

YabbaYabbaYabba

Anders Dall

Simon Lervad

Tinna María Richter

Rikke Domino Isaksen

Erhvervsakademi Kolding, IBA

Webudvikling, 2. semester

Niels Müller Larsen

Mile Ninkovic

Luise Steenholt

Indholdsfortegnelse

01	1. Indledning	4
02	2. Problemformulering	4
03	3. Metodeovervejelser	5
04	3.1 Agilt/plandrevet - Simon	5
	3.2 Git - Simon	7
	3.3 Wireframe - Simon.....	7
05	4. Research	8
06	4.1 Krav til indhold - Anders	8
	4.1.1 Twitter - Anders.....	8
	4.2 Krav til applikationen - Anders.....	10
	4.2.1 Autentificerings værktøj - Rikke	11
07	5. Analyse.....	12
08	5.1 Node.js - Tinna.....	12
	5.1.1 Express - Tinna.....	12
	5.1.2 Indhold - Anders	12
	5.1.3 Wireframe - Tinna	13
	5.1.4 Node.js moduler	13
	5.2 MongoDB - Tinna	14
	5.2.1 MongoDB Atlas - Simon	14
	5.2.2 Mongoose - Tinna	14
	5.2.3 Indhold - Anders	15
	5.3 Hashing - Tinna	16
	5.3.1 Passport - Rikke	16
09	6. Konstruktion.....	18
10		

11	6.1 Bruger.....	19
12	6.1.1 Registrering - Rikke.....	19
13	6.1.2 Autentificering - Rikke & Anders	23
14	6.1.3 Login - Rikke.....	26
15	6.1.4 Validators og Sanitizers - Anders	28
16	6.1.5 Profil - Rikke & Simon.....	29
17	6.1.6 Profilbillede - Simon.....	31
18	6.1.7 Dark Theme - Rikke.....	31
19	6.1.8 Følger - Rikke & Tinna.....	33
20	6.2 Social Media	36
	6.2.1 Opslag - Rikke & Simon.....	36
	6.2.2 Billedopslag - Rikke & Simon	39
	6.2.3 Tags - Rikke	41
	6.2.4 Kommentar - Simon	43
	6.3 Design	46
	6.3.1 Navigation og menuer - Simon	46
	6.3.2 Reklamer - Simon & Anders	46
	6.3.3 Post - Simon	48
	7. Evaluering af proces og produkt.....	49
	8. Konklusion.....	50
	9. Referencer	51

Vigtig information

Send emails

kodeord og mail til verificering af bruger:

user: "yabba3times@gmail.com"

pass: "@yabba1234"

Den bedste løsning er at lave en fil kaldet ".env", sætte den i mappen med følgende tekst:

EMAIL=yabba3times@gmail.com

PASSWORD=@yabba1234

Databasen

mongo "mongodb+srv://yabba-aampa.mongodb.net/test"
--username JohnDoe123

Adgangskoden: JohnDoe123

1. Indledning

Der skal laves et socialt medie, hvor man skal oprette sig som bruger og logge ind for at anvende siden. Når brugerne er logget ind, skal de have mulighed for at skrive opslag med tilhørende billede og hashtags. Derudover skal det være muligt at følge andre brugere og kommentere på deres opslag. For at siden kan gøres dynamisk skal alle informationerne gemmes i en database med tilhørende to kollektioner: "user" og "post". I disse kollektioner er der yderligere objekter og arrays, som informationerne bliver gemt i. Alt dette bliver udarbejdet i Node.js med frameworket Express sammen med MongoDB og modulet Mongoose.

2. Problemformulering

Hvordan man lave et socialt medie, vha. Node.JS, Express og Mongoose, hvor man kan oprette en bruger, hvorefter brugerne kan følge hinanden og kommunikere med korte beskeder?

3. Metodeovervejelser

3.1 Agilt/plandrevet - Simon

For at opretholde god struktur, holde styr på arbejdsopgaverne og fordele dem, er der blevet udarbejdet et Gantt kort, hvor man kan se, hvilke arbejdsopgaver der skal udføres, og hvornår der er blevet sat tid af til dem. For at sikre et godt samarbejde, og for at kunne hjælpe hinanden, står der også, hvilke dage gruppemedlemmerne, hver især, står til rådighed. Dette giver et overblik over projektforløbet og opgaverne dertil, der skal udføres, hvilket hjælper med at færdiggøre dem efter planen, så projektet kan blive færdigt til tiden.

	A	L	M	N	O	P	Q	R	S
1									
2	Opgaver	mandag 4/5	tirsdag 5/5	onsdag 6/5	torsdag 7/5	fredag 8/5	lørdag 9/5	søndag 10/5	mandag 11/5
3	Mødes								
4	Anders								
5	Simon								
6	Tinna								
7	Rikke								
8	Arbejder på kode								
9	Videomøde	kl. 8	kl. 9	kl. 9	kl. 8	kl. 9			kl. 9
10	Generelt								
11	Indledning								
12	Problemformulering								
13	Wireframe								
14	Metode								
15	Research								
16	Analyse								
17	Konstruktion								
18	Evaluering af proces og produkt								
19	Konklusion								
20	Referencer								
21									
22	InDesign								
23	Programmering								
24	Research								
25									
26									
27	Konstruktion								

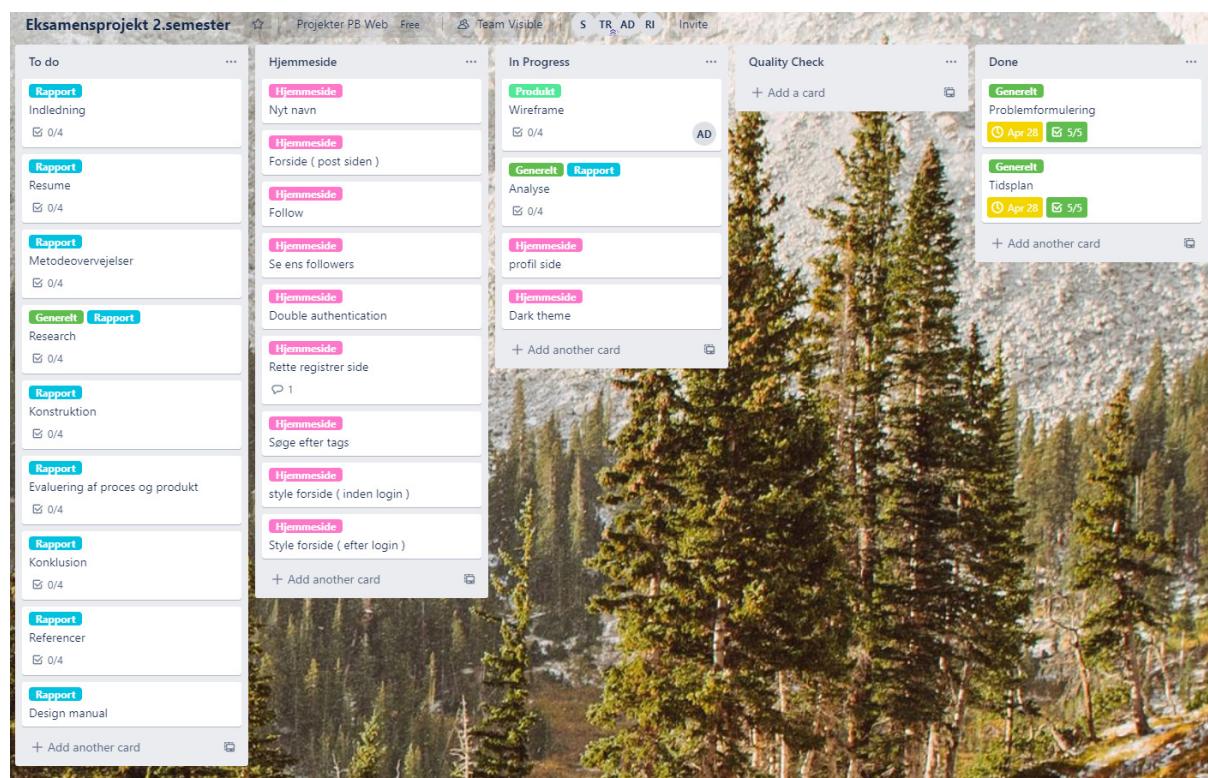
(Gantt)

Som hjælp til at holde styr på, hvornår de forskellige opgaver udføres, samt hvem der har udført dem, er værktøjet Trello blevet anvendt. Som billede viser, kan man se en liste over to do, som er opgaver, der mangler at blive udført.

Til højre, ser man endnu en bjælke, der indeholder opgaver, som skal udarbejdes på hjemmesiden.

Dette hjælper med at give overblik over arbejdsopgaverne, samt at få udført det hele efter planen. De tre sidste bjælker man ser på billedet er ‘in progress’, ‘quality check’ og ‘done’.

De repræsenterer hver især, det de er navngivet. Bjælken ‘in progress’ hjælper med at holde styr på, hvad der bliver arbejdet på lige nu. ‘Quality check’ er afsluttede opgaver, som mangler gruppens godkendelse før den endelig bliver rykket til ‘done’.



(Trello)

Med udgangspunkt i de to værktøjer beskrevet ovenfor, er der yderligere blevet fulgt op med god jævnlig kommunikation over MS Teams. SCRUM-møder har fundet sted ved hjælp af videoopkald, skærmdeling og god kontakt. Dette skal sikre et godt samarbejde og overblik over projektet.

3.2 Git - Simon

For at alle har adgang til projekts koder, er der blevet oprettet et online repo på Github, som alle har skrive/læse adgang til gennem egne profiler. Dette hjælper med at arbejde på produktet, samt have en online sikkerhedskopi af arbejdet. For at undgå merge konflikter, er der skiftevis blevet arbejdet på hjemmesiden af en person ad gangen. Der bliver under et SCRUM møde hver mandag aftalt, hvem der har hvilke dage i den pågældende uge. Den enkelte person har på den måde kunne holde sig til arbejdsopgaverne, som var lagt ind i Trello og danne sig et overblik over, hvilke opgaver der mangler at blive udført.

