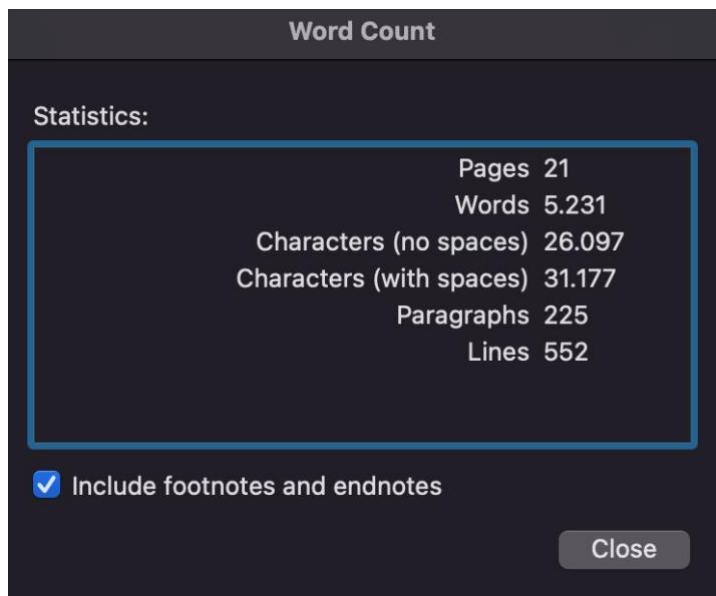


Programmering og udvikling af små systemer samt databaser

# EKSAMENSOPGAVE 2020



Studienummer: 143744

Navn: Mathias Nielsen

CBS HA (it.)

Dato: 11.12.2020

Vejleder: Nicolai Frost Jacobsen og Henrik Thorn

Sidetæl: 21

Anslag: 31.177

## Indholdsfortegnelse

<b>Kravs-specifikationer.....</b>	<b>4</b>
Funktionelle krav .....	4
Use Case Diagram .....	4
<b>Struktur og arkitektur .....</b>	<b>5</b>
MVC .....	5
Brug af MVC.....	5
<b>Analyse .....</b>	<b>8</b>
Klassediagram.....	8
Klienten.....	8
Storage.....	9
MongoDB.....	9
Fordele og ulemper ved brug af MongoDB .....	11
API / Serveren.....	11
Opdatere sin egen profil.....	12
Like en foreslået profil .....	13
Fjern match.....	14
<b>Testing og debugging .....</b>	<b>16</b>
Unit testing.....	16
Console.log().....	17
Postman .....	17
<b>Forretningsmæssige begrænsninger.....</b>	<b>19</b>
Potentielle matches .....	19
Betalende brugere .....	19
<b>Procesevalueringen .....</b>	<b>21</b>
<b>Konklusion.....</b>	<b>22</b>
<b>Litteraturliste .....</b>	<b>23</b>
<b>Bilag.....</b>	<b>27</b>

<b>1. Use case diagram .....</b>	<b>27</b>
<b>2. Klasse diagram.....</b>	<b>28</b>
<b>3. Gitlog .....</b>	<b>29</b>

## Kravspecifikationer

Dette afsnit har til formål at uddybe hvilket kravspecifikationer app'en skal opfylde.

Kravspecifikationer er med til at give overblik over programmets funktionalitet, og hvordan programmets dele skal hænge sammen.

### Funktionelle krav

1.	App'en skal tillade en bruger at oprette en profil
2.	App'en skal tillade en bruger at slette sin egen profil
3.	App'en skal tillade en bruger at opdatere sin egen profil
4.	App'en skal tillade brugeren at logge ind
5.	App'en skal tillade at hvis en bruger er logget ind kan de forblive logget ind.
6.	App'en skal gøre det muligt for en bruger at vælge like eller dislike for en foreslået profil
7.	App'en skal give brugeren en notifikation, såfremt begge profiler har liket hinanden
8.	App'en skal gøre det muligt for en bruger at logge ud
9.	App'en skal kunne vise en liste over en aktuells brugers matches
10.	App'en skal kunne vise en fuld profil for et potentielt match
11.	App'en skal give brugeren mulighed for at fjerne et match igen

### Use Case Diagram

Med oven stående funktionelle krav, er det muligt at opstille et use case diagram, for at give et bedre visuelt perspektiv. Use case diagram er et simpelt diagram, der visualiserer forholdet mellem aktørerne, systemet og dets funktionalitet.

