



Profile

Update Profile

Logout

**Make it with Creative Cloud.**

Break through with the world's best creative apps.

From \$999/mo.

[Join now](#)

Steffen Sørensen  
Steffen Sørensen  
Steffen Sørensen  
Steffen Sørensen

write something



Post

**Just BARE Chicken**

**CERTIFIED ORGANIC.**  
CERTIFIED GOODNESS.

**amazonfresh**  
SHOP NOW

Steffen Sørensen  
Steffen Sørensen  
Steffen Sørensen  
Steffen Sørensen

Lars

26 maj, 2021, 18:58

Her er en mega flot Blue Cake, smager bedre end den ser gyselig ud! #kage



Reply to this Yadda

Post

Steffen Sørensen  
Steffen Sørensen  
Steffen Sørensen  
Steffen Sørensen

Lars

26 maj, 2021, 18:56

Her er en flot dansk kage! #kage



<b>Opgavens indhold</b>	<b>Alle 3</b>
<b>Problemformulering</b>	<b>Alle 3</b>
<b>Metodeovervejelser</b>	<b>3</b>
Sikkerhed	Erik 3
<b>Research</b>	<b>Alle 4</b>
<b>Analyse</b>	<b>4</b>
Sikkerhed	Erik 4
node.js crypto.	Erik 5
bcrypt npm.	Erik 6
Smartere muligheder?	Erik 6
<b>Appens konstruktion</b>	<b>7</b>
Modules	Christoffer 7
signup - login/logout.	Christoffer + Erik 8
UpdateProfile, Coverbilled og Profilbilled	Christoffer + Nickolaj 9
UpdateProfile Includes Partials	Christoffer + Nickolaj 11
Left og Right sidebar til reklamer	Christoffer 12
Yaddas	Christoffer + Nickolaj 13
Følge funktion	Nickolaj 13
Tema funktionalitet	Nickolaj 14
<b>Procesevaluering</b>	<b>Alle 14</b>
<b>Konklusion</b>	<b>Alle 15</b>
<b>Referencer</b>	<b>15</b>
Emails sendt via NodeJS med SendGrid API.	Erik 15
<b>Afgrænsninger</b>	<b>Alle 16</b>
<b>Udbudringer til evt. lignende kommende projekt</b>	<b>16</b>
Erik's Idéer	16
Christoffer's Idéer	16
Nickolaj's Idéer	17
<b>Bilag</b>	<b>17</b>

## Opgavens indhold

I denne opgave har vi skabt vores egen udgave af en Yadda Social media website. Vores generelle tilgang har været idéen om hvordan man poster indhold på et socialt medie site som Facebook, Twitter og Instagram, og måden man kommenterer på et sådanne post. En tilføjelse er her at pointere: Hashtag funktion som beskrevet i opgavebeskrivelsen.

Oprettelse af bruger samt verifikation af data og input er en del af appen, dog uden at komprimere brugeroplevelsen. Disse eksisterer så i en database med hhv. collection posts samt users.

## Problemformulering

I hvilken forlængelse kan vi skabe en signup funktionalitet der verificerer samt krypterer brugerdata, til at skabe et sikkert socialt site til deling af offentligt data. Og hvorvidt kan vi skabe en sikker databaseforbindelse til behandling og lagring af data.

- Hvordan får vi valideret en bruger, efter de er blevet oprettet.
- Hvordan skaber vi funktionaliteten for brugere så de kan uploade og bruge billeder på sitet, i form af profilbillede og "yadda" besked/kommentar?
- Hvordan skaber vi "følge" funktionaliteten, sådanne at man kan tilføje hinanden og se hvad hver især lægger op?

## Metodeovervejelser

### Sikkerhed

Da sikkerhed er en meget udfordrende og tidskrævende aspekt af faget webudvikling, ville vi konkretisere vores brug af kryptering, frem for at udnytte bekvemmeligheden af passport. - Og dette har vi udelukkende gjort i et formål om at lære mere af opgaveforløbet.

Vi kender til passport gennem undervisningen og tidligere projekter, men vi har følt det uoverskueligt og nærmest banalt at skulle omtale passport som værende en enhed i sig selv, da passport indeholder mange funktionaliteter. Vi har forstået at passport indeholder mange strategier man kan gøre brug af hvor man inkorporerer hvilke hashing algoritmer, funktionaliteter og data der skal til for at autentificere en brugers adgang til sitet.