3.3 Wireframe - Simon

Før hjemmesiden kan laves skal der laves en wireframe af siden. En low fidelity wireframe hjælper med at give indblik i, hvordan sitet skal se ud, inden det bliver udarbejdet. På den måde er det nemt at holde samme struktur og opbygning gennem hele processen. Wireframen er tegnet i Adobe XD og er en simpel skitse over opstilling af sitet. Der er ingen farver eller endelig content, men models opstilling i forhold til placering af elementer.

4. Research

4.1 Krav til indhold - Anders

Projektet er at lave et social medie website, hvor der i projektbeskrivelsen er blevet stillet krav til, hvad der skal være af indhold, og hvilke funktioner der skal være. Der er også stillet krav til, hvordan applikationen skal bygges op.

- Brugerne skal kunne kommunikere med korte beskeder på maks. 167 tegn
- Brugerne skal kunne registrere sig
- Der skal være en autentificering af brugeren
- En bruger skal kunne følge andre brugere
- Der vises en tidslinje med de nyeste beskeder øverst
- Brugerne skal kunne vælge et “dark theme”
- Der skal være mulighed for reklamer på siden

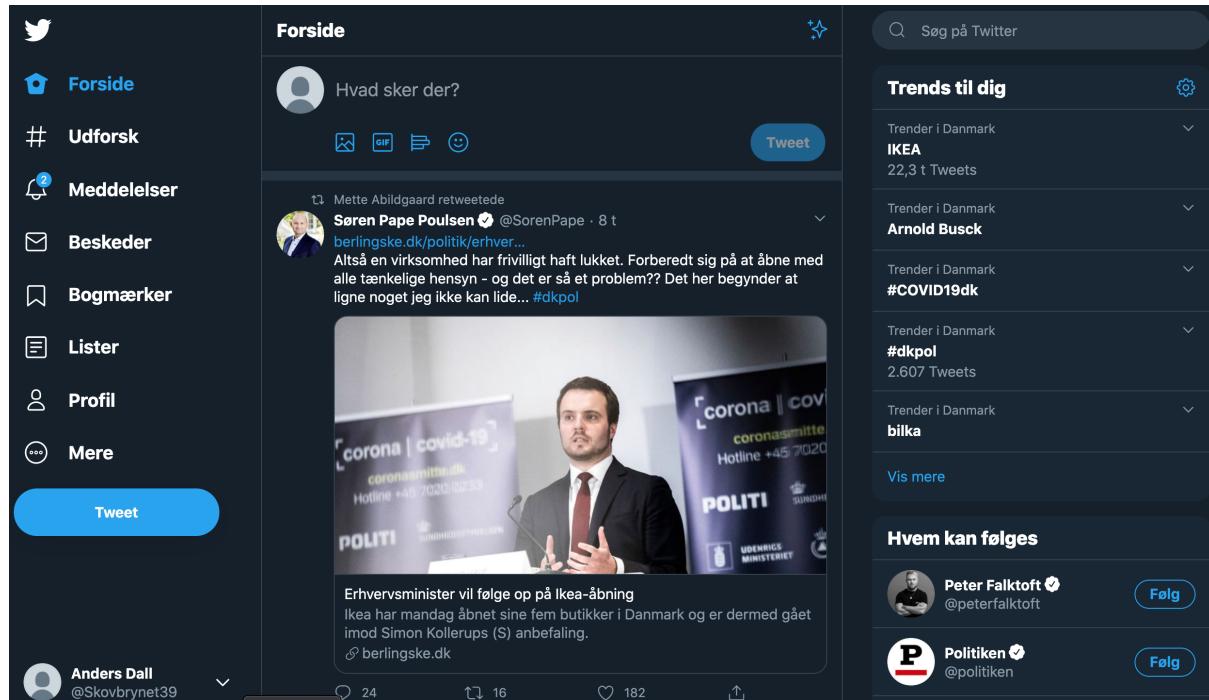
På baggrund af ovenstående kriterier er der blevet undersøgt, om der er nogen hjemmesider, der minder om produktet. Der er her blevet set på, hvilket indhold, der skal lægges vægt på samt placering af elementer på siden. Det er primært Twitter, der opfylder kravene.

4.1.1 Twitter - Anders

For at kunne bruge Twitter skal man være oprettet som bruger, herefter kan man følge personer eller virksomheder, og de kan følge dig.

Twitter er bygget op i tre vertikale sektioner. Sektionen yderst til venstre fungerer, som en menu med mulighed for at gå direkte til et nyt Tweet. Sektionen er sticky og kan ikke scrolles med.

I den midterste sektion kan man oprette et nyt Tweet og se Tweets fra alle de brugere, som man følger - sorteret efter dato/tid. Denne sektion kan scrolles, så man ser opslagene fra dem, man følger. Den sidste sektion til højre giver forslag til Tweets og forslag til nye sider, der kan følges.



(Twitter - Dark)

På Twitter kan man skrive en besked på maks. 280 tegn, og det er muligt at indsætte et billede, en gif, lave en afstemning og/eller indsætte emojis.

Andre brugere kan reagere på beskeden ved at like, kommentere eller videresende til f.eks. en mailadresse. Det er også muligt at retweete en besked fra en anden bruger og kommentere på den, så den når ud til ens egne følgere.

4.2 Krav til applikationen - Anders

I forbindelse med research delen har der været krav til brug af et Open Source Server miljø, web-interface, dokumentdatabase og håndtering af databasen. Der er undersøgt, hvad forskellige server miljøer gør.

Node.js er et Open Source Server miljø, der er bygget på JavaScript. Node.js er asynkron, så den kan håndtere flere opgaver på samme tid, uden at vente på en opgave er færdig, før den næste starter (W3School n.d.). I Node.js kan man hente forskellige web-interface designs. Express er et web-interface, der er designet til at lave webapplikationer og Application Programming Interfaces (API) (Express.js n.d.). Fordelen ved at installere Express til Node.js er, at applikationen ikke skal kodes fra bunden. Den valgte templating engine til Express, der er anvendt er Pug. En templating engine er den måde, som Express anvender de filer, der ligger i "views" folderen. Pug er valgt, da det er den template engine, gruppen har mest erfaring med. Derfor er det oplagt at anvende det i projektet. Desuden har den, som standard, en "layout.pug" fil, hvor head, header og footer kan placeres, så det fremstår ens på alle sider.

Express kan anvende API, som er en konstruktion, der tillader udviklere at gøre komplekse ting. Derfor bliver det også anvendt i projektet. For at få API til at virke, bliver der anvendt AJAX, som står for Asynchronous JavaScript And XML. Hvilket gør det muligt at læse data fra en webserver, efter siden er loaded. Derudover kan AJAX opdatere en side uden at reloade, da AJAX sender data til webserveren i baggrunden. Ved at anvende AJAX kan der bruges en kombination af browserens indbyggede XMLHttpRequest object, som beder om data fra en webserver, JavaScript og HTML DOM til at vise den pågældende data (AJAX Introduction n.d.).

Udover Express og AJAX bliver der i projektet anvendt MongoDB, som er dokumentdatabasen, hvor det er muligt at gemme data i et format, lignende JSON. Data gemmes under et valgt databasenavn i kollektioner. En database kan indeholde flere kollektioner (What Is MongoDB? n.d.).

I forbindelse med MongoDB er der valgt at anvende Mongoose, som er et Object Data Modeling (ODM) bibliotek for MongoDB. Mongoose bruger Schema, hvilket gør det muligt at give attributterne værdier, som Datatyperne: required og unik (Karnik n.d.).

4.2.1 Autentificerings værktøj - Rikke

I projektet skal man kunne oprette sig som bruger, man skal derfor overveje, hvordan brugeren skal godkendes på siden. At godkende en bruger gennem brugernavnet og adgangskoden er den mest anvendte måde, men det er god skik at anvende 2-faktor autentificering, da brugeren er mere sikker mod hacking. 2-faktor autentificering kan være et hemmeligt spørgsmål, et mobilnr., ID-kort, et ansigt, fingeraftryk eller baseret på lokationer. Man anvender oftest 2-faktor autentificering for at skabe mere sikkerhed, da mange bruger den samme adgangskode flere steder (Chipurici 2016).

5. Analyse

5.1 Node.js - Tinna

Node.js er et Open Source server miljø, der kan køres på adskillige platforme. For at få serveren til at virke, anvendes der JavaScript. Hvilket kan fungere til at lave webapplikationer uden en browser takket være Node.js. Node.js er asynkron, hvilket vil sige, at den kan udføre flere opgaver samtidig, uden at de blokerer hinanden (W3School n.d.).

5.1.1 Express - Tinna

Projektet anvender Express, som er et web framework i Node.js, der gør det nemmere at programmere en webapplikation. Det er et modul, der strukturerer applikationen. Fordelen er, at applikationen ikke skal bygges op fra bunden. Express installeres gennem pakken Express.js i Node.js. Den valgte template engine er Pug. Valget er faldet på den pga. gruppens erfaringer med den valgte template. Da Pug minder meget om HTML i syntaks og semantik, er den også meget ligetil at arbejde med. Forskellen er, at Pug er whitespace sensitiv, hvilket HTML ikke er.

5.1.2 Indhold - Anders

Projektet skal indeholde nogle sider, fx skal det være muligt at oprette sig som bruger, logge ind og skrive beskeder. Derfor er følgende sider nødvendige:

- forside, hvor det er muligt at logge ind eller oprette en ny bruger
- logge ind siden
- register siden
- en side, til når man er logget ind og kan skrive beskeder
- profilsiden

På alle sider, bortset fra forsiden, skal det være muligt at kunne se reklamer og have adgang til menuen.