Dette program er udviklet således, at der er to aktører, brugeren som besøger hjemmesiden og en admin. Da admin delen er blevet fjernet fra kravene, tager jeg kun udgangspunkt i brugeren (Bilag 1).

## Struktur og arkitektur

Strukturen af programmet har spillet en stor rolle i udviklingsprocessen af projektet. I de tidligere godkendelsesopgaver har struktur ikke haft nogen stor betydning. Der galt det bare om at programmet virkede og ikke andet.

Nu med større og mere udviklet program, har struktureringen givet bedre overblik over, programmets indhold og dynamik. Samt givet en bedre forståelse for hvorfor, det er vigtigt at have fokus på struktur, når man sidder og udvikler systemer.

Mit program er blevet struktureret efter MVC – frame Work.

## MVC

MVC står for Model, View og controller.

**Model:** Er den del af systemet der er relateret til data håndtering. Hvordan data 'en er struktureret, og hvordan den data 'en modtaget og sendt.

**View:** Er den del som brugeren får vist. Her bliver data 'en præsenteret. Alt bruger Interaktion foregår også her.

**Controller:** Er ofte bindeleddet mellem view og model. Controlleren håndterer bruger interaktionen ved at sende kommandoer til model. Det kunne f.eks. være at model skulle opdatere et dokument. Det vil sige at controlleren manipulerer med data 'en.

MVC er en populær måde at strukturere sine projekter på. Man deler ens projekt op i 3 dele, for at holde dem adskilt, men stadig sammenkoblet.

Det gør udviklingsprocessen mere effektiv, og giver mulighed for at flere kan arbejde samtidigt på projektet uden at der opstår komplikationer.

## Brug af MVC

Jeg har struktureret mit program følgende ved brug af MVC-framework.

**Model:** Min model er delt op i to dele. Jeg har en mappe der hedder models. Den indeholder filerne: adminModel.js og userModel.js.

Der er oprettet et skema i filerne. Det fungerer således, når der bliver sendt data til databasen, tager mit skema og konstruere et userobjekt af det. Det sørger for at data 'en bliver oprettet ens. Det vil altså sige at, den går ind og håndtere data 'en. For at sørge for de stadig er sammenkoblet med resten af API 'en, bliver de eksporteret videre til routes mappen, hvor alle controllerne er placeret.

Den anden del er selve databasen, hvor alt data bliver sendt til og fra. Databasen bliver tilsluttet i server.js. Her bliver den sat op ved hjælp af Mongoose.

**View:** Min view ligger i mappen "views". Her er alt placeret, som brugeren agerer med plus nogle CSS-filer.

Controller:

Mine controllers befinder sig i routes mappen. Mine routes fungerer som controllers. For at kunne bruge mine routes, skal API 'en først vide hvilke routes den kan gøre brug af. Det bliver illustreret i nedenstående kode:

```
25 //Starter mine routes op
26 app.use('/admin', adminRoutes);
27 app.use('/user', userRoutes);
```

*server.js linje 26*

Når '/user/signup' bliver efterspurgt i browseren, leder API 'en efter om der er nogle routes med '/user' i sig. For at finde dem, skal de først importeres fra routes mappen:

```
8 //Routes
9 const adminRoutes = require('./routes/adminRoutes');
10 const userRoutes = require('./routes/userRoutes');
```

*Figur 1 server.js linje 9*

Nu kigger den i userRoutes.js for at finde den præcise routes, der er blevet efterspurgt. Den kører derefter den tilhørende kode.

