

Indholdsfortegnelse

Indholdsfortegnelse	1
Forord(Mads)	2
Indledning(Sune)	5
Problemformulering	5
Metodeovervejelser(Hanna)	5
Hashing(Mads)	6
Validering(Mads)	7
Email confirmation(Mads)	8
Research	10
Sikkerhed(Mads)	10
Nodemailer (Mads)	11
Multer (Mads)	12
Moment.js (AK)	12
#tags (AK)	13
Tema (AK)	14
Analyse	15
Specifikke krav (Hanna)	15
Wireframes (Hanna)	16
Konstruktion	17
Node.js og Express (AK)	17
 2. SEMESTER IBA WEB DEVELOPER	 1

Pug (AK)	17
Templates (AK)	19
Schemas/models(Hanna)	20
Controllers(Sune/Mads/AK)	24
userController(Mads)	24
authController(Mads)	24
indexController(Sune)	24
yaddaController(Sune)	25
tagController(AK)	27
errorController(Mads)	28
adminController(Mads)	28
Routes(Sune/Mads)	29
indexRoutes (Sune)	29
usersRoutes (Mads)	29
yaddaRoutes (Sune)	29
User Management og Users API: (Mads)	30
Evaluering af proces og produkt	32
Proces (AK)	32
Evaluering af produkt(AK)	32
Udfordringer(Hanna)	34
Konklusion	36
Referencer	37
Bilag 1: package.json (Mads)	38
Bilag 2: Wireframes	44
 2. SEMESTER IBA WEB DEVELOPER	 2

Forord(Mads)

I forbindelse med vores produkt har vi lavet en sikkerheds fil der indeholder vigtig data som Username og Passwords til vores mongoDB Cloud database, Token Secrets og deres udløbsdato, Authentication data til mail servers (Mailtrap.io¹ og Sendgrid²). Denne fil er er vedhæftet i den mail vi har afleveret projektet i. Filen skal sættes ind i vores projekts repository mappe (..\yaddayaddayadda\repository\)

Det er nødvendigt at ændre i denne fil hvis man vil kunne modtage confirmation emails i vores servers development mode (npm run start).

Fra linje 15 til 18 er det muligt at indsætte username, password, host og port. Derved kan man forbinde til sin egen mailtrap.io server for opfange alle mails der bliver sendt ud.

```
14  # SMTP Settings
15  EMAIL_USERNAME=
16  EMAIL_PASSWORD=
17  EMAIL_HOST=smtp.mailtrap.io
18  EMAIL_PORT=2525
```

Såfremt der ønskes adgang til vores mongoDB Cloud database har vi oprettet en bruger på forhånd:

Username: ibakolding

Password: 09R8Exz3X2TYZusF

Commandline

```
mongo "mongodb+srv://cluster0-kxy3b.mongodb.net/test" --username ibakolding --password 09R8Exz3X2TYZusF
```

MongoDB Compass

```
mongodb+srv://ibakolding:09R8Exz3X2TYZusF@cluster0-kxy3b.mongodb.net/test
```

¹ Mailtrap.io. "Fake smtp testing server. Dummy smtp email testing". Set 24. maj 2020. Mailtrap.io

² <https://sendgrid.com/>

Forord Fortsat (Mads)

Development og Production mode

Vores server kan startes i 2 modes. Development mode (**npm run start**) og Production mode (**npm run start:prod**) Forskellen på development og production mode er at i production mode vil der ikke blive vist detaljer omkring fejlbeskeder i terminalen eller på vores servers error side.

Som backup hvis der opleves problemer med oprettelse af en bruger på vores hjemmeside har vi lavet en test bruger som man kan benytte sig af.

Email: iba@test.com

Password: 1234

Det er også en mulighed at oprette en bruger som normalt og derefter omgå email authentication ved at ændre værdien for "isConfirmed" til "true", ved den bruger man har oprettet i vores MongoDB user collection.

Indledning(Sune)

I denne rapport vil vi redegøre for vores projektforløb med YaddaYaddaYadda. Vi har i dette projekt gjort brug af en masse nye ting såsom Middleware & JWT Tokens. Vi har også som i andre projekter gjort stor brug af Mongoose & MongoDB. Ved hjælp af blandt andet disse ting har vi udformet en hjemmeside, som vi igennem vores rapport har forsøgt at forklare omkring på et skriftligt niveau.

Problemformulering

Hvordan kan vi udvikle et SoMe website i Node.js med Express, MongoDB og Mongoose, hvor brugeren har de muligheder og funktioner, som der forventes af et SoMe Website og som opfylder de givne krav til funktionaliteter.

Metodeovervejelser(Hanna)

Når man skal udvikle et produkt, er det vigtig altid at vælge hvilken udviklingsmetode man arbejder ud fra. Ved hjælp af en udviklingsmetode, er det nemmere at nå i mål, så vi kan have et produkt eller prototype. Hvor alle detaljer er tænkt godt igennem og man ved hvordan man skal gribe det an.

På nogle områder har vi taget udgangspunkt i five plane modellen. Modellen er med til at opbygge vores site, idet vi vælger det ud som er vigtigst for vores produkt. Vi har f.eks. i strategy plan, været inde at lave en masse research. Vi har valgt at researche omkring nye udfordring der opstår under udvikling af produktet. I scope plane har vi været inde og finde frem til hvilket funktionalitet siden skal have samt ting der foregår bagsiden. Efter dette har vi arbejdet med wireframes, med henblik på hvordan siden skal se ud. Men også hvad der vigtig for vores bruger,

f.eks. at de kan se hvad de klikker på. Derefter vil vi gå i gang med at konstruere selve Yadda sitet. Siden ville blive bygget ved hjælp af node.js med pug template engine samt express som framework til udviklingen og strukturering.

Hashing(Mads)

Eftersom man aldrig bør opbevare et ukrypteret kodeord i en database har vi valgt at gøre brug af bcryptjs³ til at hashe vores brugeres kodeord. Vi har valgt at bruge bcryptjs da det er den hashing vi kender til, men vi har også snakket omkring at gøre brug af md5⁴ da vi kort har haft omkring denne metode i vores undervisning .

Grunden til vi har fravalgt md5 er at denne hashing algoritme er blevet kompromitteret og er derfor mere usikker end bcrypt.js.

³ npm. "bcryptjs". Set 22. maj 2020. <https://www.npmjs.com/package/bcryptjs>.

⁴ "MD5". I Wikipedia, 12. maj 2020. <https://en.wikipedia.org/w/index.php?title=MD5&oldid=956234420>.

Validering(Mads)

På vores site har vi gjort brug af flere former for validering. Heriblandt ved hjælp af JWT (JSON Web Tokens)⁵, Validator⁶ og Express-Validator⁷. Dette er hovedsageligt brugt sammen med Mongoose⁸ i vores user models og ydda models.

Validere bruger via JWT

Ved hjælp af JWT tjekker vores site om en bruger har adgang til de sider hvor vi har tilføjet sikkerhed i forhold til om en bruger er logget ind eller ej. Derudfra har vi opstillet kriterier som gør at en bruger kun kan se de sider som han/hun skal have adgang til i vores navigationsmenu.

Validere Data inden det kommer ind i databasen

For at sikre os at brugere taster ordentlig data ind i vores input felter gør vi brug af Validators, Express-Validator og Mongoose. På den måde sikre vi os at der ikke står alt muligt tilfældigt i f.eks vores email felt.

⁵ auth0.com. "JWT.IO". Set 24. maj 2020. <http://jwt.io/>.

⁶ npm. "validator". Set 24. maj 2020. <https://www.npmjs.com/package/validator>.

⁷ "Getting Started · Express-Validator". Set 24. maj 2020. <https://express-validator.github.io/index.html>.