5.1.3 Wireframe - Tinna

Når sidens indhold er planlagt, skal der udarbejdes en wireframe omkring hjemmesidens opbygning. Der er blevet taget inspiration fra diverse sociale medie applikationer, såsom Twitter, Instagram, Facebook og Jodel. Da de havde nogen af de samme elementer, som indholds kravene i projektet. Wireframen blev holdt i low fidelity, da der udelukkende skulle fokuseres på opsætning og funktionalitet og ikke på designet.

5.1.4 Node.js moduler

For at få flere funktioner i en applikation kan man hente forskellige moduler, som kan være en hjælp til at programmere applikationen.

Multer - Rikke

Når brugerne anvender siden, skal det være muligt for dem at indsætte et billede i forbindelse med et opslag. Man kan anvende et modul, som skal hjælpe med dette. I forbindelse med projektet er der anvendt modulet, Multer, som er en hjælp til at uploadere filer - i dette tilfælde billeder (Multer n.d.; Express Multer Middleware n.d.).

Express-validator - Rikke

Udover at anvende modulet, multer, vil modulet, express-validator også blive anvendt. Express-validator modulet er et Express.js middleware, som giver et bibliotek af valideringsværktøjer. Man anvender den for at sikre siden, da dataene fra brugerne ikke kan stoles på. Man ikke ved om brugeren vil prøve at skrive en funktion i et inputfelt eller textarea, som kan ændre på brugerdataen (Getting Started n.d.).

Nodemailer - Rikke

I forbindelse med projektet skal der sendes mails ud til brugeren, for at sikre at brugere ikke er spambots. For at sende mails ud er der anvendt Nodemailer, som er et modul til Node.js applikationen (Reinman n.d.).

5.2 MongoDB - Tinna

Projektets sider er dynamiske, hvilket bl.a. er pga. MongoDB. MongoDB bliver anvendt til at styre databasen. Det er et gratis online værktøj, der anvendes til at administrere databaser med Node.js. I dette projekt er der en database, "test", som indeholder to kollektioner, "user" og "post". Inde i kollektionerne er der forskellige objekter og arrays, der indeholder de informationer, der skal gemmes i databasen.

5.2.1 MongoDB Atlas - Simon

I projektet anvendes MongoDB Atlas, som er en platform, hvor man kan oprette MongoDB gratis med 500mb til rådighed. Databasen er oprettet som et "cluster", hvor Node.js applikationen kalder på dens server + cluster, som ejes. Det har hjulpet med at fuldføre oplevelse af et socialt medie, da man kommunikere til hinanden gennem platformen.

5.2.2 Mongoose - Tinna

I forbindelse med MongoDB bliver der også arbejdet med Mongoose, som har været med til at gøre siden dynamisk. Denne udvidelse arbejder med et format, ligende JSON, schemas, som hjælper med at oprette og vedligeholde kollektioner i databasen.

5.2.3 Indhold - Anders

For at kunne oprette/huske en bruger og gemme beskeder skal informationer lagres i MongoDB. I projektet bliver der anvendt 2 kollektioner i databasen. En kollektion, der gemmer brugerens data, og en anden kollektion med opslagernes data.

I kollektionen “user”, skal der være information om brugeren:

- et unikt brugernavn
- fornavn
- efternavn
- unikt e-mail
- password
- dato for oprettelse
- mulighed for at vælge avatar
- mulighed for at vælge dark theme
- se personer man følger
- se om brugeren er valideret til at logge ind
- en secret token, som brugeren får på mail ved oprettelse

I kollektionen “post”, skal der være informationer om opslagene:

- brugernavn, for at se hvem der har skrevet beskeden
- teksten, der er selve beskeden
- mulighed for at tilføje tags
- mulighed for at vedhæfte et billede
- dato for oprettelse af besked, så der kan sorteres på, hvornår beskeden er skrevet
- mulighed for at besvare en besked

5.3 Hashing - Tinna

Da man kan oprette sig som bruger på sitet, er det vigtigt, at deres adgangskode ikke bliver synlig i databasen. Derfor skal den hashed. Hashing er forvandlingen af en string, som ofte består af færre tegn eller en nøgle, som skal repræsentere den originale string. Hashing anvendes til at indeksere og hente ting fra en database, fordi det er hurtigere at finde dem med en kortere "hashed" nøgle end den originale værdi. Det er også anvendt i mange krypteringsalgoritmer (Rouse n.d.).

5.3.1 Passport - Rikke

Passport er en autentificerings værktøj til Node.js. Den er designet til et formål, som er at håndtere autentificerings anmodninger. Når man skriver moduler, delegerer Passport funktioner ud til applikationen, hvilket gør, at koden er ren og nemt kan vedligeholdes. Når man snakker om autentificering er der forskellige måder at gøre det på. Den mest almindelige er at give brugerne lov til at lave et brugernavn og en adgangskode, men brugeren kan også få muligheden for at logge ind vha. OAuth, som er login fx vha. Facebook, Twitter, Pinterest eller Google. I projektet skal brugeren godkendes før, de kan anvende siden.

Anvender man modulet Passport vil man erfare, at der er forskellige strategier - også kaldet pakker med individuelle moduler, som man kan anvende (Hanson n.d.).

I projektet bliver der anvendt LocalStrategy, som tillader brugeren at logge ind ved godkendelse af et brugernavn og adgangskode, men for at sikre brugeren bliver der også anvendt 2-faktor godkendelse (Hanson n.d.).

Bcrypt - Tinna

For at sikre brugerens sikkerhed er deres kodeord blevet hashed. Dette er gjort vha. modulet Bcrypt fra Passport. Bcrypt er et nyt familiemedlem med hashing funktioner. Den er designet til undgå at gætte brugernes adgangskode, fordi den anvender en langsom hash konstruktion. Bcrypt anvender en variation af Blowfish krypteringsalgortime (Timpson 2017; Hale 2010). I forbindelse med projektet bliver der anvendt salt, hvilket betyder at der kommer ekstra cifre på adgangskoden, hvorefter den bliver krypteret. Udover at anvende salt kan man vælge at anvende hashing, som er fixed cifre, der kan krypteres.

MD5 - Tinna

Udover at arbejde med Bcrypt kan man anvende algoritmen MD5, da det er den mest anvendte algoritme ved hashing af kodeord.

Et MD5 hash er lavet ved at tage en string, med hvilken som helst længde, og indkode det til et 128-bitfingerprint. Indkodningen af den samme string, når man anvender MD5 algoritmen kommer altid til at resultere i den samme 128-bit hash output. MD5 hashes er mest anvendt sammen med mindre strings, når man gemmer kodeord kreditkortnumre eller andet sensitivt data i en database, dog er dens kryptering hastighed hurtig (Tools n.d.). Det er ikke den mest optimale løsning at anvende MD5, fordi krypteringshastigheden er hurtig. Derfor vil en hacker hurtigt kunne komme igennem 1.000 adgangskoder på relativ kort tid.

6. Konstruktion

I projektet er der anvendt Express med fokus på templating engine, Pug. For at installere Pug skal man i sin terminal skrive:

```
express --view=pug
```

Ved at anvende Express får man et interface, som er designet til webapplikationer og API, derudover er mappestructuren lavet på forhånd. Pug er valgt, da det er brugervenligt, og man har mulighed for at lave en fil, som indeholder den samme struktur på alle filer. Derfor skal man kun ændre indholdet på de forskellige sider, men det som går igen, fx header er ens på alle.

```
2   html
3     head
4       title= title
5       meta(charset='utf-8')
6       meta(name="viewport" content="width=device-width, initial-scale=1.0")
7
8       link(rel="icon", type="image/svg+xml", href="/favicon.svg")
9       link(rel='stylesheet', href='/stylesheets/style.css')
10      link(rel='stylesheet', href='/stylesheets/darkTheme.css')
11
12      script(type='module' src='/javascripts/darkTheme.js')
13      script(type='module' src='/javascripts/main.js')
14      script(type='module' src='/javascripts/commercials.js')
15      script(type='module' src='/javascripts/characterCounter.js')
16      script(src='https://kit.fontawesome.com/9f34627289.js')
17
18    body
19      block content
```

(layout.pug)

6.1 Bruger

Når mappestrukturen er lavet, skal man i gang med at lave de forskellige sider. I forbindelse med projektet skal der oprettes en bruger, så de kan logge ind og anvende hjemmesiden.