Vi ønskede dog en lidt mere lærerig tilgang, hvor vi selv implementerede hvad vi ville inkludere hvilket skulle give et bedre udbytte i at forstå hvad vi lavede - uden at opfinde den dybe tallerken. Vi ønsker dog i fremtiden at gøre brug af de forskellige passport frameworks<sup>1</sup> kombineret med vores egne strategier og express middleware. I denne omgang valgte vi at gøre brug af Node.js Crypto til at kryptere vores e-mailtoken der midlertidigt bruges til at verificere en bruger, samt Bcrypt til at kryptere password (selvom normal konvention ville forlange omvendt brug af disse. Mere herom i afsnittet analyse -> Sikkerhed)

## Research

For at få ny viden om sikkerhed har vi studeret informationerne tilgængelige på npmjs.com i forhold til vores Crypto<sup>2</sup> og Bcrypt<sup>3</sup> krypteringsmuligheder.

Yderligere har vi googlet os frem til yderligere informationer herom, og derved brugt bl.a google der resulterede i diverse stackoverflow samt wikipedia informationer, som har underbygget vores forståelse af kryptering til en vis grad.

Vi har også benyttet vores kendskab til at undersøge diverse guides på youtube til opsætning af vores projekt, samt hvilke funktionaliteter der kunne have interesse, før vi blev enige om hvilke vi ville benytte os af.

## Analyse

### Sikkerhed

Når vi taler sikkerhed er det vigtigt at inkorporere tidligere indlært viden, nemlig RegEx.

Regular expressions er backbone konstruktionen på sikkerhed, da RegEx hjælper os med at ekskludere tegn der ikke må forefindes i vores input.fields da disse ender i vores database. Her har vi i vores verification.js modul sikret os med en **test =**

---

<sup>1</sup> <http://www.passportjs.org/packages/>

<sup>2</sup> <https://www.npmjs.com/package/crypto-js>

<sup>3</sup> <https://www.npmjs.com/package/bcrypt>

**s.match** hvor **s** er value loopet fra **object.entries** der får sine data fra **req.fields**.

```
for (const [key, value] of Object.entries(req.fields)) {  
  if (key !== "email" && key !== "accessToken" && key !== "post") {  
    let test = value.match(/^[a-zA-Z0-9#!., ]/)
```

For yderligere at efterkomme valideringen af den data der kommer ind, har vi via mongoose.schema sat op hvordan info skal medtages i databasen, hvilket sikrer at det er korrekt terminologisk input.

node.js crypto.

```
User.findOne({$or:[{email : email}, {username : username}]}).exec((err,user)=>{  
  if (user == null) {  
    var userObj = {  
      "name": name,  
      "username": username,  
      "email": email,  
      "password": password,  
      "gender": gender,  
      "emailToken": crypto.randomBytes(64).toString("hex"),  
      "isverified": false,  
    }  
  }  
})
```

Til vores verificering af email, har vi valgt at indføre en midlertidig emailToken som har formålet at være "unik" i en kort periode, da denne omdannes til null når brugeren følger linket de får på email når de opretter deres profil. Dette link tilsendes brugeren asynkront via SendGrid mail og indtager emailToken hex strengen og sammenligner den med databasen.

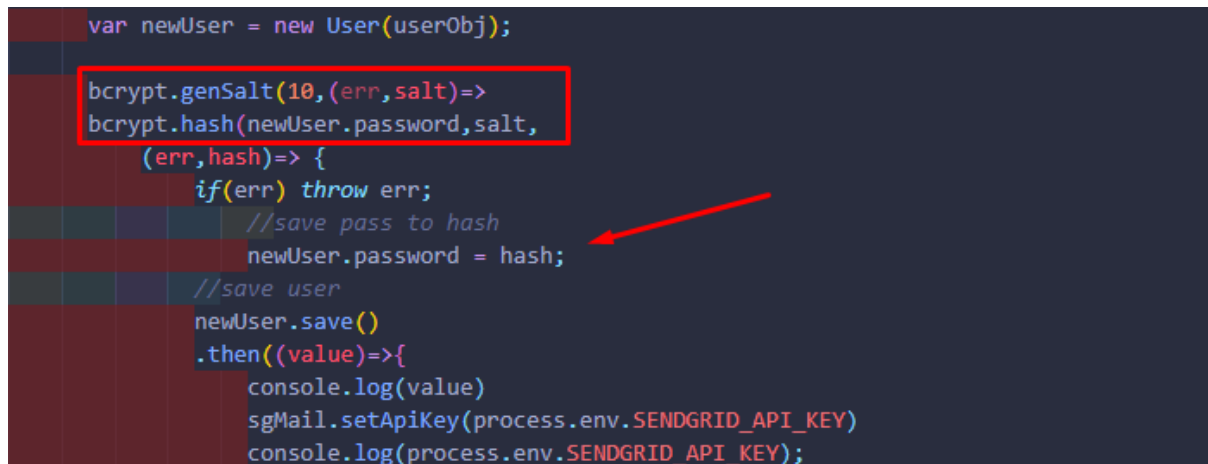
Grunden til at vi laver en *toString("hex")* på vores kryptering, er simpelthen fordi base64 encoding ikke er url-sikkert og derfor ikke kan bruges i vores email verifikation sådanne som den er sat op.