⁸ "Mongoose v5.9.15: Validation". Set 24. maj 2020. <https://mongoosejs.com/docs/validation.html>.

```
passwordConfirm: {
  type: String,
  required: [true, 'Please confirm your password'],
  validate: {
    // This only works on CREATE and SAVE - Remember when updating
    validator: function (el) {
      return el === this.password;
    },
    message: 'Passwords do not match!',
  },
},
email: {
  type: String,
  required: [true, 'Please provide your email'],
  unique: true,
  lowercase: true,
  validate: [validator.isEmail, ' Please provide a valid email']
},
```

Email confirmation(Mads)

For at bruge vores website kræver det at man har oprettet en bruger og at den er blevet godkendt. Måden det fungerer på er, at når en person har oprettet en bruger vil der blive sendt en confirmation email ud til den email som personen har tastet ind via formen da han/hun oprettede en bruger. Mailen indeholder et unikt token som er sat ind i et link. Når personen klikker på linket i mailen, vil han/hun blive taget til vores hjemmeside, som læser det token der er blevet indsat i linket. Derefter vil den bruger der er blevet oprettet, blive godkendt og kan nu logge ind på vores hjemmeside. Dette sker såfremt at det token som er sat i linket, ikke er blevet modificeret og at brugeren har klikket på linket i confirmation mailen inden udløbstiden er blevet overskredet.

Det data som tokenet indeholder er i grunden krypteret data med information om den bruger der er blevet oprettet. Det indeholder for eksempel information om brugernavn, email, et krypteret password, oprettelsesdato og et token's udløbsdato. Når et tokens udløbsdato er overskredet vil det ikke længere være muligt at bruge

det link som der er sendt ud. Derfor har vi lavet den mulighed at en bruger kan få genudsendt en ny godkendelses email via vores login side såfremt han/hun oplyser sin email og kodeord. Derved har vi tilføjet et ekstra lag af sikkerhed som forhindre at hvis en anden person end den bruger som har oprettet sig på vores hjemmeside, får fat i det token som er sendt ud. Så kan den person kun gøre brug af det inden for den tidsgrænse som der er sat ind i det tokenet ellers er det ubrugeligt.

Research

Sikkerhed(Mads)

Når det kommer til sikkerhed vil der være mange ting at tage fat på. I dette afsnit vil vi komme ind på nogle af de ting som vi har kigget på for at forbedre sikkerheden på vores server.

Database sikkerhed: For at sikre vores database har vi krypteret brugeres passwords med hasing og salting via bcrypt. Derudover var det meningen at vi også ville have lavet en reset token funktion med SHA256⁹, hvor brugeren skulle have mulighed for at resette sit password i tilfælde han/hun havde glemt det.

Brute force angreb: *“Når en angriber prøver at gætte kodeord ved at prøve en million tilfældig genereret kodeord indtil han/hun finder det rigtige”.*

For at forhindre denne type angreb kan vi gøre vores login request langsom hvilket bcrypt egner sig godt til. Dette gør at tiden for hver gang angriberen kan lave et nyt forsøg bliver forøget.

En anden metode vi har gjort brug af er Rate Limiting ved hjælp af express-rate-limit¹⁰ Ved hjælp af express-rate-limit har vi sat en grænse for hvor mange requests det kan laves fra en enkelt ip-adresse pr time.

```
const limiter = rateLimit({
  max: 1000,
  windowMs: 60 * 60 * 1000,
  message: 'Too many requests from this IP, please try again in an hour!',
});
```

En ting vi yderligere kunne have gjort ville være at sætte en grænse for hvor mange login forsøg en person har pr time.

⁹ “SHA-2”. I Wikipedia, 16. maj 2020. <https://en.wikipedia.org/w/index.php?title=SHA-2&oldid=956953707>.

¹⁰ npm. “express-rate-limit”. Set 22. maj 2020. <https://www.npmjs.com/package/express-rate-limit>.

XXS Angreb: Cross Site Scripting angreb. *“Når en angriber forsøger at få adgang til fx. localStorage, for at få adgang til data ved at køre scripts på vores side, ved at taste skadelig data ind i input felter”.*

For at forhindre dette har vi sat validators på alle vores input felter ved hjælp af forskellige node js moduler. Derudover har vi gjort at vores JWT bliver opbevaret i HTTPOnly cookies som gør at browseren kun kan modtage og sende cookies men ikke modificere eller tilgå den på nogen måde. Det gør det umuligt for en angriber at stjæle en JWT.

I backenden har vi brugt en middleware kaldet Helmet¹¹ til at sætte nogle speciale HTTP headers som også gør det sværere en angriber at benytte XSS angreb.

Denial of service (DDOS): For at forhindre denial of service angreb har vi sat en grænse på 10kb i størrelsen af data som kan blive sat i body på en post / patch request. Derudover har vi stadig vores Rate Limit som også hjælper med denne type angreb.

CONFIG.ENV: En anden type af sikkerhed er at lave en config.env fil som opbevare token secrets, username og passwords som vi har beskrevet i vores forord i denne rapport.

Nodemailer¹² (Mads)

For at vi har kunne sende emails har vi gjort brug af nodemailer modulet. Via nodemailer laver man en transporter hvor man sætter data ind fra ens SMTP. Det vil typisk være en host adresse, port nummer, authentication i form af username og password. Derefter har man mulighed for at definere en besked man vil sende ud til ens bruger. Dette har vi gjort smartere ved at bruge pug templates som vi kalder på ved hjælp af en funktion hver gang vi vil sende en email ud til vores brugere. På den måde har vi fået lavet email skabeloner.

¹¹ npm. “helmet”. Set 22. maj 2020. <https://www.npmjs.com/package/helmet>.

¹² “Nodemailer :: Nodemailer”. Set 24. maj 2020. <https://nodemailer.com/about/>.

Multer¹³ (Mads)

For at give brugeren mulighed for at uploade en avatar til sin profil og til vores server har vi været nødt til at implementere en ny middleware kaldt “multer”. Med denne middleware har vi haft mulighed for at tage en “file uploads” filnavn og sætte den ind i vores database til senere brug i vores “yaddas”.

Derudover via “multer” validere vi også at filtypen er en imagefil så brugeren ikke har mulighed for at uploade andre filtyper til vores server. Inden image filen bliver gemt i vores database bliver den konverteret om til .jpeg og får en dimension på 500x500px via memory storage som gemmer filen som en buffer der bliver behandlet af en anden middleware kaldet sharp¹⁴. Når dette er gjort bliver den modificerede gemt i vores avatar mappe på vores server.

Moment.js (AK)

Moment.js er et JavaScript library, som gør det muligt at parse, validere, formatere og wrappe Date objekter, så de bliver lettere at arbejde med.¹⁵

I vores Yadda Shema, har vi property *created* med typen Date, som default er Date.now. Dette giver os en meget lang og ikke særlig læsbar Timestamp, når vi trækker dataen ud fra databasen i vores Pug filer. Vi har derfor brugt Moment.js til at gøre Timestamp for hvornår en Yadda er oprettet mere læsbar ved at wrappe vores yadda.created object med en fromNow() funktion.

```
div.post-created
a(href="" title=`${yadda.created}`) #{moment(yadda.created).fromNow()}
```

Dette gør at en Yadda's Timestamp i stedet vil vise tiden fra nu, til da den blev oprettet.

¹³ npm. “multer”. Set 24. maj 2020. <https://www.npmjs.com/package/multer>.

¹⁴ npm. “sharp”. Set 24. maj 2020. <https://www.npmjs.com/package/sharp>.

¹⁵ <https://momentjs.com/>, senest besøgt d. 22-05-20

#tags (AK)

Udover tekst og billede, skal en Yadda også kunne indeholde tags. Tags har præfix på tegnet # og kan indeholde tal fra 0-9, bogstaver fra aA-zZ og specielle tegn som underscore.¹⁶

```
let tags = /^(^|\B)#!([0-9_]+\b)([a-zA-Z0-9_]{1,30})(\b|\r)/g;
```

Til at tjekke og matche på ord der skrives ind i tag-inputtet, har vi brugt et Regex (Regular expression), som kan tilpasses til ens behov. Et Regex er en sekvens af tegn, der definerer et mønster til søgning eller matching af tekststreng. ¹⁷ Det gør det derfor også muligt at søge efter tags i Yadda's.