Koden fortæller derefter at app 'en, hvis ingen fejl er opstået, skal den render index filen til brugeren for eksempel.

```
39 res.render('index');
```

*Figur 2 userRoutes.js linje 39*

Det er sådan forbindelsen mellem view og controller fungerer i programmet.

Der opstod ofte forvirring, når jeg havde 8 forskellige endpoints i samme fil.

En ændring jeg kunne have foretaget mig. Vil være at splitte routes op i forskellige filer. Således routes i samme fil render den samme side, eller i hvert fald ikke gå videre til en anden.

Det vil også give et bedre overblik over hvilke funktionelle krav hver mappe dækker.

Det samme gælder for views mappen, læg filerne i forskellige mapper. Det vil gøre det mere overskueligt for andre og mig selv. Især hvis programmet skal udvides på et senere tidspunkt.

Samlet set har MVC framework været til en stor hjælp. Det har givet mig et bedre overblik over strukturen i programmet og forhåbentligt også for andre der læser min kode.

Samtidig har det også gjort det nemmer at rette mine fejl, da alt er separeret fra hinanden.

## Analyse

### Klassediagram

Klassediagram er med til at give en bedre forståelse for programmet funktionalitet og struktur. Use case diagrammet giver et overblik over sammenhængen mellem klasserne i programmet, hvor klassediagram viser klassernes funktionalitet og hvordan de spiller sammen. Derved skabe et grundlag for et objektorienteret program. Med det i bagtanke konstruerede jeg et klassediagram med UML-notation med MVC-framework som baggrund. Diagrammet er opbygget på baggrund af det endelige resultat af programme (Bilag 2).

### Klienten

Til klienten har jeg valgt at gøre brug af en template engine. En template engine gør det nemmer at indsætte data i et HTML dokument.

Jeg har valgt at bruge EJS som template til mit frontend. Det bibeholder stadig HTML syntax 'en. Det er nemt for begyndere, da det er almen JavaScript, det bliver skrevet i. Det gør at man ikke skal lære en syntax eller lignende.

EJS gør det muligt at gøre brug `res.render()` i express. Første parameter den tager, er hvilken fil den skal render til brugeren. Derefter kan den tage hvilken data, der skal bruges på given fil. I nedenstående billede bliver `home.ejs` rendered til brugeren når der bliver logget ind. `"users[0]"` og `"list"` bliver også sendt med. `users[0]` er brugeroplysninger på personen der logger ind. `"list"` er en liste over alle brugere i databasen, på nær den der logget ind.



```
69 res.render('home', { 'user': users[0], 'ul': list});
```

Figur 3 `userRoutes.js` linje 69

Ved at sende oplysningerne med, gør det muligt at vise disse oplysninger for brugeren. For at vise en liste over alle brugere laver jeg et for each loop i `home.ejs`.



```

145     <% ul.forEach(function (ul) { %>
146         <form method="POST" action= "http://localhost:3000/user/like/<%=user.id%>" name="LikeForm"
147         <h3><%= ul.name %></h3>
148         <ul>
149             <li> Age: <%=ul.age%> </li>
150             <li> Interest: <%=ul.interest%> </li>
151             <li> Email: <%=ul.email%> </li>
152         </ul>
153     <br>

```

Figur 4 home.ejs linje 145

Alt kode der er inde i funktionen, bliver gentaget antallet af brugere i databasen. Det medfører man ikke behøver at gentage en masse kode for at vise alle brugere.

EJS arbejder med Express, så man skal sætte det op i API 'en. Problemet ved dette er, at frontend bliver derfor en del af API 'en. Hvis serveren ikke kører, er det ikke muligt at åbne EJS filerne i browseren. Det forhindrer systemet i at være 3 tier model, da klienten og serveren ikke er adskilt.

Derudover har jeg også brugt simpel CSS for at gøre frontend mere brugervenligt.

## Storage

### MongoDB

MongoDB er en objekt- orienteret NoSQL database Det er baseret på at gemme data i dokument form.