Men stemmer de to strings overens skulle en funktion have ændret denne *emailToken* til null, og lavet *isVerified* om til true så brugeren kan logge ind.

Ved login tjekker loginsiden så igennem *User.findOne* vha email input om *bcrypt.compare(password, user.password, (err, isMatch) =>* er passwordet korrekt udføres så *if ( user.isVerified == true)* defineres en *var acessToken = jwt.sign* hvor denne acessToken så bliver opdateret i systemet ved hjælp af *user.findOneAndUpdate* hvilket giver brugeren en ny acessToken som bruges til at

verificere data på sitet i fremtiden. Brugeren skal altså være “logget ind” med den accessToken for at kunne poste.

bcrypt npm.



```
var newUser = new User(userObj);

bcrypt.genSalt(10, (err, salt) =>
  bcrypt.hash(newUser.password, salt,
    (err, hash) => {
      if(err) throw err;
      //save pass to hash
      newUser.password = hash;
      //save user
      newUser.save()
        .then((value) => {
          console.log(value)
          sgMail.setApiKey(process.env.SENDGRID_API_KEY)
          console.log(process.env.SENDGRID_API_KEY);
        })
    })
  )
```

For yderligere at efterkomme sikkerhed, har vi naturligvis valgt at gøre brug af mere end én krypteringsform. På den måde øger vi sikkerheden af sitet, i det at selvom den ene krypteringsform eksponeres, så vil den anden stadig være gyldig og ikke røbe yderligere data fra databasen.

### Smartere muligheder?

Vi kunne have gjort brug af OAuth, som bl.a. kan gøre brug af 3rd party authentication. Vi kunne altså have indført at en bruger er/bliver verificeret igennem Facebook eller Google, og derigennem får adgang til vores side. På den måde gemmer vi mindst mulig data omkring brugerens login, og gør dermed brugen på vores site væsentligt mere sikker for begge parter.

En fordel er her også, at hvis vores side eller brugeren på en måde kan være komprimeret, at brugeren faktisk sendes til OAuth server, frem for vores app/site.

OAuth(passport)<sup>4</sup> som vi har stiftet bekendtskab med er express-kompatibel og derfor ville det have givet god harmoni at bruge, da vi allerede gjorde brug af node.express funktionaliteter. En ekstra positiv instans med OAuth er også at

---

<sup>4</sup> <https://www.npmjs.com/package/passport>

udviklerne selv ikke arbejder med passwords fra brugere, og der derved internt sitet/virksomheden/udvikleren ikke kan gemme og dermed eksponere denne.

Dog er der et lille “catch” i dette scenarie. For hvis en bruger så mister fx. sin mobil hvor mobiltelefonen automatisk er logget ind i messenger fra Facebook, er denne jo til hver en tid verificeret og kan dermed logge ind på vores Yasite(Yadda side). Så det ville være en ren og skær bekvemmeligheds situation hvor man opgiver sikkerhed direkte på sitet og giver denne videre til tredjepart virksomheden. (Vi er enige om at man kunne efterkomme denne situation med en 2-part autentifikation, men det ødelægger idéen med bekvemmeligheden og er dermed under vores forståelse en ugyldig løsning).

Sikkerheds Disclaimer: reel konvention ville forlange at man bruger den mere sikre krypteringsmulighed, nemlig Node.js Crypto, på passwordet frem for emailToken. grundet at Crypto benytter sig af SHA256 samt AES som er mere omtalt, anerkendt og gør brug af 128bit block size kryptering hvor Bcrypt som er en udviklet form af Blowfish baseret på en 64 block size kryptering.

## Appens konstruktion

### Modules

Med dette projekt var ideén at vi tog inspiration fra de forskellige sociale medier som allerede eksisterer. Vi tog primært vores inspiration fra facebook med hensyn til profil opsætningen samt twitters newsfeed funktionalitet og design. Et eksempel herpå er den måde man kommenterer på diverse Yadda på. Vi fulgte en online tutorial på youtube der gennemgår skabelsen af et socialt medie site med Node JS og MongoDB<sup>5</sup>, skabt af youtube creator Adnan Afzal. Vi brugte ikke kun guiden til at have et “framework” at starte ud fra, men også for at kunne komme med vores egne twists and turns på hvad vi synes skulle være anderledes. Vi var ikke stor fan af den måde han havde sat filerne op på, så der var en del ændringer der skulle foretages,

---

5

<https://www.youtube.com/watch?v=Co5a3QbSonU&list=PLsY8aWop1tAHigCqvzZ61XK3a8l509VRx>

bl.a ændrede vi vores opsætning til at bruge mongoose til schema. Men med hensyn til hovedfunktioner så var det en stor hjælp at have guiden.