6.1.1 Registrering - Rikke

Når en bruger registrerer sig, bliver brugerens data tilføjet til databasen. I den forbindelse er der lavet et Mongoose Schema, som gør, at indholdet i databasen altid er ens. Ved at gøre det på den måde, får man de samme informationer ind i databasen, og der er ikke nogen informationer, som pludselig bliver glemt undervejs. I dette tilfælde er det informationer omkring brugernes brugernavn, indstillinger, e-mail, navn, adgangskode, følgere og hvornår de er blevet oprettet. Der er nogle indstillinger, som er generelle for alle brugere. Fx starter alle brugere ud med et lyst tema, og som værende ikke godkendte brugere.

```

1  const mongoose = require('mongoose');
2
3  const UserSchema = new mongoose.Schema({
4      username: {
5          type: String,
6          required: true,
7          unique: true
8      },
9      darkTheme: {
10         type: Boolean,
11         default: false
12     },
13     approved: {
14         type: Boolean,
15         default: false
16     },
17     email: {
18         type: String,
19         required: true,
20         unique: true
21     },
22     password: {
23         type: String,
24         required: true
25     },
26     firstName: {
27         type: String,
28         required: true,
29     },
30     lastName: {
31         type: String,
32         required: true,
33     },
34     avatar: {
35         type: String,
36         default: "images/avatar.jpeg"
37     },
38     following: [],
39     created: {
40         type: Date,
41         default: Date.now
42     },
43     secretToken: {
44         type: String,
45         required: true,
46     }
47 });
48
49 const User = mongoose.model('User', UserSchema, 'user');
50

```

(Mongoose Schema - User)

Når skemaet er lavet, skal man indsætte det ind i databasen. Der er derfor lavet en side *register.pug*, hvor brugerne kan registrere sig.

```
1  extends layout
2
3  block content
4      div#header
5          div
6          div
7              h2= title
8          div
9              a(href="/users/register") Register
10             a(href="/users/login") Login
11         section#logReg
12             h1 Welcome to YabbaYabbaYabba
13             p The place where you can write to people
14             form(action='/users/register' method='post').shadow
15                 label Username
16                 input(type='text' name='username' required)
17                 label Firstname
18                 input(type='text' name='firstName' required)
19                 label Lastname
20                 input(type='text' name='lastName' required)
21                 label Email
22                 input(type='email' name='email' required)
23                 label Password
24                 input(type='password' name='password' required)
25                 label Repeat password
26                 input(type='password' name='password2' required)
27                 input(type='submit' value='Registrer')
28                 if errors != undefined && errors.length != 0
29                     each error in errors
30                         div(role='alert' id='a1') #{error.msg}
31                 if success_msg != ''
32                     div(role='alert' id='a2') #{success_msg}
33                 if error_msg != ''
34                     div(role='alert' id='a3') #{error_msg}
35                 if error != ''
36                     div(role='alert' id='a4') #{error}
37             p Have an account? 
```

(register.pug)

I registreringen skriver brugerne deres brugernavn, e-mail, navn og adgangskode, herefter kører funktionen i *authController.js* filen, som sikrer, at adgangskoderne er ens og indeholder min 6 tegn og at brugernavn og e-mail er unikke. Hvis der er en fejl, får brugeren en fejlmeldelse på siden. Bliver adgangskoden godkendt bliver den hashed, og alle informationerne bliver sendt ind i databasen - I databasen kan man ikke se brugerens adgangskode, men man kan se den hashed adgangskode.

```
exports.postRegister = async function (req, res) {
  const { username, firstName, lastName, email, password, password2 } = req.body;
  let errors = [];

  if (!username || !firstName || !lastName || !email || !password || !password2) {
    errors.push({ msg: 'Please enter all fields' });
  }

  if (password != password2) {
    errors.push({ msg: 'Passwords do not match' });
  }

  if (password.length < 6) {
    errors.push({ msg: 'Password must be at least 6 characters' });
  }

  if (errors.length > 0) {
    res.render('register', {
      errors,
      username,
      firstName,
      lastName,
      email,
      password,
      password2
    });
  } else {
    User.findOne({ username: username, email: email }).then(function (user) {
      if (user) {
        errors.push({ msg: 'Username or email already exists' });
        res.render('register', {
          errors,
          username,
          firstName,
          lastName,
          email,
          password,
          password2
        });
      }
    });
  }
}
```

(*authController.js*)

6.1.2 Autentificering - Rikke & Anders

Nu hvor brugeren har mulighed for at registrere sig, skal de også have adgang til at logge ind på siden. Først skal de verificere at den mail, de er oprettet med, er en de selv har adgang til.

Når brugeren opretter sig, bliver der genereret en tilfældig kode i *authController.js*, som bliver genereret af modulet randomstring.

```
    } else {
        //Generate secret token
        const secretToken = randomstring.generate();

        const newUser = new User({
            username,
            firstName,
            lastName,
            email,
            password,
            secretToken
        });
    }
```

(*authController.js*)

Koden får brugeren tilsendt i en mail. Herefter skal brugerne indtaste koden på en side for at blive verificeret og få lov til at logge ind. Det er ikke muligt for brugeren at logge ind uden at have verificeret sin konto.

I mailen er der et link til siden, hvor koden skal tastes for at gøre det nemmere for brugeren at komme i gang med at bruge siden. Derudover kommer den hemmelige kode også ind i brugerens informationer i databasen, så de kan blive approved. En bruger bliver, som default, oprettet med approved false i databasen. Brugerne kan ikke logge ind før approved bliver ændret til true.

```
{
    "_id" : ObjectId("5eba526ff3afebc53bd19609"),
    "darkTheme" : false,
    "approved" : false,
    "avatar" : "images/avatar.jpeg",
    "following" : [ ],
    "username" : "Anders10",
    "firstName" : "Anders",
    "lastName" : "Dall",
    "email" : "anders1603@hotmail10.com",
    "password" : "$2a$10$/692SisczzbU5X8UepZ0Q02qTQyYt92xIt0UMACr7WLf1PsXqPy
e6",
    "secretToken" : "hFyrLmGSu8OIsR1vFdtOgwCaVojdX6uV",
    "created" : ISODate("2020-05-12T07:38:23.613Z"),
    "__v" : 0
}
```

(Database - user)

Når secret token bliver indtastet, kører der et check i *authController.js*, der finder ud af, om koden eksisterer. Gør den ikke det, bliver brugeren redirected til siden, hvor de får mulighed for at taste en ny kode. Hvis koden passer, bliver brugeren opdateret i databasen, så approved er true, og secretToken ændres til en tom streng, da den ikke længere skal anvendes.

```
exports.postVerifyemail = async function (req, res, next) { //Find secretToken to verify email

    let secretToken = req.body.secretToken.trim();

    //find the user that matches secret token
    let users = await mon.retrieve(User, { 'secretToken': secretToken }, {});
    console.log(users[0]);
    if (!users[0]) {
        res.redirect('verify');
    }

    let chk = { _id: users[0]._id }
    let user = new User({
        darkTheme: users[0].darkTheme,
        approved: true,
        avatar: users[0].avatar,
        _id: users[0]._id,
        username: users[0].username,
        firstName: users[0].firstName,
        lastName: users[0].lastName,
        email: users[0].email,
        following: users[0].following,
        secretToken: ''
    });
    let cs = await mon.upsert(User, user, chk);
    console.log(users[0]);
    res.redirect('login');

};
```

(authController.js)

For at brugeren kan blive approved skal mailen sendes. Dette er gjort ved hjælp af modulet Nodemailer. Derudover er modulet ‘dotenv’ anvendt for at beskytte/gemme den mail og kodeord, der bliver sendt.

Der er blevet lavet et modul ‘sendEmail’, som er en funktion med parametererne: ‘email’ og ‘secretToken’, som bliver sendt fra bruger databasen i *authController.js*. I modulet bliver der oprettet forbindelse til en gmailadresse, som er oprettet til formålet om at sende mails til brugerne. Derefter bliver indholdet til mailen lavet, hvor der bliver skrevet, hvilken mail der skal sendes til og indholdet. Herefter bliver mailen sendt vha. en sendMail funktion, som er indbygget i Nodemailer.

```
let sendEmail = async function (email, secretToken) {
  let transporter = nodemailer.createTransport({
    service: 'gmail',
    auth: {
      user: process.env.EMAIL,
      pass: process.env.PASSWORD
    },
    tls: {
      rejectUnauthorized: false,
      secureProtocol: "TLSv1_method"
    }
  });

  let mailOptions = {
    from: 'Team Yabba',
    to: email,
    subject: "Hello ✓",
    text: `Hello! Thanks for registering!  
  

    You can soon begin to Yabba.  
  

    Verify Your email by typing this token: <br>
    <br>` + secretToken + `<br><br>
    Have a Yabba day!`,
    html: `Hello! <br><br>
    Thanks for registering! <br><br>
    You can soon begin to Yabba. <br><br>
    Verify Your email by typing this token: <br>
    <br>`+ secretToken + `</br><br>
    On the following page:
    <a href="https://localhost:3000/users/verifyemail">localhost:3000/users/verifyemail</a> <br><br>
    Have a Yabba day! `,
  };

  transporter.sendMail(mailOptions, function (err, data) {
    if (err) {
      console.log('Error had happend', err);
    } else {
      console.log('Email sent');
    }
  });
}

module.exports = sendEmail;
```

(*mailer.js*)

Efter godkendelsen kan brugerne logge ind med brugernavn og adgangskode. Der er her anvendt LocalStrategy, som undersøger om brugernavnet og adgangskoden hænger sammen.

6.1.3 Login - Rikke

Efter at brugerne har verificeret sin e-mail, skal de nu have mulighed for at logge ind på siden *login.pug*.

```
extends layout

block content
  div#header
    div
    div
      h2= title
    div
      a(href="/users/register") Register
      a(href="/users/login") Login
  section#logReg
    h1 Welcome to YabbaYabbaYabba
    p The place where you can write to people
    h3.warning= warning
    form(action='/users/login' method='post')
      label Email
      input(type='text' name='email' required)
      label Password
      input(type='password' name='password' required)
      input(type='submit' value='Login')
      if errors != undefined && errors.length != 0
        each error in errors
          div(role='alert' id='a1') #{error.msg}
      if success_msg != ''
        div(role='alert' id='a2') #{success_msg}
      if error_msg != ''
        div(role='alert' id='a3') #{error_msg}
      if error != ''
        div(role='alert' id='a4') #{error}
      p No account?&nbsp;
        a(href='/users/register') Register
  include footer.pug
```

(*login.pug*)

For at logge ind skal brugerne skrive deres adgangskode og e-mail. Hvis de ikke stemmer overens, vil der komme en advarselsbesked, som fortæller brugeren, at deres e-mail eller adgangskode er forkert. Adgangskoden og brugernavnet bliver tjekket i *authController.js* filen. Hvis informationerne er korrekte, vil brugerne komme ind på siden. Den første side de møder vil være dashboardet, som er fyldt med opslag.

```
exports.login = function (req, res) { // vis login siden
    res.render('login', {
        title: "YabbaYabbaYabba"
    });
};

exports.postLogin = async function (req, res, next) { //login
    //check if user is validated
    let data = await mon.retrieve(User, { email: req.body.email }, {});
    let approved = data[0].approved;
    console.log(approved);

    if (!approved) { // not approved

        res.render('login', {
            warning: 'Please verify Your email'
        });
    } else {
        passport.authenticate('local', {
            successRedirect: '/dashboard',
            failureRedirect: '/users/login',
            failureFlash: true
        })(req, res, next);
    }
};
```

