27 på billedet ovenover, opsætter vi vores email vha. HTML, for at designe hvordan den skal se ud når en bruger modtager den.

Hello Kenn.

Thank you for creating a user on our Yadda social media.

Please confirm your email by clicking on the following link to be able to login to your account.

[Confirm your email adress](#)

Best regards

Yadda M, C, K

Trykker man så på linket “Confirm your email address”, bliver man sendt ind på vores side, hvorefter man så kan logge ind.

Dump-mappen indeholder .bson- og metadata.json-filer til vores accounts, yaddas og followers. Filernes primære formål er at opfyldes og opbevare de datasæt, som udfyldes af vores brugere i diverse inputfelter og knapper, når der oprettes nye brugere, laves nye yaddas og følges brugere.

Models-mappen indeholder ligeledes filer til vores accounts, yaddas og followers, bare i form af skemaer og handlers hertil. Det er eksempelvis accounts.js og handlerAccounts.js. Filerne uden “handler” i navnet definerer typer og krav til vores mongoose-schema for indholdet af brugere, mens vores handler-filer derefter behandler informationen givet, når en bruger eksempelvis oprettes. I handlerAccounts.js bliver brugernes passwords eksempelvis hashet med bcrypt, brugernes valgte farvetema defineres med mere. Et eksempel herunder er vores default billede, som brugere får når de oprettes og ikke vedhæfter et profilbillede. Heri funktionen dummyImage indsættes billedet defaultUser.jpg, som findes i publicmappen, der er beskrevet senere.

```

194 dummyImage: async function (req, res) {
195   let path = "public/images/defaultUser.jpg";
196   let content = "image/jpeg";
197   await fs.readFile(path, function (err, data) {
198     // awaits async read
199     if (err) {
200       console.log(`Not found file: ${path}`);
201     } else {
202       res.writeHead(httpStatus.OK, {
203         // yes, write header
204         "Content-Type": content,
205       });
206       console.log(`served routed file: ${path}.`);
207       res.write(data);
208       res.end();
209     }
210   });
211 },

```



image

09:33 | 18 May 2021

dsa

Reply

I node_modules-mappen findes alle de packages og moduler, der er eller kan være nødvendige i projektet. Der findes eksempelvis en undermappe til bcrypt. Mappen kom med npm install, da projektet først blev oprettet. Når vi i løbet af projektet har installeret diverse packages, så er de automatisk blevet kopieret ind i node_modules.

Public-mappen indeholder tre undermapper; images, javascripts og stylesheets. Images bliver brugt til at opbevare brugernes default profilbillede, som indsættes i tilfælde af, at de ikke selv vælger en billedfil. Javascripts indeholder én js-fil; date.js. Date.js bruger vi til at formatere/style datomærkningen ud for hvert opslag, brugerne opretter. Filen indeholder et array med måneder og tre funktioner. Den ene funktion, formattedTime, til at formatere datoen, en anden, fullTime, til at formatere tidspunktet, og en tredje, fullDate, til at sammensætte det hele, så vi får vores ønskede opstilling, der ses herunder:

09:33 | 18 May 2021

Stylesheets-mappen indeholder vores css- og less-filer, der som mappen beskriver, bruges til at kontrollere udseendet mm. på tværs af hele applikationen. Det er blandt andet her vi har angivet udseendet for vores dark theme, som kan aktiveres på profilsiden.

Anden nederste mappe er routesmappen. Heri har vi to filer; index.js og users.js. users.js er en fil, der automatisk blev oprettet da vi lavede projektet via npm. Filen har ikke et pt. et formål. I index.js opbevarer vi alle vores get- og post-requests, som bruges på tværs af siden. Det er eksempelvis den nedenstående get-request:

```

74 router.get("/logout", function (req, res) {
75   user = false;
76   admin = false;
77   // console.log(admin);
78   req.session.destroy();
79   res.redirect("/login");
80 });

```


Den bruges til vores logout-knap og sørger for, at brugerens session destrueres, når knappen trykkes på. Den fører herefter brugeren ud på login-siden igen, via en `res.redirect`, fordi der ikke skal gives adgang til andre sider, når brugeren ikke er logget ind.

Sidst har vi views-mappen, som indeholder alle vores pug-filer. Filerne heri er naturligvis forbundet med vores get- og post-requests, som set ovenfor. Vi valgte igen at bruge pug som vores template engine, fordi det er den vi har bedst kendskab til, så den virkede som det åbenlyse valg. Pug-filernes indhold er en vigtig del af den overordnede funktionalitet af vores applikation, da det er data fra disse filers inputfelter, der bliver brugt i mange af de get- og post-requests, vi har. Et eksempel er nedenstående form, der indeholder det inputfelt og den knap, som er der hvor brugerne udformer og sender deres nye opslag, hvorefter de printes på siden og gemmes i yadda-kollektionen i databasen. Billederne kommer fra `index.pug`.

```

7   form(action="/" method="post")
8     input(type="text" name="yadda" placeholder=`${currentUser.username}'s besked`)
9     button yadda!