Løbende fik vi installeret de modules vi havde brug for. Express har vi brugt for at kunne starte projektet. Hertil har vi brugt express formidable for at kunne håndtere input felterne, mongodb som er brugt til vores database, http som starter serveren. Bcrypt er blevet brugt som vores main krypteringsmodul til kryptering af passwords i databasen. Til at gemme filer har vi benyttet os af fs (filesystem) i vores projekt, samt brugt Json-webtoken som vores user-token. Cookies erstattes i dette scenarie med brugen af webtokens for at kunne authenticate brugeren og deres data via accesstokens. EJS er brugt som brugertilgangen til vores design såsom HTML og CSS. Socket.io skulle være brugt til real-time kommunikation. Et eksempel var hvis vi havde haft mere tid til projektet, kunne vi have tænkt os at muliggøre en chatfunktion mellem brugere og der ville socket.io så være til hjælp da den er med til at skabe en real-time kommunikation uden at skulle refreshe websiden. Nodemon er brugt for udviklingsmæssige årsager, frem for alt den indbyggede funktion at server ikke altid skal genstartes hvis der foretages ændringer i koden.

### **signup - login/logout.**

Vi startede med at oprettet vores signup/login system. Starten inkluderede filerne, login.ejs og signup.ejs. Udover de 2 filer tilføjede vi en public-mappe<sup>6</sup> som var hentet fra youtube creatoren Adnan Afzals videoguide, som sidenhen over flere omgange er blevet bearbejdet og ændret med eget design samt lightmode og darkmode og header samt footer. Efter vi havde tilføjet vores css i både vores signup og login. Så tilføjede vi en form som indeholder de ting en signup form skal indeholde. Informationer som name, username, email, password bliver gemt i vores database, vi gør brug af bcrypt for kryptering af vores password som bliver hashet under vores login funktionalitet inden informationerne valideres i schema og videresendes til databasen. I vores tilfælde har vi valgt ikke at have profilbilled med i signup processen. Derimod får brugeren lov at indrette sin profil under Update Profile siden,

---

<sup>6</sup>

[https://github.com/adnanafzal565/social-network-resources?fbclid=IwAR13wy35pMK8zY7ybZqQTPgDRct\\_aSbJ3PCDJA\\_xnjT3IOPIKo4bqrFYclM](https://github.com/adnanafzal565/social-network-resources?fbclid=IwAR13wy35pMK8zY7ybZqQTPgDRct_aSbJ3PCDJA_xnjT3IOPIKo4bqrFYclM)



hvor informationer som profilbillede, coverbillede, navn fødselsdato og yderligere info kan indsættes eller opdateres til enhver tid.

Dernæst dannede vi vores route til login hvor har en funktion som requester fields, dernæst tjekker den databasen efter e-mailen for at se hvis det er en registreret bruger, hvis e-mailen ikke er registreret så giver den en error message og siger emailen ikke eksisterer. Hvis emailen findes så går den videre til en else hvor den sammenligner passwordet med det vi har liggende i mongodb databasen og den sender så en parameter som hedder isVerify og hvis passwordet så stemmer overens så vil den så generere en accesstoken til brugeren via vores jwt module (json webtoken). Og det den gør er at den går så ind og opdaterer brugeren via findOneAndUpdate og gi'r den en accesstoken og så går den videre til at sige at brugeren successfully er blevet logget ind med sin nye accesstoken og redirecter dig til vores page som hedder /updateProfile. Viser det sig at passwordet er forkert så giver den dig bare en error og siger password is incorrect please try again. Og så har vi vores logout funktion som så redirecter dig til /login efter du successfully er blevet logget ud.

## **UpdateProfile, Coverbilled og Profilbilled**

UpdateProfile er en af vores hovedsider hvor vores fokus var at prøve at oprette noget i stil med facebook. Man skulle kunne oprette et profilbilled, coverbillede, og en "about me" sektion. Vi fik taklet den her opgave ved at starte med oprettelsen af UpdateProfile.ejs og en route. Da det var færdigt skulle nogle div'er oprettes som indeholder class's til vores css og i de div'er skulle vi have en form til hver. skulle have input felter som er type="file" og accept="image/" da jeg kun vil have den skal kunne indeholde billeder og hvert indeholder deres egne funktion "uploadCoverPhoto" og "uploadImage" Efter vi har dannet de diver og inputs så skal vi videre ned og oprette et <script> tag hvor vi får tilføjet noget javascript.