¹⁶ <https://stackoverflow.com/questions/42065872/regex-for-a-valid-hashtag/42551826>, senest besøgt d. 22-05-20

¹⁷ https://en.wikipedia.org/wiki/Regular_expression, senest besøgt d. 22-05-20

Tema (AK)

Brugeren skal have mulighed for at skifte mellem to temaer. Vi har valgt at implementere et lyst og et mørkt tema i vores applikation. Et krav til opgaven er at Brugerens valg af tema skal afspejle sig i vores database, ved at være true eller false. Vi har istedet valgt at gemme brugerens valg af tema med localStorage¹⁸, hvilket betyder at dataen bliver gemt uden en udløbsdato, selv hvis browseren lukkes. Dette har vi gjort i overensstemmelse med vores lærer som har givet grønt lys til denne ændring.

```
1  "use strict";
2  const toggleSwitch = document.querySelector('.theme-switch input[type="checkbox"]');
3  const currentTheme = localStorage.getItem('theme');
4
5  if (currentTheme) {
6    document.documentElement.setAttribute('data-theme', currentTheme);
7
8    if (currentTheme === 'dark') {
9      toggleSwitch.checked = true;
10   }
11 }
12
13 function switchTheme(e) {
14   if (e.target.checked) {
15     document.documentElement.setAttribute('data-theme', 'dark');
16     localStorage.setItem('theme', 'dark');
17   }
18   else {
19     document.documentElement.setAttribute('data-theme', 'light');
20     localStorage.setItem('theme', 'light');
21   }
22 }
23 toggleSwitch.addEventListener('change', switchTheme, false);
```

Mørkt tema være aktivt indtil brugeren selv vælger at skifte til lyst tema, og omvendt. At gemme dataen med localStorage i stedet for serveren, giver i sidste ende også mere mening. Brugeren vil ofte tilgå siden via den samme computer, og preference af tema skal være aktivt selvom brugeren er logget ud. Vores kode til ændring af tema er baseret på et eksempel fundet på dev.to.¹⁹

¹⁸ https://www.w3schools.com/jsref/prop_win_localstorage.asp, senest besøgt d. 22-05-20

¹⁹ <https://dev.to/ananyaneogi/create-a-dark-light-mode-switch-with-css-variables-34l8>, senest besøgt d. 18-05-20

Analyse

Specifikke krav (Hanna)

For at vi kan tage ordentligt hul på selve opgaven, valgte vi at starte med at identificere de specifikke krav for hjemmesiden. Ved at lave denne liste har vi god mulighed, for at få et godt overblik over selve opgaven. Metoden giver os lov til at arbejde med funktionaliteten med henblik på systembeskrivelser og hvad selve systemet skal kunne gøre. Herudover vil vi også kigge på, hvilket content som der skal være på selve hjemmesiden. Kigger man på vores content kan man se at vores primære mål er at kunne læse og skrive Yaddas. Dog er der lavet plads ude i højre side til reklamer. Eftersom vi ved hvilket primære content der skal være. Kan man allerede der begyndte at lave en liste over funktioner. Første punkter er vi skulle have en side hvor vi kan vise disse Yaddas, som skal kunne kommenteres. Hvor man først kan se formen til at kunne kommentere, efter at have klikket på en yadda. Hver Yadda skal kunne indeholde et profilbillede, dato/tid, username og eventuelle tags de laver. Ud fra det finder vi automatisk ud af, at der er brug et login system. Brugeren skal have mulighed for, at register sig til sitet. Vores plan for denne proces er brugeren, skal kunne registrere sig via email authentication. Hvilket ville sige de skal bekræfte en mail for at kunne oprette en bruger. Derudover skal de igennem en meget ofte anvendt proces i oprettelse af profil, username, email, navn og kodeord. Brugerens kode vil blive hashet, så sikkerhed imod hakning er højere. Efter oprettelse skal man så logge ind, her ville man have en bruger som andre kan følge. Alle brugerne på sitet vil have mulighed for at se deres følgere og hvem de følger. Inde på brugerens profil, har de mulighed for at vælge tema på hjemmesiden. Temaet ville blive gemt i localStorage, sådan at det valgte temaet bliver gemt for brugeren.

Wireframes (Hanna)

Inden vi valgte at gå i gang med at konstruere koden, valgte vi at lave wireframes. Disse skulle være med til at samle vores tanker om, hvordan vores site skulle se ud. Vores Wireframes er lavet på forskellige måder, men primært bygget på low- og high fidelity. Low fidelity for os er udarbejdet både på papir og i Adobe XD. Ved at gøre det på denne måde, får vi hurtig skabt en tidlig ide om hvordan applikationen skal se ud visuelt. Idet vi ikke har en specifik målgruppe, har vi ikke rigtig kunne specificere et bestemt design. Så vi har valgt at søge inspiration fra diverse steder, hvor sider minder lidt om formål med sitet. På en af vores high-fidelity frames, har vi valgt at lægge ikoner og farver ind. Sådan det var tættere et slutresultat i forhold til designet (se bilag 2). I vores navigationsdesign har vi valgt at gøre brug af global navigation. Denne navigationsmetode er oplagt for os, efter vi gerne ville have en simpel side. Global navigation er bygge på store hovedpunkter, så man kan altid komme tilbage til. Når man ankommer på forsiden, er formålet at fange brugeren hurtig. Derfor har vi valgt der skal være top Yaddas på forsiden. Forsiden skal nemlig sende et budskab om hvad sitet handler om, og fange brugeren. Efter login vil brugeren komme ind på Dashboard. Dashboard er stedet hvor brugeren har mulighed for at lave en Yadda. Udover at lave Yaddas, er det også stedet hvor brugeren kan se de Yaddas fra dem de følger. Idet brugerne skal kunne svare tilbage på andre Yaddas, er vores tanke at lave en tråd under opslaget. Tråden skal være med til at guide brugerne, sådan de nemmere kan finde ud af hvor hvilken Yadda hører til.

Konstruktion

Node.js og Express (AK)

Node.js er brugt til opsætning og håndtering af vores server og Express er brugt som application framework til udvikling og strukturering af vores web applikation. Express hjælper også med håndtering af vores responds og requests via routing. Med Node's package manager system(*npm*), er det nemt at finde moduler til ens applikationer, som f.eks Express.

Pug (AK)

Vi har tidligere brugt *.ejs* som template engine, men har denne gang valgt at bruge Pug.ejs.²⁰ Dette valg er primært på baggrund af at, opgaverne i sikkerhed op til projektet er lavet med Pug, så vi har derfor haft et godt udgangspunkt. Det har også givet os mulighed for at lære en ny template engine at kende. Derudover er Pug's syntax simpel, let at læse og kræver mindre tastetryk, da man ikke skal tænke på at starte og lukke sine tags. Dette gør kodningsprocessen hurtigere og mere produktiv. Man skal dog huske på at indrykke tags og attributter ordentligt, ellers melder siden fejl i koden.

Med Pug er det også muligt at render data hentet via API, genbruge kode i andre filer og køre JavaScript i vores templates. Pug er også det oplagte valg når der arbejdes med databaser og API, da vi ved hjælp af interpolation og iteration, kan trække data ud, direkte i vores Pug filer.