(*authController.pug*)

6.1.4 Validators og Sanitizers - Anders

For at sikre at alle felter, i forbindelse med login og registrering, bliver overholdt er der installeret et modul til hjælp, express-validator. Der indeholder et bibliotek af måder, hvor man kan undersøge om den data, der bliver tastet, følger et bestemt format og om noget af indtastningen skal modificeres (Wexler 2019). Hvis brugeren kommer til at trykke på mellemrumstasten, efter at have tastet brugernavn, bliver det ignoreret. Skriver brugeren sin mailadresse med store og små bogstaver, bliver det ignoreret. Det sikrer også, at en bruger ikke kan tilføje HTML og JavaScript i applikationen.

```
router.get('/register', forwardAuthenticated, auth.register);
router.post('/register', [
  check('username')    // express-validator
    .trim()
    .isLength({ min: 3 })
    .escape()
    .withMessage('You need a username'),
  check('firstName')
    .trim()
    .isLength({ min: 2 })
    .escape()
    .withMessage('What is your first name?'),
  check('lastName')
    .trim()
    .isLength({ min: 2 })
    .escape()
    .withMessage('What is your last name?'),
  check('email')
    .trim()
    .isEmail()
    .normalizeEmail()
    .withMessage('you need a email')
], auth.postRegister);
```

(*users.js*)

6.1.5 Profil - Rikke & Simon

Når brugeren er logget ind, kommer de ind på deres profil. På brugernes profil er deres informationer hentet fra databasen.

```
section#content
  div#profileHead
    div#avatar
      img(src=avatar alt='Avatar')
      button(id="changeProfile" class="profilePicture") Change profilepicture
    div#profile
      div.follow
        h2 #{user.username}
      div.follow
        div
          p Posts
          p#postCount=postCount
        div
          button#showFollowers Followers
            div#myModalF.modal
              div.modal-content#modalContent
                span.close &times;
                br
                p#followersCount 0
        div
          button#showFollowing Following
            div#myModal.modal
              div.modal-content
                span.close &times;
                each users in user.following
                  div.usernameLink
                    form#userProfile(action="/userPage" method="post")
                      input(type="hidden" name="username" value=users)
                      input(type="submit" value=users)
                      br
                p#followingCount=numberOfFollowing
```

(user.pug)

Derfor vil navnet, email og brugernavnet ændre sig alt efter, hvem der er logget ind. Derudover vil brugeren få en oversigt over, hvem de følger. Informationerne, som er hentet fra databasen, bliver hentet gennem *indexController.js*, som fortæller, at siderne gerne må få adgang til brugernes data. Hvis man ikke har skrevet denne information i *indexController.js*, under den pågældende side, vil informationerne ikke blive vist.

```

exports.user = async function (req,res) { //the profil site
    let checkPost = {username: req.user.username}
    //undersøger ens posts
    let cp = await mon.retrieve(Post, checkPost, { sort: {created: -1}})
    let postCount = cp.length;

    //tjekker ens følgere
    let following = await mon.retrieve(User, checkPost, { sort: {created: -1}})
    let number = following[0].following;
    let numberOffFollowing = number.length;

    res.render('user', {
        title: "YabbaYabbaYabba",
        user: req.user,
        avatar: req.user.avatar,
        posts: cp,
        postCount: postCount,
        numberOffFollowing: numberOffFollowing
    });
};

```

(indexController.js)

Hvis man klikker på en brugers navn i applikationen, bliver man ført til deres profil. Hvis det er ens egen profil, kan man se alle indstillinger fra databasen, hvis det derimod ikke er ens egen profil, vil man kun blive præsenteret med avatar-billedet, brugernavn, følgere, navn og e-mail. Yderligere bliver der kun fremvist de opslag, som er postet af den brugerprofil, man er inde på. Brugeren har også mulighed for at følge brugerprofilerne, som de besøger. Hvilket gør, at brugerens opslag bliver vist på forsiden. Hvis man ikke følger nogen, vil alle post blive fremvist på forsiden.

6.1.6 Profilbillede - Simon

I forbindelse med profilen skal der være et profilbillede. Ved hjælp af samme modul, multer, som uploader billeder til opslag, kan man ændre sit profilbillede. Når brugeren “hover” over sit profilbillede, på sin profil, vil muligheden for at skifte profilbillede komme frem, og brugeren vil blive spurgt om at udfylde en form med et billede, som opdaterer brugerens information i databasen med det nye billede. Ulempen ved denne metode er, at billederne bliver gemt i et filesystem og eksisterer flere gange, hvis en bruger vælger samme profilbillede. Fordelen er, at man ikke har en fyldt database med billeder, nu hvor databasen “kun” har 500mb til rådighed.

6.1.7 Dark Theme - Rikke

På de forskellige sider kan brugeren ændre deres tema alt efter om, de ønsker et lyst eller mørkt tema (Bruun 2019). Dette sker vha. en togglefunktion i *darkTheme.js*, som går ind og læser, samt ændrer databasens indhold.

```
const showTheme = function (e) {
    let userTheme = JSON.parse(e.target.responseText);
    console.log(userTheme.darkTheme)
    let theme = userTheme.darkTheme; //true or false

    const toggleSwitch = document.querySelector('.theme-switch input[type="checkbox"]');
    const currentTheme = localStorage.getItem("theme") ? localStorage.getItem("theme") : null;

    if(theme){
        toggleSwitch.checked = true;
        document.documentElement.setAttribute("data-theme", "dark");
        localStorage.setItem("theme", "dark");
        //$("#checkbox").setAttribute("checked", "true");
    } else {
        toggleSwitch.checked = false;
        document.documentElement.setAttribute("data-theme", "light");
        localStorage.setItem("theme", "light");
        //$("#checkbox").setAttribute("checked", "false");
    }

    function changeTheme(e){
        if(e.target.checked){
            document.documentElement.setAttribute("data-theme", "dark");
            localStorage.setItem("theme", "dark");
            window.location.href = "changeTheme";
        } else {
            document.documentElement.setAttribute("data-theme", "light");
            localStorage.setItem("theme", "light");
            window.location.href = "changeTheme";
        }
    }
    toggleSwitch.addEventListener("change", changeTheme);
};


```

(*darkTheme.js*)

Når man checker togglefunktionen til og fra vil brugerens tema blive skiftet. Filen *indexController.js* sørger for, at databasen bliver opdateret, i det brugeren anvender togglefunktionen.

```
exports.darkTheme = function (req,res) { //Checks what theme the user has
    res.json(req.user);
};

exports.changeTheme = async function (req, res, next) { //change the theme
    //console.log(req.user);

    if(req.user.darkTheme){
        var change = false;
    }else{
        var change = true;
    }
    let users = await userHandler.upsertUser(req, change);
    res.redirect(req.get('referer'));
};
```

(*indexController.js*)

Sitets farvevalg vil blive skiftet alt efter om, det valgte tema er lyst eller mørkt. I *style.css* er der lavet variabler, som indeholder forskellige farvekoder, der vil blive ændret alt efter om, temaet er mørkt eller lyst. Ved at anvende variabler i CSS skal man kun skrive farvekoden et sted. De steder, hvor man skal anvende farven, skriver man variablen. Det gør det mere overskueligt for programmøren, som kun skal henvise til variablerne.

```
:root {
    --background-color: #ffd8c3;
    --color: #f2986d;
    --color2: #c1eae8;
    --font-color: #4b4b4b;
    --white: #f2f2f2;
    --black: #232323;
    --grey: #5f5f5f;
    --heading: #5e7376;
    --links: #de5e05;
    --hover: #de5e05;
    --gradient: linear-gradient(180deg, var(--color2), var(--background-color));
    --yabbaLink: var(--white);
    --input: var(--background-color);
}

[data-theme="dark"] {
    --background-color: #28293D;
    --color: #33333D;
    --color2: #1C1C28;
    --font-color: var(--white);
    --grey: #2e373b;
    --heading: #DEA5E9;
    --links: #DEA5E9;
    --hover: #AAEFF2;
    --gradient: linear-gradient(180deg, var(--color2), var(--background-color));
    --yabbaLink: var(--black);
    --input: var(--heading);
}
```

(*style.css*)

6.1.8 Følger - Rikke & Tinna

Når man anvender et socialt medie, er det sjovest, hvis man kan følge nogle og se deres opslag. Derfor skal det også være muligt at følge personer i forbindelse med produktet. Der er derfor lavet en followknap på de andre brugeres profiler. Når man klikker på den ændrer databasen sig, så følgere kommer ind på sitet.