For at fremvise vores nuværende profil skal kriteriet: isUpdateProfile = true; Efterfølgende skal [showProfileData](#) funktionen udføres der via brugerens accessToken henter de rigtige billeder/strings fra databasen som vi vil have vist frem, hvilket er Coverbilled, profilbilled, name, email, dob, city, country og about me. De vil

altid være tomme når en ny brugere bliver oprettet eller logger ind for første gang, men efterfølgende vil de være til at gemme den nye data som brugeren indsender. Efter det er blevet dannet så har vi oprettet en javascript funktion for at kunne oprette vores coverbilleder og profilbilled, og vi skabte så et ajax objekt og den satte vi til (POST) som så er routen i dette tilfælde og når dataen så bliver modtaget og accepteret så vil den fremvise billedet i vores profil eller cover tag, og den bliver så parset tilbage til json string objekt. Derefter skabte vi en formdata hvor den opretter en access token sammen med localStorage så den ved hvilken bruger på databasen der netop har de billeder uploadet. Det er igen det samme for både profil og coverbilled.

Dernæst skal vi have fat i vores route som ligger i profile.js. Begge routes ligner hinanden, derfor dokumenterer vi cover photo i dette tilfælde.

I requesten `router.post('/upload/coverPhoto', async (req, res) => {` modtager vi accesstoken af brugeren som sender requesten / coverbilledet

Og så skal vi have tjekket hvis brugeren overhovedet eksisterer og det gør vi ved at tjekke accesstoken, og hvis accesstoken er udløbet så sender den et null objekt og giver en error.

```
} else {  
  if (req.files.coverPhoto.size > 0 && req.files.coverPhoto.type.includes("image")) {  
    if (user.coverPhoto != "") {  
      filesystem.unlink(user.coverPhoto, function(error) {  
        });  
    }  
    coverPhoto = "public/images/" + new Date().getTime() + "-" + req.files.coverPhoto.name;
```

message, hvor den siger at brugeren er logget ud og log venligst ind igen. Derefter tjekker vi hvis filen som bliver uploadet rent faktisk er et image eller ej. Og så tjekker vi hvis brugeren allerede har et coverbilled, og hvis de godkendes går den ind og unlinker det billede og erstatter det med det nye billede og opdaterer det i databasen via `User.findOneAndUpdate` via brugerens accesstoken. Er det første gang du uploader et billed gemmer den det i en mappe som vi har kaldt "public/images" Det er der hvor både vores profilbilleder, coverbilleder og post / kommentar billeder bliver gemt og når du opdatere et billed så ligger det gamle billed stadig i mappen.

Til allersidst så går den igennem for at se hvis brugeren er logget ind, er accesstoken udløbet så opdaterer den ikke billedet og i stedet sender den en fejlmeddelelse, er brugeren derimod logget ind så giver vi beskeden cover photo has been updated. Der er lige en lille del til sidst som skulle have været en delete funktion til når man bare vil slette et billede i stedet for at erstatte det, men det var som sagt nogle problemer med tidspres så den glemte vi alt om men det er som sagt noget man vil kunne tilføje senere.

## **UpdateProfile Includes Partial**

På alle sider der vises har vi `<%- include ("includes/header") %>` og `<%- include ("includes/footer") %>` og de to partials har vi primært på alle sider så vi kan fremvise den menu som er brug for. For at uddybe denne menu, så viser den forskellige ting alt efter om man er logget ind med accessToken eller ej. Er man verificeret vises Profile, Update og Logout. Er man ikke verificeret vises lin og logout. Udover det har vi efter vores script tag en section tag og inde i den har vi en masse div tags som har sine egne class's til css en af de divtags indeholder vores `<%- include ("includes/left-sidebar") %>` og længere nede sætter vi også vores right sidebar ind og det er for at kunne have vores reklame på blandt andet også den her side. Vi laver bagefter en form onSubmit som returner vores doUpdateProfile funktion og i den har vi en form som indeholder Full name, email, date of birth, city, country og about me og til sidst har vi en knap som vi kalder save hvor du submitter det du har skrevet og det bliver så gemt i vores database.

I routen har vi en router.get med hvor vi henter vores updateprofile og efter det har vi vores router.post hvor vi tilføjer de nye fields vi skal have tilføjet ind i database. Når vi så skal have opdateret det så laver vi i vores user.findOneAndUpdate via brugerens accessToken som går ind og opdatere de felter som vi skrev i updateprofile. Kan den ikke aflæse en accessToken så laver den en error hvor den siger at brugeren er blevet logget ud og skal logge ind igen. Hvis den kan aflæse brugerens accessToken så bliver der sendt en response hvor den siger at profilen er blevet opdateret.