²⁰ <https://pugjs.org/api/getting-started.html>, senest besøgt d. 22-05-20

```
each yadda in yaddas
  div.post-container
    div.post-image
      img(src="../../../images/avatar/" + yadda.avatar onerror="this.src='../images/avatar/avatar.png'" width="100px" height="100px")
    div.post-content
      div.post-head
        div.post-username
          if yadda.username != user.username
            form(action="/userProfile" method="POST" id="yaddaUserNameForm")
              input(type="hidden" name="yaddaUsername" value=`${yadda.username}` id="yaddaUsername")
              input(type="submit" value=`${yadda.username}`)
          else
            a(href="/profile") #{user.username}
        div.post-created
          a(href="" title=`${yadda.created}`) #{moment(yadda.created).fromNow()}
        div.post-delete
          if user.username == yadda.username
            form(action="/api/v1/yaddas/delYadda" method="POST")
              input(type="hidden" name="_id" value= yadda._id)
              button(type="submit") Delete
          else
            div.post-delete(style="display: none;")
        div.post-text
          p #{yadda.text}
        div.post-tags
          a(href="") #{yadda.tags}
      div.post-bottom
```

Templates (AK)

Template Inheritance er en feature i Pug, som gør det muligt at opdele sine templates i mindre og mere enkle filer.²¹ Man opdeler templates ved at bruges keywords *block* og *extend*. *Block* er en blok som et child template kan erstatte, som for eksempel *block content*. For at udvide et layout i en anden Pug fil, bruges keyword *extend* efterfulgt af stien til ens parent template. I vores tilfælde er vores parent template kaldt *default*, indeholdende et *block content*, vores head med vores links og scripts, vores *includes* *_header* og *footer*.

```
doctype
html
  head
    title VVV | #{title}
    meta(charset='utf-8')
    meta(name="viewport" content="width=device-width, initial-scale=1.0")
    link(href="https://fonts.googleapis.com/css2?family=Montserrat:wght@200;300;500;600&display=swap" rel="stylesheet")
    link(href="https://fonts.googleapis.com/css2?family=Open+Sans:wght@400;600&display=swap" rel="stylesheet")
    link(rel="icon", type="image/svg+xml", href="/favicon.svg")
    link(rel="stylesheet", href="/stylesheets/style.css")
    link(rel="stylesheet", href="/stylesheets/slidtheme.css")
    script(type="module" src="/javascripts/darkTheme.js")
    script(src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js")
    // script(src="https://cdnjs.cloudflare.com/ajax/libs/axios/0.19.2/axios.min.js")

  body
    // Header
    include _header

    // CONTENT
    block content

    // FOOTER
    include _footer

  .adswrapper
    .adstextbox: iframe(width="100%" height="100%" src="https://www.youtube.com/embed/Cxqca4RQd_M" frameborder="0" allow="accelerometer;

    script(src="/javascripts/clock.js")
    script(src="/javascripts/bundle.js")
    script(src="https://kit.fontawesome.com/a99e7cd5cc.js")
    script(type="module" src="/javascripts/main.js")
```

Med *include* kan man indsætte indhold fra en Pug over i en anden. I vores *_header* har vi vores navigation og i footer har vi vores footer. Når vi bruger *extend default* i vores filer, vil vi få vist vores navigation og footer på vores sider. Vi slipper for at skulle kopiere lange stykker kode ind i hver eneste fil, og vi kan i stedet nøjes med at skrive *extend default* som det første i vores filer.

²¹ <https://pugjs.org/language/inheritance.html>, senest besøgt d. 22-05-20

Schemas/models(Hanna)

Da vi har valgt at gøre brug, af mongoose package. Den package er et værktøj man kan arbejde med, så der kommer et bestemt syntax lag mellem ens applikation og den database man bruger. Udover det har vi også mulighed for, med mongoose package at omdanne vores data. Når vi omdanner denne data, kan vi tilpasse det en vis model struktur. For at skabe vores yadda website, har vi gjort meget brug af model struktur. Strukturen gør at vi kan hente data fra de forskellige steder. Vi har lavet to schemas en til yadda og user, disse schemas er med til at lægge et frame for hvordan vi ville have vores data struktureret. I forhold til de forskellige objects, der skal vores i vores applikation. Vores schemas er bygget på på key:properties. Hvilket ville sige at keys er de ting vi defineret på forhånd i vores database. Hvor vores properties er hvordan vi ville have de skal vises. Måden vi har lavet dem på kan se på billederne nedenunder:

```
1 const mongoose = require('mongoose');
2
3 const YaddaSchema = new mongoose.Schema({
4   username: {
5     type: String,
6     required: true,
7   },
8
9   image: {
10    type: String,
11    default: null,
12  },
13
14  text: {
15    type: String,
16    required: true,
17  },
18
19  tags: {
20    type: [String],
21    default: null,
22  },
23
24  avatar: {
25    type: String
26  },
27
28  reply: {
29    type: String,
30    default: null,
31  },
32
33  created: {
34    type: Date,
35    default: Date.now,
36  },
37 });
38
39 const Yadda = mongoose.model('Yadda', YaddaSchema, 'yadda');
```

```
1 const crypto = require('crypto');
2 const mongoose = require('mongoose');
3 const validator = require('validator');
4 const bcrypt = require('bcryptjs');
5
6 const usersSchema = new mongoose.Schema({
7   username: {
8     type: String,
9     required: [true, 'Please provide a username'],
10    unique: true,
11  },
12  role: {
13    type: String,
14    enum: ['member', 'moderator', 'admin'],
15    default: 'member',
16  },
17  password: {
18    type: String,
19    required: [true, 'Please provide a password'],
20    minlength: 4,
21    select: false, // Makes sure password is never shown in any output
22  },
23  passwordConfirm: {
24    type: String,
25    required: [true, 'Please confirm your password'],
26    validate: {
27      // This only works on CREATE and SAVE - Remember when updating user (do not use findOneAndUpdate) use SAVE!!!
28      validator: function (el) {
29        return el === this.password;
30      },
31      message: 'Passwords do not match!',
32    },
33  },
34  email: {
35    type: String,
36    required: [true, 'Please provide your email'],
37    unique: true,
38    lowercase: true,
39    validate: [validator.isEmail, 'Please provide a valid email'], // Checks if this field is an email string - https://www.npmjs.com/package/validator
40  },
41  firstName: {
42    type: String,
43    required: [true, 'Please enter your firstname'],
44  },
45  lastName: {
46    type: String,
47    required: [true, 'Please enter your lastname'],
48  },
49 },
```

```
49   avatar: {
50     type: String,
51     default: 'avatar.png',
52   },
53   isConfirmed: {
54     type: Boolean,
55     default: false,
56   },
57   theme: {
58     type: Boolean,
59     default: false,
60   },
61   following: [],
62   created: {
63     type: Date,
64     default: Date.now,
65   },
66   passwordChangedAt: Date,
67   passwordResetToken: String,
68   passwordResetExpires: Date,
69   active: {
70     type: Boolean,
71     default: true,
72     select: false,
73   },
74 });
75
76 // Pre-saved middleware (Document middleware)
77 // I mellen det moment hvor vi modtager data og det moment hvor data bliver sendt til vores database
78 // Det perfekte tidspunkt at manipulere data til encryption
79 userSchema.pre('save', async function (next) {
80   // Only run this function if password was actually modified
81   if (!this.isModified('password')) return next();
82
83   // Hash password with bcrypt and cost of 12
84   this.password = await bcrypt.hash(this.password, 12);
85
86   // Stopper passwordconfirm fra at blive sendt til vores database
87   this.passwordConfirm = undefined;
88   next();
89 });
90
91 userSchema.pre('save', function (next) {
92   if (!this.isModified('password') || this.isNew) return next();
93
94   this.passwordChangedAt = Date.now() - 1000; // Makes sure the token is always created after the password has been changed
95   next();
96 });
```

```
91 userSchema.pre('save', function (next) {
92   if (!this.isModified('password') || this.isNew) return next();
93
94   this.passwordChangedAt = Date.now() - 1000; // Makes sure the token is always created after the password has been changed
95   next();
96 });
97
98 // tjekker om en user er active
99 userSchema.pre(/^find/, function (next) {
100   // this points to the current query
101   this.find({ active: { $ne: false } });
102   next();
103 });
104
105 // Instance method - available to all documents of a certain collection
106 // Login sammenligning af passwords
107 userSchema.methods.correctPassword = async function (candidatePassword, userPassword) {
108   return await bcrypt.compare(candidatePassword, userPassword);
109 };
110
111 // Tjekker om en bruger forsøger at logge ind efter han/hun har skiftet kodeord efter JWT token er oprettet
112 userSchema.methods.changedPasswordAfter = function (JWTTimestamp) {
113   if (this.passwordChangedAt) {
114     const changedTimestamp = parseInt(this.passwordChangedAt.getTime() / 1000, 10);
115
116     return JWTTimestamp < changedTimestamp;
117   }
118
119   // False Means NOT changed
120   return false;
121 };
122
123 // Token som bliver sendt til brugeren for at resette sit password
124 userSchema.methods.createPasswordResetToken = function () {
125   const resetToken = crypto.randomBytes(32).toString('hex');
126
127   this.passwordResetToken = crypto.createHash('sha256').update(resetToken).digest('hex');
128   console.log({ resetToken }, this.passwordResetToken);
129
130   this.passwordResetExpires = Date.now() + 10 * 60 * 1000;
131
132   return resetToken;
133 };
134
135 const User = mongoose.model('User', userSchema, 'user');
136
137 module.exports = User;
138
```