I *indexController.js* bliver databasen ændret, så det er muligt at følge personer. Når en bruger går ind på en anden person, vil der blive loopt igennem brugerens følgere, for at tjekke om brugeren følger den person, man er inde på.

```
exports.newFollow = async function (req, res, next) { //ny follow

    let iAmFollowing = req.body.followID;
    let following = req.user.following
    console.log(following);
    //console.log(req.body.followID);

    for(var i = 0; i < following.length; i++){
        if(iAmFollowing === following[i]){ //Følger man personen
            console.log("Du følger brugeren");
            following.splice(i, 1);
            console.log(following);

            let chk = {_id: req.user._id}
            let user = new User({
                darkTheme: req.user.darkTheme,
                approved: req.user.approved,
                avatar: req.user.avatar,
                _id: req.user._id,
                username: req.user.username,
                firstName: req.user.firstName,
                lastName: req.user.lastName,
                email: req.user.email,
                following: following
            });
            let cs = await mon.upsert(User, user, chk);

            return res.redirect("/user");
        }
    }
}
```

(*indexController.js*)

Hvis man ikke følger personen, bliver det muligt at følge dem vha. en knap i *profile.pug*, som går ind og ændre databasen. Følger brugeren allerede den pågældende bruger vil knappen blive ændret, hvorefter det vil være muligt at unfollow den anden bruger.

```
div
  button#showFollowing Following
    div#myModal.modal
      div.modal-content
        span.close &times;
        each users in user.following
          div.usernameLink
            form#userProfile(action="/userPage" method="post")
              input(type="hidden" name="username" value=users)
              input(type="submit" value=users)
              br
  p#followingCount=number0fFollowing
```

(*profile.pug*)

For at det skal kunne være muligt at se ens følgere, er der ligesom ved “following” blevet lavet en modal boks, der kommer frem, når man trykker på “followers”, som indeholder en liste over ens følgere. Det er her muligt at trykke sig ind på hver enkelt profil.

```
div
  button#showFollowers Followers
    div#myModalF.modal
      div.modal-content#modalContent
        span.close &times;
        br
  p#followersCount 0
```

(*user.pug*)

I *main.js* filer bliver der vha. HTML DOM genereret en form med tilhørende værdier fra databasen. Hvor der bliver lavet elementer og tilføjet attributter til de givne elementer. For at modal boksen kan komme frem på siden, bliver der først undersøgt, hvad ens eget brugernavn er, hvorefter dette brugernavn, bliver sammenlignet med followers fra andre brugere. På den måde kan man få en oversigt, hvor mange gange ens eget navn står i databasens ‘user following’ i kollektionen “user”.

```

if ($("#followersCounts")) {
    let otherUser = $("otherUser").innerText;
    console.log(users);
    console.log(otherUser);
    for (var i = 0; i < users.length; i++) {
        for (var j = 0; j < users[i].following.length; j++) {
            if (otherUser === users[i].following[j]) {
                console.log(users[i].username + "følger dig");
                count = count + 1;
                $("followersCounts").innerHTML = count;

                let div = document.createElement("div");
                div.setAttribute("class", "usernameLink");
                let followerForm = document.createElement("form");
                followerForm.setAttribute("id", "followerForm");
                followerForm.setAttribute("action", "/userPage");
                followerForm.setAttribute("method", "post");
                let input = document.createElement("input");
                input.setAttribute("type", "hidden");
                input.setAttribute("name", "username");
                input.setAttribute("value", users[i].username);
                followerForm.appendChild(input);
                let input1 = document.createElement("input");
                input1.setAttribute("type", "submit");
                input1.setAttribute("value", users[i].username);
                followerForm.appendChild(input1);

                div.appendChild(followerForm);
                $("#modalContent").appendChild(div);
            }
        }
    }
}

```

(main.js)

På andre brugeres profiler bliver samme metode anvendt, hvor ownUser blot bliver erstattet med otherUser. Der går ind og henter brugernavnet inde fra *profile.pug* filen.

6.2 Social Media

Udover at arbejde med brugerne skal der være mulighed for, at brugerne kan lave opslag på siden.

6.2.1 Opslug - Rikke & Simon

Der er i forbindelse med opslagene lavet et Mongoose Skema, der - ligesom tidligere - skal hjælpe med at holde orden på informationerne og sikre, at informationerne står ens i databasen.

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

const PostSchema = new mongoose.Schema({
  username: {
    type: String,
    required: true,
  },
  picture: {
    type: String,
    default: "none"
  },
  text: {
    type: String,
    required: true,
  },
  tag: [],
  created: {
    type: Date,
    default: Date.now
  },
  replyTo: {
    type: String,
    default: "none"
  }
});

const Post = mongoose.model('Post', PostSchema, 'post');

module.exports = Post;
```

(Mongoose Schema - Post)

Når forsiden bliver indlæst vil der altid, blive sendt to objekter med ind i den virtuelle dom, som bliver indlæst og fremvist, som dynamisk data på siden. Den ene er selve brugerens oplysninger så siden ved, hvem brugeren er. Det andet er et objekt, som indeholder alle posts. I *dashboard.pug* er der et loop, som indlæser og fremviser alle dataene fra objektet. På den måde er der en dynamisk fremvisning af alle posts. Post bliver sorteret med

```
{ sort: { created: -1 } }
```

som gør, at det nyeste opslag altid vil blive vist øverst.

```
div#posts
  each post in posts
    // if post.replyTo === 'none'
    div(class='post post${post.username}' id='post${post._id}')
      div.row
        div.row
          div.profileImage
            img(class='avatar${post.username}')
          div
            if post.username != user.username
              div.usernameLink
                form(action="/userPage" method="post")
                  input(type="hidden" name="username" value='${post.username}')
                  input(type="submit" value='${post.username}')
            else
              div.usernameLink
                a(href="/user")= user.username
            div
              label(id='created${post._id}')= post.created
        ...
      ...
    ...
  ...

```

(*dashboard.pug*)

Dashboardet indeholder alle posts, som standard. Hvis brugeren følger minimum en person, vil det kun være brugerens følgeres post, som bliver fremvist på siden.

```
constgetPost = function (ev) {
  let req = Object.create(Ajax);
  req.init();
  req.getFile("/getPost", showPosts);
};

constgetUser = function (ev) {
  let req = Object.create(Ajax);
  req.init();
  req.getFile("/getUsers", showUsers);
};
```

(*main.js*)

I *main.js* er der et AJAX kald, som henter alle data omkring posts og users samt stiller dataen til rådighed, så den kan manipuleres i DOMen. Dataen bliver brugt til at bestemme om brugeren, som er logget ind, følger nogen, så alle andre opslag bliver fjernet. På den måde er det kun opslag fra brugernes følger, som vil blive fremvist på siden.

Visning af opslagene skal ændre sig alt efter, hvem man følger. Derfor er der i *main.js* lavet en if/else statement, hvor der bliver undersøgt, hvor mange man følger. Hvis man følger over 1, bliver man ført ind i den først if statement, hvor der er 3 loops, som tjekker, hvor mange følgere man har, deres posts, samt hvilke posts der har en replyTo eller ej. Her bliver posts sorteret efter, hvem man følger eller ej. Hvis der er en post eller et replyTo, som er lavet af ens følgere bliver de vist. Hvis man ikke følger nogle, vil den vise alle posts, der er fra alle bruger.

```
const showUser = function (e) {
    let user = JSON.parse(e.target.responseText);
    console.log(user);
    console.log(posts);
    if (user.following.length > 0) {
        let allPosts = document.getElementsByClassName("post");
        for (var i = 0; i < allPosts.length; i++) {
            allPosts[i].style.display = "none";
        }
        for (var i = 0; i < user.following.length; i++) {
            console.log(user.following[i]);
            let followedPosts = document.getElementsByClassName("post" + user.following[i]); // loops through posts from that user
            for (var j = 0; j < followedPosts.length; j++) { // loops through own posts
                followedPosts[j].style.display = "block"; // display all those posts
            }
            let ownPosts = document.getElementsByClassName("post" + user.username); // finds user's own posts
            for (var q = 0; q < ownPosts.length; q++) { // loops through own posts
                ownPosts[q].style.display = "block"; // displays own posts
            }
            for (var k = 0; k < posts.length; k++) {
                if (posts[k].replyTo != "none") { // if comment
                    if (posts[k].username === user.following[i]) { // if user of comment is the same as user
                        $("post" + posts[k].replyTo).style.display = "block"; // show original post
                    }
                    let originalPost = $("post" + posts[k].replyTo);
                    if (originalPost.style.display === "block") { // if post comment replies
                        $("post" + posts[k]._id).style.display = "block"; // show all comments
                    }
                }
            }
        }
    } else {
        console.log("show all posts");
    }
}
```

(*main.js*)

6.2.2 Billedopslag - Rikke & Simon

Når der bliver lavet et opslag på siden, er det vigtigt, at brugeren kan skrive tekst, lave hashtags og indsætte et billede. Men for at man kan indsætte et billede, skal den gemmes i MongoDB, så den ved at, der er et billede til opslaget. Den nemmeste måde at give brugerne lov til at uploadere billeder er ved at installere modulet multer, som gør det muligt at uploadere i en mappe og skrive filnavnet i MongoDB (Nodejs File Upload with Mongodbs - Create a Photo Gallery Using Multer n.d.; Express Multer Middleware n.d.). For at uploadere filer og gemme dem kan man lave en fil, *upload.js*, der indeholder stien til filerne, generering af filnavnet, samt begrænsninger for brugerens valg af fil. Der bliver sikret, at den uploadedede fil er et billede, så brugerne ikke har mulighed for at uploadere HTML, JS, exe eller andet. Derudover er der en begrænsning 2 megabytes, så billedstørrelsen ikke overbelaster serveren.

```
const multer = require('multer');
const path = require('path');

/** Storage Engine */
const storageEngine = multer.diskStorage({
  destination: './public/images/upload/',
  filename: function(req, file, cb){
    cb(null, new Date().getTime().toString()+'-'+file.fieldname+path.extname(file.originalname));
  }
});
//init
const upload = multer({
  storage: storageEngine,
  limits: { fileSize: 2097152 },
  fileFilter: function(req, file, callback){
    validateFile(file, callback);
  }
}).single('picture');

var validateFile = function(file, cb ){
  allowedFileTypes = /jpeg|jpg|png|gif/;
  const extension = allowedFileTypes.test(path.extname(file.originalname).toLowerCase());
  const mimeType = allowedFileTypes.test(file.mimetype);
  if(extension && mimeType){
    return cb(null, true);
  }else{
    cb("Invalid file type. Only JPEG, PNG and GIF file are allowed.");
  }
}
```

(*upload.js*)

Når filstien, -navnet og begrænsninger er lavet, laves der en funktion i *authController.js*, hvor man henviser til *upload.js* filen. På den måde bliver det muligt at gemme brugerens billede i serveren og fremvise den i et opslag vha. MongoDB.