Alt efter hvilken bruger der logger ind vil de modtage deres specifikke data via den her getuser. Vi konvertere bagefter response tilbage til Json, hvis der er success så vil den fremvise ShowProfileData, vores funktion til at fremvise vores header menu. Og til sidst så laver vi en formdata hvor vi appender en accesstoken kan den ikke hente accesstoken så fjerner den accesstoken og kalder vores showMainMenu.

```
function showMainMenu() {
    var html = "";

    if (localStorage.getItem("accessToken")) {
        html += '<li>';
        html += '<a class="navLink" href="/profile/u/${user.username}">Profile</a>';
        html += '</li>';

        html += '<li>';
        html += '<a class="navLink" href="/profile/update">Update profile</a>';
        html += '</li>';

        html += '<li>';
        html += '<a class="navLink" href="/logout" onclick="return doLogout();">Logout</a>';
        html += '</li>';
    }
}
```

Vi laver derefter vores showMainMenu funktion og i den har vi vores localStorage.getItem("accessToken") som ser hvis brugeren er logget ind, og er de logget ind fremviser den følgende menu, som har userprofile, updateprofile og logout.

Er brugeren ikke logget ind eller at siden ikke kan se en accesstoken så laver vi et else statement hvor vi fremviser en anden menu som indeholder en Login og Signup og den bliver fremvist på forsiden. Og selvfølgelig så har vi til allersidst vores DoLogout funktion som fjerner localStorage.accessToken og returner true.

## Left og Right sidebar til reklamer

```
<aside class="sidebar static">
  <div class="widget">
    
  </div>
</aside>
```

I vores left og right sidebar skabt til reklamer, har vi oprettet en aside med en class som gør at de blive sat over i hver deres side, og inde i den har vi lavet en div og inde i div'en har vi så et img tag hvor vi får sat vores billed ind med en width og en

height som passer ind med det vi har lavet, i dette tilfælde har vi valgt at bruge skyscraper ad typen da den passer perfekt ind i både højre og venstre side på vores hjemmeside.

## **Yaddas**

Brugergeneret indhold er vigtigt for et socialt medie, og på vores side har vi opslag som kaldes Yaddas. Yaddas kan indeholde en besked, et billede og en video. På forsiden har vi en tidslinje som viser opslagene.

Når brugeren loader siden, bliver der sendt en request til serveren om at modtage, opslagene for tidslinjen. Over tidslinjen har vi et felt hvor brugeren kan oprette et opslag. Serveren gemmer så billedet/videoen i en mappe, og de resterende data bliver gemt i databasen.

Vi havde nogle problemer i starten fordi vi gemte også linket til brugeren avatar i post dataen. Så når en bruger opdaterede deres avatar, ville den nye ikke blive vist på opslaget. Det problem løste helt simpelt ved at når tidslinjen bliver genereret, finder vi profilbilledet i vores user collection.

Når brugeren opretter et nyt opslag, har de mulighed for at tilføje et hashtag som fungerer som et link. Når man klikker på linket, vil brugeren blive vist en tidslinje hvor kun Yaddas med det samme hashtag bliver brugt.

## **Følge funktion**

Vi skulle have en funktionalitet der muliggjorde det for brugerne at kunne følge samt blive fulgt af andre profiler. Udover det når en bruger vælger at følge en anden, vil kun de fulgte opslag (Yaddas) blive vist på tidslinjen. Hvis brugeren ikke følger nogen, vil alle opslag i databasen blive vist på tidslinjen.

Alle brugere har en personlig profil, hvor man kan se diverse information om dem, f.eks. navn, fødselsdag by, land og en personlig beskrivelse. Hver profil har en egen tidslinje hvor kun profilen egne Yaddas vil blive vist.

Når en bruger kommer ind på profilen, bliver der sendt to request til serveren. Den ene er opslagene i tidslinjen, og den anden er en fetch post request om at få

oplysninger om hvem der følger brugeren. Den sender så samtidigt en accesstoken med. Serveren matcher om den bruger der er logget ind, er profilens ejer.

Hvis brugeren er ejeren af profilen, vil "follow" knappen forsvinde. Det er for at undgå at en bruger kan komme til at følge sig selv. Vi cheker dog ikke på serversiden om brugeren prøver at følge sig selv, da vi ikke nåede at implementere det.

En anden ting vi ikke nåede, var at give mulighed for at se dem der følger en profil, samt hvem profilen følger. Havde vi nået det, ville det nok være inspireret af den måde twitter gør det på.

### **Tema funktionalitet**

Vi har lavet en funktion der giver mulighed for brugeren at skifte mellem to temaer, lys og mørk. Det er selvfølgelig med for at fremme brugeroplevelsen, da en meget lys skærm kan være ubehageligt i et mørkt lokale. Der er nogle brugere der har personlige præferencer til et lyst eller mørkt tema.