Controllers(Sune/Mads/AK)

UserController(Mads)

Via vores userController har vi kontrol over alt hvad en bruger gøre på vores hjemmeside i forhold til data omkring sig selv. Dette indebærer også når en bruger bliver oprettet, logger ind og logger ud. Hovedsagelig er userControllerens funktion til når en bruger vil opdatere data omkring sig selv. Dette kunne f.eks være sit rigtige navn, avatar eller adgangskode

authController(Mads)

I vores authController styre vi alt hvad der har med authentication af vores brugere af gøre. Dette omhandler både JWT, om en bruger har adgang til specielle sider på vores server og om brugeren i det hele taget er oprettet og logget ind på vores social media site.

indexController(Sune)

I vores indexController har vi valgt at styre alle de funktioner og ting som har noget at gøre med at det skal vises frem på siden på den ene måde, dette omhandler både Yaddas, Following, DarkTheme og meget mere. Det er i denne controller vi render rigtigt mange af vores ting som skal kunne ses af brugeren.

Et par eksempler kunne være følgende

```
// Profile
exports.profile = async function (req, res) {
  let onlyUser = { username: req.user.username };
  let yadda = await mon.retrieve(Yadda, onlyUser, { sort: { created: -1 } });
  let users = await mon.retrieve(User, {});
  let isFollowing = { following: req.user.username };
  let following = await mon.retrieve(User, isFollowing);
  console.log(following);
  res.render('profile', {
    title: 'Profile',
    subtitle: 'Welcome to your profile',
    user: req.user,
    yaddas: yadda,
    users: users,
    followers: following,
  });
};
```

```
// Dashboard
exports.dashboard = async function (req, res) {
  let replychk = { reply: null };
  let yadda = await mon.retrieve(Yadda, replychk, { sort: { created: -1 } });

  let replychkNotNull = { reply: { $ne: null } };
  let yaddaNotNull = await mon.retrieve(Yadda, replychkNotNull, { sort: { created: 1 } });

  res.render('dashboard', {
    title: 'Dashboard',
    subtitle: 'Welcome to your dashboard',
    user: req.user,
    yaddas: yadda,
    yaddasNotNull: yaddaNotNull,
  });
};
```


På de 2 billeder ovenover, kan det ses hvordan vi henter noget bestemt data ud fra vores User & Yadda Collections, som vi så nedenunder render ud på siden.

yaddaController(Sune)

I vores yaddaController har vi valgt at lave alt hvad har med yaddas at gøre. Dette er at slette, indsætte & indsætte en kommentar Yadda. Det vil sige at vi har valgt at gøre sådan at alt som skal ind i databasen ligger i denne controller.

Delete:

```
6  exports.delYadda = async function (req, res) {
7    let del = { _id: req.body._id };
8    try {
9      let cs = await mon.remove(Yadda, del);
10     res.redirect('back');
11   } catch (e) {
12     console.log(e);
13   }
14 };
```

Indsæt Yadda m. tags:

```
16 exports.postYadda = async function (req) {
17   //finder #tags
18   let tag = req.body.tags; //kan også erstattes med text
19   //regex - regel for indhold af hashtag
20   let tags = /^(^|\B)#!([0-9_]+\b)([a-zA-Z0-9_]{1,30})(\b|\r)/g;
21   //matcher tags med tag
22   let split = tag.split(' ');
23   let match = tag.toLowerCase().match(tags);
24   //let join = tag.join(", ");
25
26   let yadda = new Yadda({
27     // create object in db-format
28     username: req.body.username,
29     text: req.body.text,
30     avatar: req.body.avatar,
31     tags: split,
32     tags: match,
33     //tags: join
34   });
35   try {
36     let cs = await yadda.save();
37     return;
38   } catch (e) {
39     console.log(e);
40   }
41   };
```

Indsæt en yadda som Reply:

```
43 exports.replyYadda = async function (req, res) {
44   let yadda = new Yadda({
45     username: req.body.username,
46     text: req.body.text,
47     avatar: req.body.avatar,
48     reply: req.body.reply,
49   });
50   let cs = await yadda.save();
51   res.redirect('/dashboard');
52   };
```

tagController(AK)

Vores tagController håndterer alt der har med #tags at gøre, udover når man skriver en Yadda. Her bliver de sendt med i vores yaddaController når man poster en Yadda.

```
exports.postYadda = async function (req) {  
  //finder #tags  
  let tag = req.body.tags; //kan også erstattes med text  
  //regex - regel for indhold af hashtag  
  let tags = /^(^|\B)(?![\0-9_]+\b)([a-zA-Z0-9_]{1,30})(\b|\r)/g;  
  //matcher tags med tag  
  let split = tag.split(' ');  
  let match = tag.toLowerCase().match(tags);  
  //let join = tag.join(", ");  
  
  let yadda = new Yadda({  
    // create object in db-format  
    username: req.body.username,  
    text: req.body.text,  
    avatar: req.body.avatar,  
    tags: split,  
    tags: match,  
    //tags: join  
  });  
  try {  
    let cs = await yadda.save();  
    return;  
  } catch (e) {  
    console.log(e);  
  }  
};
```

I *tagController*en laver vi funktionen til at kalde vores #tags side, med en *mon.retrieve* på vores Yadda data og en require på user. *searchTags* gør det muligt at søge på #tags ved at vi laver en *mon.retrieve*, hvorefter de bliver sorteret efter de nyeste Yaddas øverst.

```
1 const mon = require('../models/mongooseWrap');
2 const Yadda = require('../models/Yadda');
3
4 // Tags
5 exports.tags = async function (req, res) {
6   let yadda = await mon.retrieve(Yadda, {}, { sort: { created: -1 } });
7   res.render('tags', {
8     title: '#tags',
9     subtitle: 'Explore #tags',
10    user: req.user,
11    yaddas: yadda,
12  });
13 };
14
15 //find tags
16 exports.searchTags = async function (req, res) {
17   let yadda = await mon.retrieve(Yadda, { tags: req.body.tags }, { sort: { created: -1 } });
18   res.render('tags', {
19     title: '#tags',
20     user: req.user,
21     yaddas: yadda,
22   });
23 };
```

errorController(Mads)

For at give os selv et bedre overblik når der sker en fejl på vores side har vi lavet en error controller. I vores error controller har vi blandt andet lavet en funktion der rapportere om en unik værdi rent faktisk er unik. Hvis en der bliver forsøgt at lagre data om et brugernavn der allerede eksistere i vores database vil vores error controller sende en besked tilbage at brugernavnet allerede findes. Samme princip er lavet med email, Invalid JWT, udløbet JWT, validation error. Kort sagt vores error controller sender en besked når den opdager en fejl på vores side. Vi har valgt at opstille det på en sådan måde at i development mode får man vist en besked med alle error detaljerne, mens at i production mode vil der kun blive vist en besked som en almindelig person også til dels forstår uden alle detaljer som vigtig for os udviklere.

adminController(Mads)

I den perfekte verden ville der være tid til alt. Desværre var dette en del af vores projekt som vi har fravalgt da den ikke var et krav. Tanken omkring en adminController var som udgangspunkt at lave roller til hver enkelt bruger hvor der

ville være mulighed for at have admins, moderators og users. Admins skulle have mulighed for at slette brugere og ændre deres data. Derudover skulle en admin også have mulighed for at slette og ændre i Yadda/comments. En moderator ville skulle have adgang til de næste de samme funktioner, hvorimod en bruger kun ville have adgang til de funktioner som er tilgængelig på vores nuværende side.