```



Evaluering af proces og produkt

Projektet har overordnet været succesfuldt, og meget lærerigt. Vi havde mulighed for at komme hurtigt i gang, da meget af basen for projektet kunne replikeres fra tidligere projekter med små ændringer ift. funktions- og kollektionsnavne mm. I takt med, at vi kom længere hen i projektet stødte vi dog på nogle koderelaterede vanskeligheder, såsom emailvalidering af brugerne og hvordan man følger andre

brugere og printer det ud for den individuelle bruger. Herudover kunne vi godt have lavet flere filer i vores routes-mappe, da vi har indsat alle get- og post-requests i én fil, index.js, hvilket kan være en smule uoverskueligt. Vi kunne have delt filen op i 2-3 filer, der hver har forskellige navne afhængigt af, hvad deres indhold passer til. Det ville spare os tid fra at lede efter det specifikke indhold i index.js, som vi leder efter. Til tidligere expressprojekter har vi kun brugt en enkelt fil, men de projekter har også været betydeligt mindre end dette, hvilket sandsynligvis er hvorfor vi ikke tænkte på opdelingen før filen blev en anelse uoverskuelig. I vores slutprodukt har vi ikke nået at inkludere muligheden for at søge efter opslag, der indeholder et specifikt hashtag. Det var grundet tidsmangel til sidst at det blev nedprioriteret, så på det punkt kunne vores projektforsløb have gået en smule bedre.

Konklusion

Hvis vi kigger på de spørgsmål vi har stillet os selv i vores problemformulering, kan vi ved projektets afslutning konkludere at vi har opnået at svare på, og løse de problemer vi har stillet os selv.

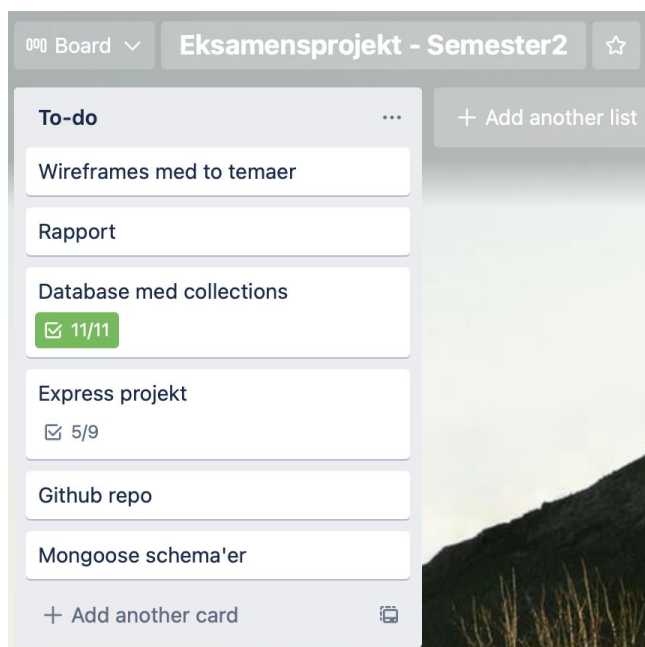
Vi har sikret os at hver person der opretter en bruger, også får en validerings mail, som giver adgang til kontoen. Og hvis man har forsøgt at oprette sig med en email man ikke har adgang til, eller man ikke trykker på aktiveringslinket, får man ikke adgang til kontoen. På den måde har vi forsøgt at forebygge oprettelse af falske profiler, fra fx. bots eller folk der bare hurtigt vil have adgang, til en hvis grad.

Derudover har vi gjort så en bruger har mulighed for kun at se de beskeder fra personer han følger, men samtidig har vi holdt fast i at brugeren også skal have adgang til at kunne se beskeder fra alle brugere.


Referencer & Bilag


1. Andris Reinman (2010) *Nodemailer* :: *Nodemailer* [online] available from <https://nodemailer.com/about/> [25 May 2021]
2. Auth0 (2019) *Jsonwebtoken* [online] available from <https://www.npmjs.com/package/jsonwebtoken> [26 May 2021]
3. Kelektiv (2021) *Bcrypt* [online] available from <https://www.npmjs.com/package/bcrypt> [25 May 2021]
4. ShowClix (2013) *Simplecrypt* [online] available from <https://www.npmjs.com/package/simplecrypt> [25 May 2021]

1. Tidsplan - overordnet



2. Tidsplan - funktionaliteter

 **Express projekt**
in list [To-do](#)



Description

Add a more detailed description...

☒ **Funktionaliteter**

Hide checked items

Delete

56%

☒ Oprette-bruger

☒ Login

☒ Tidslinje

☒ Se-egen-profil

☒ Se-andres-profiler

☐ Sortering efter tags

☐ Sortering efter nyeste Yadda's først

☐ Mulighed for follow/unfollow

☐ Mulighed for at skifte mellem to temaer

3. Tidsplan - kollektioner

Database med collections

in list [To-do](#)

Description

Add a more detailed description...

Brugere

100%

- ☒ Name
- ☒ Email
- ☒ Username
- ☒ Password
- ☒ Theme
- ☒ User-or-admin
- ☒ Profile-picture

Add an item

Yaddas

100%

- ☒ Username
- ☒ Date/time
- ☒ eventuelt-is-profilbillede
- ☒ Besked