Vi startede med at lave det så, at den tilføjede stylingen til hvert element f.eks. `body.style.backgroundColor`. Det virkede ikke da vi generer noget at siden efter klienten får svar fra serveren, så JS kunne ikke finde elementerne. Derfor var den logiske løsning at den skifter mellem at bruge to forskellige stylesheets afhængig af hvilket tema er valgt. Det giver så også den fordel at det er nemmere at tilføje og ændre i stylingen for hvert tema.

Brugerens valg bliver ikke gemt i databasen men derimod i localStorage i brugeren egen browser. Vi valgte ikke at gemme det i DB fordi, en bruger vil muligvis have forskellige temaer på forskellige enheder.

### **Procesevaluering**

Processen havde en møjsommelig begyndelse da vores morale efter 2.5 corona projekt, da dette projekt kom efter vores 2.5 Corona projektet. Motivationen er ikke særlig høj for tiden - grundet corona mm - så vi valgte faktisk i første uge at holde en hel uges pause fra kodning, hvilket man set i bakspejlet måske har været en dårlig idé, måske ikke.

Processen har dog været præget meget af stille individuelle timer hvor man løbende har opdateret hinanden over discord med hvad der i løbet af dagen/dagene var opnået. Når vi så havde opnået nogle standpunkter, satte vi koderne sammen i github og arbejdede sammen over Visual Studios online funktion: LiveShare så vi kunne sidde sammen i det, og se om der var kode konflikt så vi kunne udarbejde problematikker sammen.

## **Konklusion**

Vi kan konkludere at vores projekt ikke er slået helt fejl, selvom vi ikke har nået alle delmål. Vi havde formålet med projektet at skulle have en stor vidensudbytte i forhold til webudvikling. At lære om kryptering og indkorporere funktionalitet til at udsende en "ægte" email fra et mailsystem så vi i fremtiden ved hvordan man opretter email-funktionalitet til fx. salgssider samt service sider hvor automatiseret data kan deles gennem en sådanne funktionalitet.

Vi har nogle afgrænsninger vi er kede af at have oplevet, men man lærer ikke af at afgrænse, så vi må udarbejde disse afgrænsninger efter projektets afslutning.

## Referencer

### Emails sendt via NodeJS med SendGrid API.

Vi har benyttet følgende Youtube guide af Ian Schoonover. Videoen blev lavet af Ian i samarbejde med en elev ved navn Daryl, til et web-developer bootcamp projekt. kredibiliteten af webudvikler bootcampen går til Colt Steele som har overværet projektet.<sup>7</sup>

[https://www.youtube.com/watch?v=O14qnlCsleQ&ab\\_channel=IanSchoonover](https://www.youtube.com/watch?v=O14qnlCsleQ&ab_channel=IanSchoonover)

Sammen med deres part 2 video om verifikation af e-mail ved signup, har vi gjort brug af disse to guides, til at tilføje de ønskede elementer vi skulle bruge til vores eget projekt.

[https://www.youtube.com/watch?v=Zyc9pZrFoWE&ab\\_channel=IanSchoonover](https://www.youtube.com/watch?v=Zyc9pZrFoWE&ab_channel=IanSchoonover)

Disse elementer bestod af SendGrid API opsætning samt brugen af nodemailer og dertilhørende underelementer.

1. <http://www.passportjs.org/packages/>
2. <https://www.npmjs.com/package/crypto-js>
3. <https://www.npmjs.com/package/bcrypt>
4. <https://www.npmjs.com/package/passport>
5. <https://www.youtube.com/watch?v=Co5a3QbSonU&list=PLsY8aWop1tAHigCqvzZ61XK3a8l509VRx>
6. [https://github.com/adnanafzal565/social-network-resources?fbclid=IwAR13wy35pMK8zY7ybZqQTPgDRct\\_aSbJ3PCDJA\\_xnjT3lOPIKo4bqrFYclM](https://github.com/adnanafzal565/social-network-resources?fbclid=IwAR13wy35pMK8zY7ybZqQTPgDRct_aSbJ3PCDJA_xnjT3lOPIKo4bqrFYclM)
7. <https://generalassemb.ly/instructors/colt-steele/3055>

## Afgrænsninger

En afgrænsning er at vi har glemt at ændre så billeder kunne uploades som files frem for som en string til vores database (ups). Dette gør at vores website ikke er funktionelt online, hvilket er grunden til vi i denne omgang slet ikke har uploaded projektet online på Heroku.

Tekst og billede i en Yadda har vi glemt/ikke nået at gøre sådan at efter man trykker "post" bliver tekst samt billede fjernet. Man bør få en ny blank slate at arbejde på.

---

<sup>7</sup> <https://generalassemb.ly/instructors/colt-steele/3055>



For at imødekomme denne problematik havde vi dog allerede indført en brugervenlig funktionalitet der aktiverer en prompt som giver beskeden "Post has been uploaded."