Routes(Sune/Mads)

indexRoutes (Sune)

I vores indexRoutes, har vi valgt at lægge alle vores GET funktionskald, hvor der var behov for at få noget fra vores MongoDB. Der er dog to enkelte POST funktioner som vi er nødt til at have herinde, fordi at det er nødvendigt at GET, for så derefter at POST. I bund og grund, er indexRoutes, direkte forbundet til IndexController, Dette betyder at alt som bliver vist på siden fra databasen, bliver kaldt via denne router.

usersRoutes (Mads)

Hovedsageligt er vores userRoutes alle vores post og patch request der har direkte tilknytning til user actions. Det vil sige forbindelse til vores database med at oprette en bruger, logge ind/logge ud, få tilsendt en ny confirmation email, opdatering af password og user settings. Dette er direkte tilknyttet vores database hvilket også er illustreret via alle usersRoutes path "api/v1/users" Det samme er gældende for vores yaddaRoutes, men omhandler alt hvad der har med yaddas at gøre.

yaddaRoutes (Sune)

YaddaRoutes, er blevet brugt til at "route" alle yadda funktioner som havde en POST på formen. Sådan så at vi på en nem og overskuelig måde kunne finde dem. og se forskel på dem.

User Management og Users API: (Mads)

For at kunne bedre forstå user management på vores hjemmeside og server har vi valgt at lave denne guide for at gøre det nemmere at forstå hvad der er muligt og hvad der ikke er muligt i forhold til at administrere brugere.

Som det første skal det selvfølgelig være muligt at oprette en bruger. Dette gøres ved at gå ind på vores side og trykke på *Sign Up*. Når dette er gjort og brugeren trykker på *Create User* vil der blive lavet en Email Token via JWT som sendes i en mail som et link til brugerens email adresse. Derefter vil brugeren blive omdirigeret til vores *Sign in* side.

I vores database er brugeren nu oprettet med følgende key values:

- `_id`: random generated id
- `role` member
- `isConfirmed` false
- `theme` false
- `following` [],
- `active` true
- `username` <Valgt Brugernavn>
- `email` <Valgt Email>
- `password` <encrypted password>
- `firstName` <Valgt navn>
- `lastName` <Valgt efternavn>
- `created` Oprettelses dato og tid

Indtil brugeren klikker på linket i den sendte email vil det ikke være muligt for brugeren at logge ind da *isConfirmed* er sat til *false*. Det bliver tjekket efter når brugeren klikker på *login* via en middleware funktion som gør følgende:

1. Tjekker at Email og Password eksistere.
2. Tjekker at Email og Password er korrekt
3. Om brugeren er verificeret via email (*isConfirmed*)
4. Opretter en ny Token som der tjekkes efter når brugeren vil tilgå sider der kræver at han/hun er logget ind.

Da der er sat en timer på 10 min hvor brugeren kan verificere sin email har vi også lavet en funktion der gør at brugeren kan få gensendt sin Email Token såfremt han eller hun ikke når at verificere sin email i tide. Funktionen findes som et link på *Sign In* siden.

Når en bruger at logget ind vil han/hun have mulighed for at opdatere sit *username*, *email*, *firstName*, *lastName*, *avatar* og *password*. Dette kan gøres i *Account Settings* menuen. Alt data der bliver indtastet bliver valideret af flere middleware validators som checker efter skadelig data. Det samme er gældende for når man opretter en bruger.

Ting vi gerne ville have tilføjet forhold til users som vi ikke har lavet, da de ikke var et krav til opgaven:

- Reset Password / Forgot Password- Hvis en bruger har glemt sit kodeord og ikke kan logge ind
- Delete User - Slette sin bruger. Dette er hvad *active* feltet i vores database illustrere. Når en bruger sletter sig selv vil brugeren stadig eksistere i vores

database, men feltet *active* som default står til **true** vil ændre sig til **false** og det vil ikke være muligt at logge ind på denne bruger længere.

- Administrator Panel / Moderator Panel - Delete User Yaddas - Remove Users

Der er dog lavet funktioner i vores controllers som gør at disse ting vil kunne blive tilføjet nemt på et senere tidspunkt.

Evaluering af proces og produkt

Proces (AK)

På grund af de nuværende omstændigheder, har vi alle arbejdet hjemmefra. Vi har dog alligevel snakket sammen dagligt, hvor vi har holdt møder over Discord.²² Her har vi også kodet live og arbejdet sammen om nogle mere krævende funktionaliteter, eller hvis der var brug for et nyt sæt øjne.

Vi har haft mest fokus på vores funktionaliteter, hvor nogle måske har haft en højere prioritet end andre. Vi har dog alligevel forsøgt at få opfyldt alle kravene til opgaven så fyldestgørende som muligt. Nogle af vores funktionaliteter har nogle mangler, men vi synes selv at vi er nået godt i mål med opgaven, som opfylder både kravene og formålet med vores applikation.

Evaluering af produkt(AK)

Som tidligere nævnt har vores applikation nogle mangler, men der er småting i forhold til hvad vi har nået.

Lige nu vil man på vores Dashboard kunne se Yaddas fra alle brugere på tidslinjen. Her er det meningen at der kun skal vises Yaddas fra dem man selv følger.

²² <https://discord.com/>

Når der klikkes på den første Yadda i en tråd, er det meningen at comments til den valgte yadda vises. Det samme er meningen med formen til at kommentere en Yadda. Lige nu vil comments og formen hele tiden være synlig under den første Yadda.

Det skal også være muligt at lave en comment til en comment osv. Vi har fokuseret på at få første del til at virke, og man kan indtil videre kun svare på den første Yadda.

På profilsiden er det meningen at den skal sortere comments til en Yadda fra så de ikke vises på ens egen tidslinje, eller i hvert fald vise om det er en comment på en anden Yadda. Lige nu vil både Yaddas og comments vises på ens egen tidslinje. Når man laver en Yadda er det meningen at den udover tags og tekst, også skal kunne indeholde et billede. Lige nu er det ikke muligt i vores applikation, fordi vi igen har haft fokus på at selve funktionaliteten i at kunne skrive en Yadda.

Når man klikker på followers eller following på en profil, vil man kunne se hvem brugeren følger og følges af. Meningen var her at man også skulle kunne klikke på hver enkelt bruger, hvor man så ville blive navigeret til brugerens profil.

På siden #Tags er det muligt at søge efter tags, men når man indtaster tagget vil man være nødt til at starte det med prefixet # efterfulgt af tagget, som skal være skrevet præcist som det man søger efter. Her kunne det være smart hvis det var muligt at søge på en del af et tag.

Vi har udover kravene også kigget på nogle ekstra features man kunne implementere i vores applikation. Dette var dog kun noget vi ville kigge på, hvis vi havde tid til overs.

Når en bruger logger ind kunne man gøre det muligt at logge ind med enten username eller email, alt efter hvad brugerens preference er.

Hvis en bruger har glemt sit password skulle der inde i Account settings kunne være mulighed for at få det nulstillet via en email token. Derudover skal der også være en mulighed for at slette sin bruger.

Vi har også lavet tre ekstra sider, men som ikke rigtig har en funktion lige nu, udover at være der som placeholders. Den første side er Messages, som vi havde tiltænkt til private beskeder fra en bruger til en anden. Næste er Explore hvor det var meningen at man skulle kunne søge efter andre brugere på YaddaYaddaYadda, ligesom vi har gjort med tags. Dette ville være en smart funktionalitet i forhold til at finde brugere man ønsker at følge, da meningen med Dashboard er at der kun skal vises Yaddas fra de personer man følger. Derudover er der tilføjet et Adminpanel og Moderatorpanel til håndtering af brugere og Admin funktioner. Admin og Moderator panel kan kun ses hvis man retter en users "role" value i vores MongoDB til enten "admin" eller "moderator" standard sat som "member".