```
exports.postPost = async function (req, res, next) {
    console.log(req.body);
    console.log(req.user);

    //Splitter tags fra hinanden
    let text = req.body.text;
    console.log(text);
    let tags = /(^|\B)(?!([0-9_]+\b)([a-zA-Z0-9_]{1,30})(\b|\r)/g;
    let tagSplit = text.toLowerCase().match(tags);
    console.log(tagSplit);

    let post = new Post({
        username: req.user.username,
        tag: tagSplit,
        text: req.body.text
    });
    let cs = await mon.create(Post, post);
    console.log(cs);
    res.redirect('/dashboard');
};
```

(*authController.js*)

Udfordringen med modulet “*multer*” er opslag uden billede. Den kunne ikke finde ud af at lave objektet, med text informationerne uden et billede. Problemet blev løst ved at lave en *onChange* funktion i *main.js*, som ændrer action URL for at post et opslag. Hvilket betyder, at når brugeren vælger at poste/vedhæfte et billede til deres opslag, ændrer formen sin action URL'en fra “/users/post” til “/users/postImage”. Hvor den uden billede ikke anvender modulet. Man bliver dog nødt til lave et nyt objekt, hvor default for billede er “none”.

```
//laver en ny form, hvis der skal være et billede i et opslag
function images(){
    $("input").addEventListener("change", function() {
        $("#idSubmit").setAttribute("action", "/users/postImage");
        $("#idSubmit").setAttribute("enctype", "multipart/form-data");
        var reader = new FileReader();
        reader.onload = function(){
            var output = $('#output');
            output.src = reader.result;
        };
        reader.readAsDataURL(event.target.files[0]);
    });
}
```

(*main.js*)

6.2.3 Tags - Rikke

Når brugerne laver opslag, skal de også have muligheden for at lave tags, som de senere kan søge efter på siden. Der er i databasens kollektion “post” lavet et punkt til de forskellige tags, så opslagene er nemmere at sortere efter. Men da tagsene bliver lavet i opslaget, skal opslaget deles efter hashtaggene og sættes ind i tags arrayet.

```
{
    "_id" : ObjectId("5eb005f9d6c596f8a7ec01c4"),
    "picture" : "none",
    "tag" : [
        "#darktheme",
        "#love"
    ],
    "replyTo" : "none",
    "username" : "trolld1997",
    "text" : "Jeg elsker det mørke tema #DarkTheme #love",
    "created" : ISODate("2020-05-04T12:09:29.849Z"),
    "__v" : 0
},
{
    "_id" : ObjectId("5eb006bd0f88bdf8d4a18b1f"),
    "picture" : "none",
    "tag" : [
        "#jegharcorona",
        "#jeghardetskidt"
    ],
    "replyTo" : "none",
    "username" : "trolld1997",
    "text" : "Jeg kommer ikke i morgen #JegHarCorona #JegHarDetSkidt https://youtu.be/LLkvzMxqt3A",
    "created" : ISODate("2020-05-04T12:12:45.898Z"),
    "__v" : 0
}
```

(*MongoDB - Post*)

For at gøre det, er der i *authController.js* anvendt Regular Expressions, som leder efter et bestemt tegn, #, samt tekst og tal efter for at indsætte den i arrayet (Regular Expressions n.d.). Teksten, som kommer ind i array, er sat til at være lowercase, fordi JavaScript er case sensitive. Hvis man ikke laver ordet om til lowercase, vil der komme en udfordring, når brugerne skal søge efter tags, da de skal søge efter præcis det ord, der bliver skrevet med og uden versaler.

```

exports.postPost = async function (req, res, next) {
    console.log(req.body);
    console.log(req.user);

    //Splitter tags fra hinanden
    let text = req.body.text;
    console.log(text);
    let tags = /(^\B)#(?![0-9_]+\b)([a-zA-Z0-9_]{1,30})(\b|\r)/g;
    let tagSplit = text.toLowerCase().match(tags);
    console.log(tagSplit);

    let post = new Post({
        username: req.user.username,
        tag: tagSplit,
        text: req.body.text
    });
    let cs = await mon.create(Post, post);
    console.log(cs);
    res.redirect('/dashboard');
};

```

(authController.js)

For at fjerne udfordringen, så brugeren kan søge efter tags, bliver søgeordene lavet om til lowercase i *indexController.js*. Derfor vil tagsene matche, og brugeren vil finde, hvad de søgte efter.

```

exports.getTags = async function (req,res) { //the tags site
//console.log(req.user);
let posts = await mon.retrieve(Post, {}, {sort: {created: -1}});
res.render('tags', {
    title: "YabbaYabbaYabba",
    user: req.user,
    posts: posts
});
};

exports.findTags = async function (req,res) { //Find the tags
//console.log(req.user);
console.log(req.body.tag);
let lowerCase = req.body.tag.toLowerCase();
let posts = await mon.retrieve(Post, {tag: lowerCase}, {sort: {created: -1}});
console.log(posts);
//console.log(posts);
res.render('tags', {
    title: "YabbaYabbaYabba",
    user: req.user,
    posts: posts
});
};

```

(indexController.js)

6.2.4 Kommentar - Simon

Udover at lave tags skal det også være muligt for brugeren, at indsætte en kommentar på de forskellige opslag. For at skelne opslagene med kommentarerne, er der i databasekollektionen post indsat punktet “replyTo”, som vil være ‘none’, medmindre, der bliver skrevet en kommentar til et opslag.

```
{  
    "_id" : ObjectId("5eb00986a9219ef9d0acd288"),  
    "picture" : "none",  
    "tag" : null,  
    "replyTo" : "5eb00008770af51b0a5d634a",  
    "username" : "trold1997",  
    "text" : "Hej Tinna 😊",  
    "created" : ISODate("2020-05-04T12:24:38.350Z"),  
    "updated" : ISODate("2020-05-04T12:24:38.350Z")  
}
```

MongoDB - Post

Når brugeren skriver en kommentar til et opslag, vil den få en værdi fra opslagets id nummer, som bliver sendt fra *dashboard.pug* til *authController.js*.

```
    img(src=' ${post.picture}', alt='test',  
div.postReplies  
    div.commentReplies  
        p(id=`comments${post._id}`) 0  
        label Comments  
    div(style="width:100%;")  
        div.comment  
            div.profileImage  
                img(class='avatar${post.username}')  
            form(action="users/postReply" method="POST")  
                textarea(type="text" name="text" placeholder="What is happening?" maxlength="167" required)  
                //input(type="text" name="tag" placeholder="Tags")  
                input(type="hidden" name="replyTo" value=`${post._id}`)  
                input(type="submit" value="Send")
```

(dashboard.pug)

I *authController.js* vil opslagets id-nummer blive sat ind i et punktet “replyTo”, så de 2 posts får en sammenkobling.

```

exports.postReply = async function (req, res, next) {
    console.log(req.body);
    //console.log(req.user);

    //Splitter tags fra hinanden
    let text = req.body.text;
    console.log(text);
    let tags = /(^\B)#(?![0-9_]+\b)([a-zA-Z0-9_]{1,30})(\b|\r)/g;
    let tagSplit = text.toLowerCase().match(tags);
    console.log(tagSplit);

    let post = new Post({
        username: req.user.username,
        tag: tagSplit,
        text: req.body.text,
        replyTo: req.body.replyTo
    });
    let cs = await mon.create(Post, post);
    console.log(cs);
    //res.redirect(req.get('referer'));
    res.redirect("/dashboard");
}

```

(authController.js)

Alle kommentar bliver fremvist på forsiden, som ligesom alle andre opslag. På den måde kan man bruge de samme funktioner på profilsiderne, som for tags. Hvis brugeren befinder sig på enten profilsiden eller tagssiden vil opslag, som er en “replyTo” et andet opslag få tilføjet en ekstra rubrik. Rubrikken vil indeholde “answer to: xxxx - post.text”, som linker til det oprindelige opslag. På den måde bliver brugeren orienteret om, at det er en kommentar, hvad der bliver kommenteret på, og hvem der har skrevet det. Hvis brugeren befinder sig på forsiden, *dashboard.pug*, vil opslaget, som indeholder en “replyTo” blive fjernet og tilføjet via append til det opslag, som det en kommentar til, hvilket kan ses i *main.js*.