Hvert dokument indeholder properties med tilhørende værdier, hvor hver værdi kan defineres ud fra hvilken type data, der arbejdes med f.eks. strings, array eller booleans. Alle dokumenter får også deres eget unikke ObjectId, så hvert dokument kan adskilles fra hinanden. For at kategorisere dokumenter, kan de blive delt op i forskellige collections.

Collections er tilsvarende til "tabel" i SQL Server. (Lauren Schaefer, 2020)

```

_id: ObjectId("5fafd24dddeddf3e900b34")
name: "Mads "
age: 21
interest: "Food "
email: "mads@gmail.com "
password: "123 "
likes: Array
  0: "5fba47efe08b9552979dc392 "
  1: "5fa55236860eb237fce14f58 "
matches: Array
  0: "5fa55236860eb237fce14f58 "
_v: 0

```

```

ObjectId
String
Int32
Int32
String
String
String
Array
String
String
Array
String
String
Int32

```

Figur 5 udkast fra MongoDB collection - user

Udover MongoDB, bruger jeg også brug af Mongoose.

Mongoose er et Node.js bibliotek for MongoDB. Det er et værktøj, der hjælper med at strukturere ens data, så det er kommunikationsledet mellem API 'en og MongoDB.

(Mongoose, 2020)

For at definere hvordan indkommende data skal struktureres bruges 'Schema' fra Mongoose biblioteket. I nedenstående kode oprettes en klasse ved hjælp af 'Schema'.

```
4  const userSchema = mongoose.Schema({
5    _id: mongoose.Schema.Types.ObjectId,
6    name: String,
7    age: Number,
8    interest: String,
9    email: String,
10   password: String,
11   likes: Array,
12   matches: Array
13 });
```

Figur 6 userModel.js linje 4

Hver 'key' definerer en property i dokumentet med en associerende værdi. Alle brugere der bliver oprettet, vil komme til at have ovenstående properties. Likes og matches vil være 2 tomme arrays, da brugeren ikke selv kan direkte give dem værdi. Dette sker først, når de har liket en anden bruger. Schema bliver brugt, når der bliver oprettet en bruger. For at tilgå klassen, skal den requires inde i userRoutes.js filen.

### Oprettelse af profil

Efter signupForm er blevet udfyldt og der bliver trykket på "Create User" knappen bliver Json data POST 'ed til /user/signup routeren, hvor informationen vil blive behandlet.

```
<form method="POST" action="http://localhost:3000/user/signup" id="form2" name="signupForm" role="form" enctype="application/json">
```

Figur 7 index.ejs linje 77

For at kontrollere at e-mailen der er blevet indtastet ikke allerede, er i brug. Bruger jeg "find()" metoden (MongoDB, db.collection.find(), 2020). Den finder det/de dokumenter i collectionen der matcher givet kriterie.

```
19   User.find({ email: req.body.email})
```

Figur 8 userRoutes.js linje 19

Ovenstående kode går ind i "User" collectionen og kigger i alle e-mail attributterne, og returnere et array af de brugere, som har samme e-mail. Herefter har lavet et if/ else

statement. Hvis længden af dette array er  $\geq 1$  får brugeren en besked om at den e-mail, de har indtastet allerede, er i brug.

Hvis det ikke er tilfældet, bliver der oprettet en nyt objekt ud fra det Schema med alt informationen og kalder derefter "save()", der gemmer den nye bruger i databasen. Efter brugeren er blevet oprettet, beder routeren om at render index.js.

### Fordele og ulemper ved brug af MongoDB

Ud fra opgavebeskrivelse er der ikke behov for at gøre brug af en database. Programmet skal ikke kunne håndtere store mængder data, med mange forskellige data strukturer. Det er relativ simple objekter.

Samtidig har jeg brugt meget af min tid på at lære om MongoDB og dens funktioner. Hvilket har medført, at jeg nok også har været med til at overkompliceret projektet. Tiden kunne have være blevet brugt på noget andet, såsom en anden storage form, som var blevet gennemgået i løbet af kursusens forløb. Jeg vil derved have en fundamel forståelse for fremgangsmåden i forvejen.