Udfordringer(Hanna)

En udfordring som vi har haft igennem hele projektet, har været omkring vores struktur i repo samt filer. I starten af projektet havde vi en overordnet god struktur, men i det at vi begyndte at tilføje flere package, kom der mange filer. I takt med vi også lavede ny filer og mapper, for at være sikre på vi ikke modarbejde de filer vi havde i forvejen. Endte der med at opstå mange filer, som faktisk ikke blev brugt til sidst.

Udover struktur har vi også haft en del udfordringer, omkring at opbygge vores Yadda side. En vigtig del for at oprette sig som bruger inde på Yadda, er man skal modtage en bekræftelses mail. For at få det til at faldt vi frem til, at man skulle bruge nodemailer. Hvilket gør at man kan sende emails fra Node.js, for så at kunne bekræfte denne mail. Resultatet blev at man inde på mailtrap kan oprette en fake mail, der passer med en server sådan man kan modtage sin mail.

Efter man er logget ind, kommer brugeren ind til deres dashboard. På den side, ville det være muligt at se yaddas fra dem de følger. Udfordring lå i et få vidst disse yaddas på forsiden. Vi endte med at løse dette ved hjælp af et for each loop, her har vi mulighed for at vise de ting som bliver postet.

Når brugerne kan læse andres yaddas, er det vigtig de også kan kommentar på dem. Derfor arbejde vi meget på at få denne funktion til at virke. Det som var bragt os besvær, var vi skulle få svaret til at passe med det yadda id de besvarede.

Når du laver en Yadda skal der være mulighed for, at lave hashtags som man bagefter kan søge på. Et problem som opstod efter vi havde lavet tags inputfeltet, var hvordan vi skulle splitte dem. I starten hvis man skrev tags og lavede et mellemrum eller komma, så blev det vist som et langt tag. I sidste ende fik vi dette løst med en split.

Efter vi har fået vist kommentarerne, var vores mål at få dem vist under det enkelte yadda. Derfor har vi været inde at arbejde med accordion. Accordion er en god måde at skifte samt vise imellem, det content der skal være nedenunder. Dog endte vi med at gå væk fra accordion. Da vi ikke syntes at kunne få den til at ramme en box uden for funktionen.

Vi havde en del problemer i forbindelse med at skulle have skrevet data ud på andre brugere, Dette vil sige når man er logget ind ville man gerne have mulighed for at kunne se data på andre users som ikke er logget ind. Den første måde vi forsøgte at gøre det på. fungerede ikke for os, men efter en lang snak og lidt hjælp fra gruppe 1. Blev vi enige om at hente user data for brugere som ikke er logget ind via API.

Den største udfordring ved vores follow funktion har været at få vores hjemmeside til at genindlæse den brugerprofil som man er inde på. Dette burde have været en meget simpel funktion, men grundet det data som skal indlæses for at finde frem til den rigtige brugerprofil har dette været en kæmpemæssig udfordring som vi ikke fik løst som vi gerne ville. I stedet for at siden bliver genindlæst på den brugerprofil man er inde på bliver man sendt tilbage dashboard siden hvor man har klikket på en bruger for at komme ind på en brugerprofil side.

Som en mulighed inde på siden, skal en bruger kunne vælge om de ville have mørk eller lys tema. Vores udfordring lå i at få denne handling afspejlet i vores database, hvilket vi brugte meget energi på. Vi endte dog med at køre det ud fra localStorage, sådan det ikke er gemt på serveren.

Konklusion

Vi har igennem hele projektforsløbet haft tæt kontakt med hinanden for at kunne holde hinanden opdateret og for at vide præcist hvor langt vi har procesmæssigt. Vi har researchet en del på forskellige middleware og andre SoMe hjemmesider for at skabe vores egen version og for at kunne skabe en strategi for at lave det bedst mulige produkt efter vores egen overbevisning. Grundet den givne situation med covid-19 som vi oplever for tiden, har vi desværre ikke haft stor fokus på brugerundersøgelser, men mere funktionalitet og kodning af hjemmesiden. Der er ikke tvivl om at covid-19 giver nogle komplikationer i forhold til gruppearbejde, og i vores tilfælde har der været både negative og positive oplevelser. Vores evne til at samarbejde og kommunikere er blevet styrket en hel del efter dette projektforsløb.

Referencer

<https://codingheroes.io/resources/#node>

<https://dev.to/ananyaneogi/create-a-dark-light-mode-switch-with-css-variables-34l8>

<https://discord.com/>

<https://mongoosejs.com/docs/validation.html>

<https://momentjs.com/>

<https://mailtrap.io/> - *Development Environment*

<https://pugjs.org/language/inheritance.html>

<https://pugjs.org/api/getting-started.html>

<https://www.postman.com/>

<https://stackoverflow.com/questions/42065872/regex-for-a-valid-hashtag/42551826>

<https://sendgrid.com/> - *Production Environment*

<https://www.twilio.com/docs/authy/tutorials/two-factor-authentication-node-express>

<https://www.sitepoint.com/a-beginners-guide-to-pug/>

<https://www.youtube.com/watch?v=76tKpVbjhu8>

https://www.w3schools.com/jsref/prop_win_localstorage.asp

https://en.wikipedia.org/wiki/Regular_expression

Bilag 1: package.json (Mads)

Dependencies	Description from NPMJS
"@babel/polyfill": "^7.8.7".	<p>Babel includes a polyfill that includes a custom regenerator runtime and core-js. This will emulate a full ES2015+ environment (no < Stage 4 proposals) and is intended to be used in an application rather than a library/tool. (this polyfill is automatically loaded when using babel-node).</p> <p>This means you can use new built-ins like Promise or WeakMap, static methods like Array.from or Object.assign, instance methods like Array.prototype.includes, and generator functions (provided you use the regenerator plugin). The polyfill adds to the global scope as well as native prototypes like String in order to do this.</p>
"axios": "^0.19.2".	<p>Axios does HTTP requests,</p> <ul style="list-style-type: none"> • Make XMLHttpRequests from the browser • Make http requests from node.js • Supports the Promise API • Intercept request and response • Transform request and response data • Cancel requests • Automatic transforms for JSON data • Client side support for protecting against XSRF
"bcryptjs": "^2.4.3".	<p>bcryptjs does bcrypt encryption for storing passwords,</p>