```

const showUser = function (e) {
    let user = JSON.parse(e.target.responseText);
    console.log(user);
    console.log(posts);
    if (user.following.length > 0) {
        let allPosts = document.getElementsByClassName("post");
        for (var i = 0; i < allPosts.length; i++) {
            allPosts[i].style.display = "none";
        }
        for (var i = 0; i < user.following.length; i++) {
            console.log(user.following[i]);
            let followedPosts = document.getElementsByClassName("post" + user.following[i]); // loops through posts from that user
            for (var j = 0; j < followedPosts.length; j++) { // loops through posts from that user
                followedPosts[j].style.display = "block"; // display all those posts
            }
            let ownPosts = document.getElementsByClassName("post" + user.username); // finds user's own posts
            for (var q = 0; q < ownPosts.length; q++) { // loops through own posts
                ownPosts[q].style.display = "block"; // displays own posts
            }
            for (var k = 0; k < posts.length; k++) {
                if (posts[k].replyTo != "none") { // if comment
                    if (posts[k].username === user.following[i]) { // if user of comment is the same as user
                        $("post" + posts[k].replyTo).style.display = "block"; // show original post
                    }
                    let originalPost = $("post" + posts[k].replyTo)
                    if (originalPost.style.display === "block") { // if post comment replies to
                        $("post" + posts[k]._id).style.display = "block"; // show all comments to that post
                    }
                }
            }
        }
    } else {
        console.log("show all posts");
    }
}

```

(main.js)

6.3 Design

Udover funktionalitet er det også vigtigt, at brugeroplevelsen er god. Derfor er det vigtigt, at siden er brugervenlig.

6.3.1 Navigation og menuer - Simon

For at sikre brugervenligheden er der lavet tre fastlåste menuer på siden; en top bar, der fungerer som en header, og to sidebar, en i hver side. Topbaren er placeret fixed top i toppen af siden og indeholder dynamisk data baseret på, hvilken bruger der er logget ind, og hvad sidens titel er. Sidebaren til venstre indeholder sidens navigation og er fixed lige under top barens position. Den indeholder alle de nødvendige links, en bruger har behov for på siden. Den sidste sidebar er placeret i højre side under top barens placering og er også fixed. Dens indhold er beregnet til at indeholde reklamer for sitet. Da den også er fixed, vil det ikke give mulighed for særlig mange reklamer, da baren selvfølgelig kun fylder en side.

6.3.2 Reklamer - Simon & Anders

Der er ikke mulighed for særlig mange reklamer på siden. Derfor er der planlagt at bruge samme strategi som Facebook, hvor man evt. laver hver tredje opslag til en reklame eller tilføje reklamevideoer i midten af hver film, som bliver uploadet. Det er ikke blevet udarbejdet på siden og bliver holdt rent teoretisk, men er en god mulighed, hvis siden skal lanceres.

Da der ikke er udarbejdet mulighed for reklamer mellem posts, er der afsat plads til det, til højre på siden. Der er indsats nogle eksempler, som skifter ved load af siden, så det giver en fornemmelse af, hvordan det kunne se ud, hvis der blev indsats rigtige reklamer.

YabbaYabbaYabba

Anders Dall

Home

#Tags

Profile

Logout

Write Yabba

Tyranitar
Wed May 06 2020 19:40:19 GMT+0200 (Central European Summer Time)

Er så småt ved at være morgenfrisk, men nu er klokken så også næsten 20.00 :-)

0Comments

trold1997
Wed May 06 2020 14:07:21 GMT+0200 (Central European Summer Time)

Jeg har døbt ham Leif #sødHund

0Comments

COROLLA HYBRID 2.0

180 HK OG LANGT PÅ LITEREN.

LIGE NU NEDSAT 20.000 KR.

SE MERE HER

Get den Corolla Hybrid 2.0 model til 19.500,- længere tager haves. Den viste model er med ekstraudstyr og teknologi til 180HK. Brændstoforbrug v/til. kørsel 18,9-20,4 km/l, CO₂-udslip 110-120 g/km.

(eksempel på reklamer)

6.3.3 Post - Simon

Post vil blive fremvist på siden, som set herunder - selve opslaget er lidt fremhævet som en box med afrundede hjørner. Man kan se billede, navn, samt dato for hvornår opslaget er lavet, og hvem der har skrevet det. Brugerne bliver præsenteret med indholdet af opslaget og en label, som fremviser antallet af kommentarer. Nederst på et opslag vil brugeren have mulighed for kort at skrive en kommentar og se andre kommentarer, som andre brugere har skrevet. Da der bliver kørt en online database, kan det fungere, som en live chat til hinanden, som fremvist herunder. En kommentar indeholder billede, dato og navn for brugeren, som har skrevet det og efterfølgende indholdet af kommentaren.

The screenshot shows a social media interface with a light blue header and a white main area. A post by user 'tmrichter' is displayed, featuring a profile picture of a woman, the name 'tmrichter', and the date '2020-05-04'. The post content is 'Hej Rikke , kan du se min post #coronaTimes'. Below the post, it says '3 Comments'. The first comment is from 'trold1997' (profile picture of a beach scene) on '2020-05-06' with the text 'Nice! 😊'. The second comment is from 'Lervad' (profile picture of a person sitting) on '2020-05-06' with the text 'Kommentar virker :-)'. The third comment is from 'trold1997' again on '2020-05-05' with the text 'Hey Tinna 😊'. Each comment has a small trash can icon to its right.

(/dashboard)

7. Evaluering af proces og produkt

Ved projektets opstart blev der lavet et Gantt kort, for at danne overblik over projektets tidshorisont, og et Trello Board, som har hjulpet med at danne overblik over opgaverne. Ved at anvende Gantt og Trello på denne måde, har gruppens medlemmer fået et overblik over, hvem der havde kodnings dag, hvem man kunne kontakte, hvis der var brug for hjælp - i tilfælde af at medlemmerne har haft andre aftaler. Men det har også været en hjælp i den forstand, at der altid har været et overblik over opgaverne, så gruppens medlemmer altid har vidst, hvad de skulle lave.

For at lave et produkt, som opfylder projektbeskrivelsens krav og for at skabe et godt samarbejde, har det været vigtigt, at der har været kommunikation mellem gruppemedlemmerne. Ved andre omstændigheder ville gruppen have mødt hinanden på daglig basis. I stedet er der valgt at holde et dagligt videomøde om morgenen, for at opsummere gårsdagens løsninger og diskuterer udfordringer, som er løst i fællesskab. For at få den opdaterede kode er der regelmæssigt blevet pushet til repoet på GitHub, hvor alle har haft adgang til den nyeste version af koden.

8. Konklusion

For at lave et socialt medie, hvor man kan oprette sig som bruger, logge ind og skrive beskeder til andre brugere, skal der laves en MongoDB database, vha. Mongoose, med 2 kollektioner. En der indeholder information om brugeren og en til posts.

Hjemmesiden er lavet vha. Node.js, hvor der er anvendt Express og Pug, som templating engine.

Når man opretter sig som bruger, bliver der sendt en mail med en verificeringskode, som er lavet gennem en tilfældig generet kode. Verificeringskoden skal anvendes til at verificere brugeren. Når den er indtastet, kan man logge ind på siden, hvor brugerens adgangskode bliver hashet vha. Bcrypt, så den ikke kan læses af andre i databasen. Når brugeren er logget ind, kan de skrive beskeder på maks. 167 tegn, samt tilføje et billede eller hashtags. Der er sat en begrænsning på tegn vha. en JavaScript funktion, for at brugerne ikke har mulighed for at skrive længere opslag, og hashtagene er opdelt vha. regular expressions. I opslagene er det muligt at vedhæfte billeder, som er lavet vha. modulet multer. Når brugerne anvender siden for første gang, vil der være opslag fra alle brugere, men når brugerne begynder at følge hinanden, vil der kun blive vist opslag fra dem de følger og de opslag, som dem de følger har kommenteret på. På deres egen profil kan man ændre profilbilledet, hvilket er gjort muligt vha. multer. For at gøre siden mere brugervenlig, er det muligt at søge efter de opslag, som indeholder hashtags. Her går man ind på undersiden "Tags" for at søge efter et bestemt hashtag. Når man søger, er det vigtigt, at '#' er foran det ord, man søger efter, ellers kommer det ønskede opslag ikke frem.

9. Referencer

AJAX Introduction (n.d.) available from <https://www.w3schools.com/js/js_ajax_intro.asp> [28 April 2020]

Chipurici, C. (2016) 'Why You Should Start Using Two-Factor Authentication Now'. [20 January 2016] available from <<https://heimdalsecurity.com/blog/start-using-two-factor-authentication/>> [29 April 2020]

Express Multer Middleware (n.d.) available from <<http://expressjs.com/en/resources/middleware/multer.html>> [1 May 2020]

Express.js (n.d.) Express - Node.Js Web Application Framework [online] available from <<https://expressjs.com/>> [28 April 2020]

Getting Started (n.d.) available from <<https://express-validator.github.io/index.html>> [6 May 2020]

Hale, C. (2010) How To Safely Store A Password [online] available from <<https://codahale.com//how-to-safely-store-a-password/>> [6 May 2020]

Hanson, J. (n.d.) Passport.Js [online] available from <<http://www.passportjs.org/>> [29 April 2020a]

Hanson, J. (n.d.) Passport-Local [online] available from <<http://www.passportjs.org/packages/passport-local/>> [29 April 2020b]

Hanson, J. (n.d.) Passport-Auth0 [online] available from <<http://www.passportjs.org/packages/passport-auth0/>> [29 April 2020c]

Karnik, N. (n.d.) Introduction to Mongoose for MongoDB [online] available from <<https://www.freecodecamp.org/news/introduction-to-mongoose-for-mongodb-d2a7aa593c57/>> [28 April 2020]

Multer (n.d.) available from <<https://www.npmjs.com/package/multer>> [6 May 2020]

Nodejs File Upload with MongodB - Create a Photo Gallery Using Multer (n.d.) available from <<https://programmerblog.net/nodejs-file-upload-tutorial/>> [1 May 2020]

Reinman, A. (n.d.) Nodemailer [online] available from <<https://nodemailer.com/about/>> [7 May 2020]

Rouse, M. (n.d.) What Is Hashing? [online] available from <<https://searchsqlserver.techtarget.com/definition/hashing>> [28 April 2020]

Timpson, W. (2017) Node.Js Authentication with Passport [online] available from <<https://blog.cloudboost.io/node-js-authentication-with-passport-4a125f264cd4>> [6 May 2020]

Tools, D. (n.d.) MD5 Hash Generator [online] available from <<https://www.md5hashgenerator.com/>> [28 April 2020]

W3School (n.d.) Node.Js Introduction [online] available from <https://www.w3schools.com/nodejs/nodejs_intro.asp> [28 April 2020]

Wexler, Y. (2019) Get Programming With Node.Js. Shelter Island, NY: Manning Publications Co

What Is MongoDB? (n.d.) available from <<https://www.mongodb.com/what-is-mongodb>> [28 April 2020]