Med det sagt, er der mange fordele ved brug af MongoDB. Når man først har sat sig ind fremgangsmåde og have lært det fundamentale. Har MongoDB meget at byde på. Det har indbygget metoder i Mongoose, der gør det meget nemt at håndtere og manipulere med objekterne. Et godt eksempel er: deleteOne() og find(). deleteOne() sletter et dokument ud fra de kriterier man giver den. MongoDB (2020). Da alle objekter har et unikt Id, er det nemt at vælge et specifikt man vil have slettet. Find() har stort set brugt i alle mine CRUD endpoints, den er simpel og man slipper for at skrive mange linjer kode. Det sparer både tid og det resulterer i en mere overskuelig kode. Alternativet til find() vil være et for loop, der looper igennem alle dokumenterne i den givne collection. Med en if statement bagefter der kontroller om det givne objekt har det ønskede Id og derefter returnere det.

### API / Serveren

Serveren er 'hjertet' af programmet. Det er bindleddet mellem resten af Api 'en.

Serveren har jeg opsat i filen server.js. Den er udarbejdet med Express som middleware.

Udover Express har jeg også gjort brug af 2 andre værktøjer. Nodemon var noget af det første værktøj, jeg gjorde brug af. Nodemon genstarter ens API efter ændringen er foretaget

i koden. Det giver en mere flydende arbejdsproces, da man ikke hele tiden selv skal genstarte ens API.

Bodyparser er et værktøj til Node.js baseret applikationer. Det håndterer indkommende requested data i et middleware, som kan hentes ved brug af req. body.

I alle mine routes har jeg gjort brug af asynkron programmering.

Jeg har i mine routes, funktioner som har brug for hinanden. Jeg har valgt at benytte promises til dette med metoderne then() og catch(). Then() er en metode, der siger hvad der skal ske når et promise er kørt. Catch() bruges når et promise har fejlet ( Jacobsen, N. F., 2020).

### Opdatere sin egen profil

For at gøre det muligt at brugeren kan opdatere sine oplysninger. Har jeg først defineret et objekt "updatedUser" med property 's, der definerer de oplysninger, der er mulige at ændre. Med værdien af tilhørende inputfelt. Dvs. når brugeren trykker "Update", bliver der lavet et nyt objekt med de nye informationer.

```
86     var updateUser = {  
87         name: req.body.name,  
88         age: req.body.age,  
89         interest: req.body.interest,  
90         email: req.body.email,  
91         password: req.body.password  
92     }
```

Figur 9 userRoutes.js linje 86

Det næste der bruges er updateOne(), hvilket er en metode der opdatere et enkelt dokument i databasen. Den tager to parameter, det første er hvilket dokument der skal opdateres, det andet er hvad der skal opdateres i dokumentet. Første parameter leder efter property 'en "\_id" som har værdien req.body.id. Problematikken ved dette er, at brugeren først selv skal indtaste sit ID, for at routeren forstår hvilken bruger der skal ændres. En simpel løsning på det kunne være at give input feltet værdien af brugerens ID, og derefter bruge "style="display:none;" ", da brugeren ikke har behov for at vide sit ID.

Efterfølgende gør jeg brug af \$set: til at definere hvad der skal opdateres på det valgte dokument. \$set kan bruges til at erstatte allerede eksisterende data med ny i et dokument (MongoDB 2020). Til sidst tilføjes det ny oprettet objekt "updatedUser", da jeg gerne vil

erstatte det gamle data med det nye. Det er vigtigt at property 'ene har præcis samme navn som user schema 'et. Ellers vil der blive oprettet helt nye property 's i dokumentet.

Problemet ved denne fremgangsmåde er, at brugeren bliver nødt til at ændre alle ens oplysninger for ikke at slette resten af oplysningerne. Hvis et af input- felterne ikke bliver udfyldt, vil property 'en blive tom.

En løsning på dette vil være at give input felterne brugerens oplysninger på forhånd. Ved at give dem en værdi, således: `value='<%= user.name %>'`

### Like en forslået profil

Jeg endte ud med to forskellige løsninger, der kunne hjælpe med løse problemet.