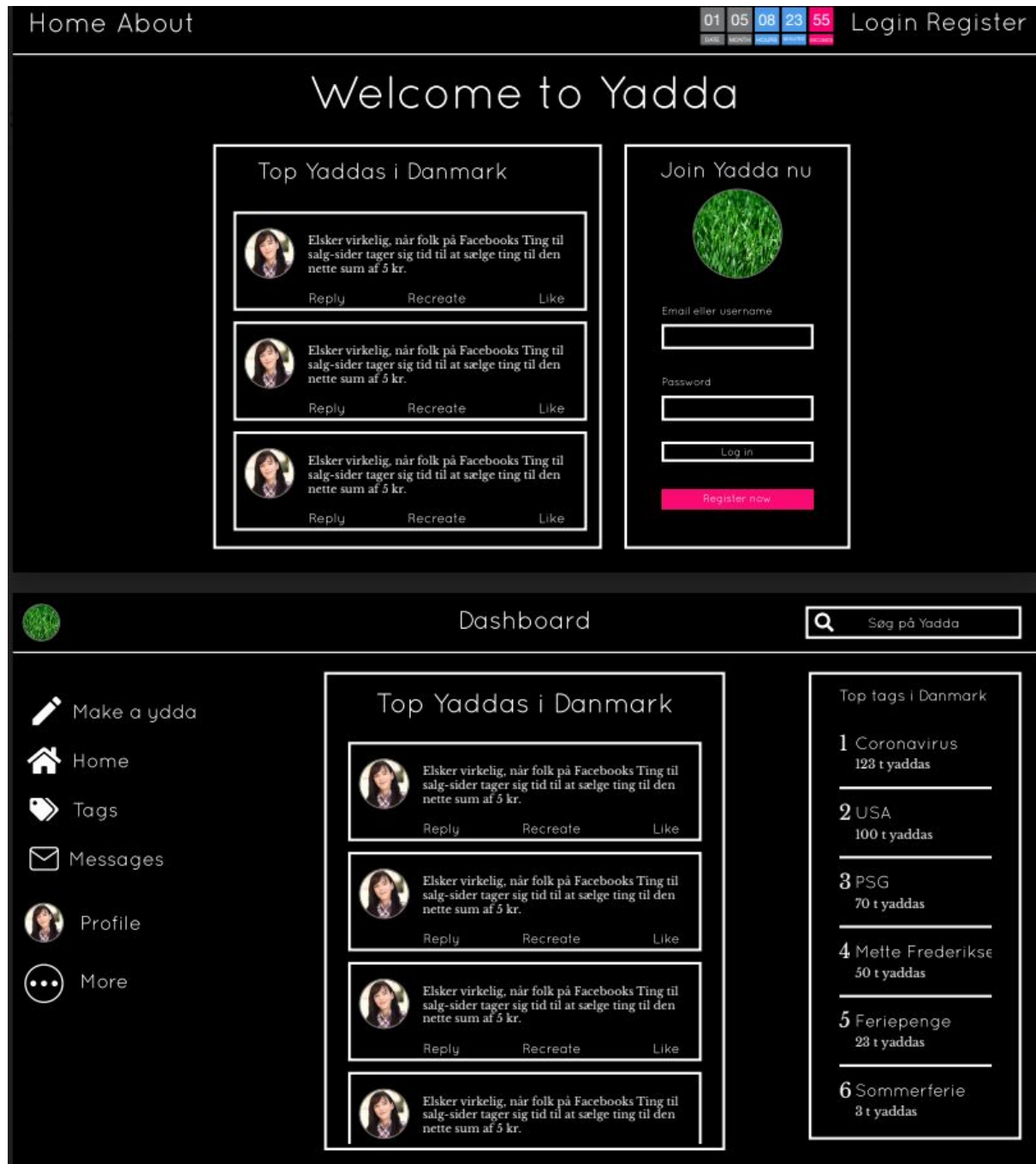
	Optimized bcrypt in JavaScript with zero dependencies. Compatible to the C++ bcrypt binding on node.js and also working in the browser.
"connect-flash": "^0.1.1",	The flash is a special area of the session used for storing messages. Messages are written to the flash and cleared after being displayed to the user. The flash is typically used in combination with redirects, ensuring that the message is available to the next page that is to be rendered.
"cookie-parser": "^1.4.5",	Parse Cookie header and populate req.cookies with an object keyed by the cookie names. Optionally you may enable signed cookie support by passing a secret string, which assigns req.secret so it may be used by other middleware.
"debug": "~2.6.9",	A tiny JavaScript debugging utility modelled after Node.js core's debugging technique. Works in Node.js and web browsers.
"dotenv": "^8.2.0",	Dotenv is a zero-dependency module that loads environment variables from a .env file into process.env . Storing configuration in the environment separate from code is based on The Twelve-Factor App methodology.
"express": "^4.16.4", "express-fileupload": "^1.1.7-alpha.3", "express-mongo-sanitize": "^2.0.0", "express-rate-limit": "^5.1.3", "express-session": "^1.17.1", "express-validator": "^6.4.1",	express is a web server, express-* are express middlewares,

<u>"helmet": "^3.22.0",</u>	Helmet helps you secure your Express apps by setting various HTTP headers. It's not a silver bullet, but it can help! (https://securityheaders.com/) Also modifies other headers,
<u>"formidable": "^1.2.2",</u>	A Node.js module for parsing form data, especially file uploads.
<u>"hpp": "^0.2.3",</u>	Express middleware to protect against HTTP Parameter Pollution attacks
<u>"html-to-text": "^5.1.1",</u>	An advanced converter that parses HTML and returns beautiful text. It was mainly designed to transform HTML E-Mail templates to a text representation. So it is currently optimized for table layouts.
<u>"http-errors": "~1.6.3",</u>	Create HTTP errors for Express, Koa, Connect, etc. with ease.
<u>"jsonwebtoken": "^8.5.1",</u>	JSON Web Tokens are an open, industry standard RFC 7519 method for representing claims securely between two parties. JWT.IO allows you to decode, verify and generate JWT.
<u>"moment": "^2.25.3",</u>	A lightweight JavaScript date library for parsing, validating, manipulating, and formatting dates.
<u>"mongoose": "^5.9.9",</u>	Mongoose is a MongoDB object modeling tool designed to work in an asynchronous environment. Mongoose supports both promises and callbacks.
<u>"morgan": "~1.9.1",</u>	HTTP request logger middleware for node.js

<u>"multer": "^1.4.2",</u>	Multer is a node.js middleware for handling multipart/form-data, which is primarily used for uploading files. It is written on top of <u>busboy</u> for maximum efficiency. NOTE: Multer will not process any form which is not multipart (multipart/form-data).
<u>"nodemailer": "^6.4.6",</u>	Send e-mails from Node.js
<u>"nodemon": "^2.0.3",</u>	Nodemon is a tool that helps develop node.js based applications by automatically restarting the node application when file changes in the directory are detected. nodemon does not require any additional changes to your code or method of development. nodemon is a replacement wrapper for node. To use nodemon, replace the word node on the command line when executing your script.
<u>"passport": "^0.4.1",</u> <u>"passport-local": "^1.0.0",</u>	Passport is <u>Express</u> -compatible authentication middleware for <u>Node.js</u> . Passport's sole purpose is to authenticate requests, which it does through an extensible set of plugins known as strategies. Passport does not mount routes or assume any particular database schema, which maximizes flexibility and allows application-level decisions to be made by the developer. The API is simple: you provide Passport a request to authenticate, and Passport provides hooks for controlling what occurs when authentication succeeds or fails.
<u>"pug": "^2.0.4",</u>	Full documentation is at <u>pugjs.org</u> Pug is a high performance template engine heavily influenced by <u>Haml</u> and implemented with JavaScript for <u>Node.js</u> and browsers. For bug reports, feature

	requests and questions, open an issue . For discussion join the chat room . You can test drive Pug online here .
"sharp": "^0.25.3" .	<p>The typical use case for this high speed Node.js module is to convert large images in common formats to smaller, web-friendly JPEG, PNG and WebP images of varying dimensions.</p> <p>Resizing an image is typically 4x-5x faster than using the quickest ImageMagick and GraphicsMagick settings due to its use of libvips.</p> <p>Colour spaces, embedded ICC profiles and alpha transparency channels are all handled correctly. Lanczos resampling ensures quality is not sacrificed for speed. As well as image resizing, operations such as rotation, extraction, compositing and gamma correction are available.</p> <p>Most modern macOS, Windows and Linux systems running Node.js v10+ do not require any additional install or runtime dependencies.</p>
"validator": "^13.0.0" .	A library of string validators and sanitizers.
"xss-clean": "^0.1.1"	Node.js Connect middleware to sanitize user input coming from POST body, GET queries, and url params. Works with Express , Restify , or any other Connect app.

Bilag 2: Wireframes



[Logo here](#) [Home](#) [Ur](#) [Register](#) [Login](#)

Trending Yaddas

1

What's on your mind (username)?

1 day · 10000 likes

2

What's on your mind (username)?

1 day · 10000 likes

3

What's on your mind (username)?

1 day · 10000 likes

© 2020 | YaddaYaddaYadda | Gruppe 2

[Logo here](#) [Home](#) [Ur](#) [Register](#) [Login](#)

Register User

Username


Email

First Name

Last Name

Password

Confirm Password

Avatar


[Register](#)

[Have an account? Login](#)

© 2020 | YaddaYaddaYadda | Gruppe 2

[Logo here](#) [Home](#) [Ur](#) [Register](#) [Login](#)

Login

Username

Password

Login

[No account? Register](#)

© 2020 | YaddaYaddaYadda | Gruppe 2


Logo here

Home

Ur

Register


Login



What's on your mind (username)?


☐
☐

Yadda




What's on your mind (username)?

☐
☐



What's on your mind (username)?

☐
☐



What's on your mind (username)?

☐
☐

© 2020 | YaddaYaddaYadda | Gruppe 2

[illegible]