Sitet er ikke særlig responsivt, og det er en fejl fra vores side at vi simpelthen har glemt vores multimediedesigner 101: Mobile first. - Vi skulle have skabt appen med responsiv design rettet mod mobiler fra starten, og tilføjet css og yderligere muligheder i tilfældet af en browser/desktop experience.

Zotero kunne vi ikke få til at virke korrekt, så vores links til diverse under referencer er ikke opsat korrekt som de skulle have været.

## **Udbedringer til evt. lignende kommende projekt**

### **Erik's Idéer**

Der kunne være en ekstra sikkerhed på sitet, der gør det endnu mere sikkert at have en privat underside som ingen andre end brugeren selv kunne se. Så man kunne have en form for file-management system hvor man kunne gemme billeder, evt. til senere brug på YaSite appen.

### **Christoffer's Idéer**

En sikker Private chat funktion hvor man kan chatte med folk man har været inde og fulgt / tilføjet som ven

### **Nickolaj's Idéer**

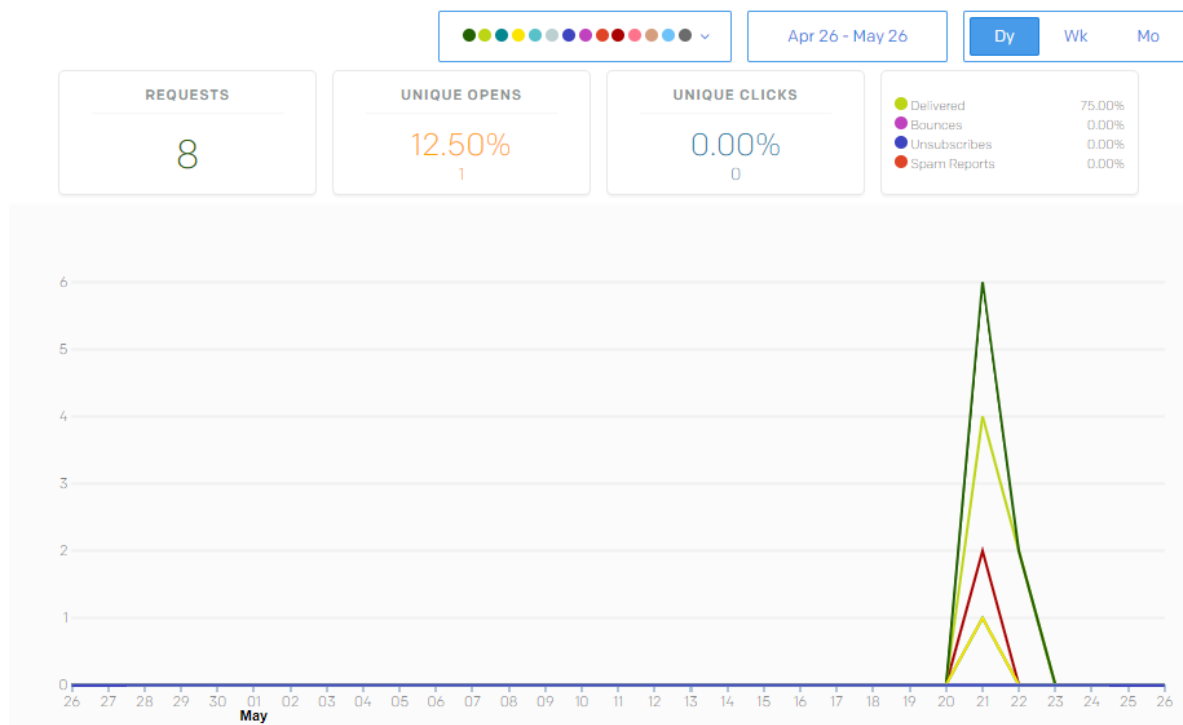
Det kunne være sjovt at have et spil som man kan spille imod hinanden i appen med og lave en slags leaderboard.

## **Bilag**

### **Bilag 1.**

Her ses at vores funktionalitet til email verifikation virker, hvor man kan se i sendgrid at vi har lavet nogle testforsøg, efterfulgt af en email der faktisk er blevet modtaget af Nickolaj, fra Erik. - Som der står i koden, hvis nogen(Niels) ønsker så kan vi gerne dele API Key midlertidigt.

## Statistics Overview



## Sending with SendGrid is Fun



Denne meddelelse er blevet identificeret som uønsket. Ikke uønsket



erikhey7@hotmail.com <erikhey7@hotmail.com>

22-05-2021 11:33

Til: [nickolaj99@outlook.com](mailto:nickolaj99@outlook.com)

and easy to do anywhere, even with Node.js