Nedenstående eksempel er det, jeg valgte at implementere i programmet. Jeg startede med at oprette to variabler: "firstId" og "secondId". FirstId defineres som "req.params.id". req.params er et objekt, der indeholder de parameter som er en del af URL 'en (Express, 2020). Denne måde gør det nemmere at adskille brugeren som er logget ind med resten af brugerne i databasen. secondId defineres som "req.body.id", som er ID 'et på den likede profil.

Derefter bliver secondId tilføjet til brugerens property "likes", som er et array af alle personer, brugeren har liked. Hvilket er det første promise i denne funktion. Til dette gør jeg brug af metoden \$addToSet (MongoDB, \$addToSet, 2020). Den tilføjer en given værdi til et array, medmindre værdien allerede er præsenteret i array 'et. Det er også med til at forhindre brugeren i at like den samme person 2 gange.

```
134 | User.updateOne({_id: firstId}, {$addToSet: {"likes": secondId}})
```

Figur 10 userRoute.js linje 134

Bagefter bruger jeg then() til beskrive hvad der skal ske efter det promise har kørt.

Efterfølgende opretter jeg en variable, hvor jeg bruger findOne() til at finde brugeren, man lige har liket. Det bliver det andet promise. Dernæst skal jeg finde ud af om brugeren man lige har liket, også har liket en selv. Her looper den igennem den likede bruger egne likes array med et for loop. Hvis begge bruger har liket hinanden, bliver deres ID tilføjet til hinandens property "matches", og der er nu dannet et match for begge brugere.

En alternativ løsning er den første jeg lavede. Tanken bag denne, var at lave en helt ny collection med de forskellige brugers matches. Således at "matches" property 'en blev fjernet og hver match vil blive smidt ind i en til collection. Jeg brugte metoden insert(), der indsætter et eller flere dokumenter i en collection (MongoDB, Insert Documents, 2020). Jeg valgte at gå væk fra den her løsning, fordi der vil blive oprettet alt for mange collection til at holde styr på. Lad os sige der er 100 brugere, der hver har et match. Dvs. at der bliver lavet 100 nye collections. Det medfører, at det bliver svært at holde styr på hvilken collection hører til hvilken bruger.

Derfor virkede det bedre at oprette 2 nye property som arrays til hver bruger, hvor den likede brugers Id ryger ind.

### Fjern match

For at fjerne et match har jeg gjort brug af en POST request. Der går til routen: "http://localhost:3000/user/matches/:id/delete". Jeg smider Id 'et på brugeren ind i routeren, for nemmere at kunne adskille mellem den bruger der er logget ind og den brugers matches.

Der bliver brugt 2 metoder fra MongoDB: update() og \$pull.

**Update:** Det første der skal ske, er at finde det objekt, hvor matchet skal fjernes. Update() ændrer et eller flere dokumenter og tager 2 parameter. 1. parameter er kriteriet for hvilket dokument der skal ændres. 2. parameter er hvilken property der skal ændres i.

**\$pull:** er en operatør, der bruges til at manipulere med arrays. Den fjerner elementer der matcher givet parameter. \$pull tager en property og en værdi, der indikerer hvilke/hvilket element der skal fjernes (MongoDB, \$pull, 2020).

```
186 router.post('/matches/:id/delete', (req, res) =>{
187   // $pull fjerner element i array der matcher en given værdi
188   User.update({_id: req.params.id}, {$pull: {matches: {$in: req.body.id}, likes: req.body.id}})
189   .then(
190     User.update({_id: req.body.id}, {$pull: {matches: {$in: req.params.id}, likes: req.params.id}})
191     .then(
192       res.send('Match deleted --- press back arrow')
193     )
194   );
195 });
```

Figur 11 userRoutes.js, linje 186

I linje 188 defineres først hvilket dokument der skal opdateres ud fra `_id` property.

`Req.params.id` er brugerens eget ID. 2. parameter tager fat i de 2 property's der skal opdateres, hvilket er `matches` og `likes`.

Da begge af disse datatyper er arrays, er det muligt at bruge `$pull`. Den fjerner alle elementer der matcher `req.body.id`, som er id 'et på brugerens match. I linje 190 bliver det samme også gjort for `matches` profil ved at bytte om på `req.params.id` og `req.body.id`. For at fjerne `matches` fra hans profil også.

Fordelen ved ovenstående eksempel er at det medfører simpel og let læseligt kode. Der har meget funktionalitet

Ulempen er, at det er to metoder der er forbundet med MongoDB, så hvis man ikke har kendskab til MongoDB, forstår man ikke kodens funktionalitet og syntax. Det kræver man sætter sig ind i det.

En alternativ fremgangsmåde ville være: Lavet et for loop der looper igennem "likes" og "matches" property efterfulgt af en if statement, der leder efter id 'et på den bruger man vil fjerne. Efterfulgt af `arr.splice([i], 1)`.

`Splice` metoden ændrer indholdet af et array, ved enten ved at fjerne et element eller erstattet. Første parameter angiver index-pladsen, hvor man vil starte.

2. parameter skal være en integer, der angiver antal elementer der skal fjernes fra start værdien (1. parameter).

Hvis man vil erstatte et element, skal man også angive et 3. parameter, som skal være det, der skal smides ind i arrayet f.eks. en string.

Min brug af CRUD endpoints har heller ikke været helt optimal. Jeg har brugt POST request på alle mine routes, selvom jeg ikke skulle POST noget. Jeg kunne have brugt en GET request i routen, der gør det muligt at vise brugerens matches. I den her router opretter jeg ikke en nyt match eller lignede. Routen henter kun data. Derfor vil en GET request være fordelagtig.

## Testing og debugging

Testing og debugging har haft stor betydning i forløbet af udviklingen af programmet. Det har været med at opdage fejl og mangler, for at få koden til at køre optimalt uden udfordringer. For at opnå dette, har jeg gjort brug af forskellige testing metoder passende til programmets forskellige funktionalitet. Min fremgangsmåde til testing af programmet har været, at jeg undervejs af kodningen har foretaget test af programmet for at fejl kunne undgås og sikre at programmet kører optimalt.

Det har været med til at give et bedre overblik over hvilke dele af koden, jeg er blevet færdig med og virker, så jeg kan begynde at fokusere på næste del, uden at tænke over om tidligere kode rent faktisk fungerer.

## Unit testing

Jeg har valgt at lave en unit test på kravspecifikationen ” App’en skal tillade en bruger at oprette en profil ”. For at lave min unit test har jeg brugt framework Mocha.

Til mine assertions har gjort brug af Chai. Jeg har valgt at benytte mig af ”Chai expect”. Chai er et bibliotek der hjælper med at sammenligne outputtet med den forventet værdi. Chai ’s syntax er næsten ligesom alment engelsk.



```
26 | expect(test).to.be.a('object');
```

Figur 12 test.js linje 26

Ovenstående kode tjekker om ”test” er et objekt.

Jeg har oprettet en enkel test case med mange assertions. Dvs. der bliver udført en enkel test, men den enkel test tester for mange ting. Den første assertions tjekker, om brugeren der bliver oprettet, er et objekt. De næste assertions tjekker om dette objekt har de rigtige property ’s.

Det vil være bedre at lave flere forskellige test cases med færre assertions. Det vil give et bedre overblik over hvilke test, der bliver godkendt og hvilke der fejler.

Dog er der et problem med denne fremgangs måde. Hver gang der bliver kørt en test, bliver der oprettet en ny bruger i databasen. Så efter at have kørt min test et par gange, var der lige pludselig mange brugere i databasen. Så hvis jeg havde lavet endnu en test case, ville der blive oprettet to bruger, hver gang der blev testet.



## Console.log()

Console.log har jeg gjort meget brug af i løbet af udviklingen af programmet. Jeg har tit placeret console.log efter kompliceret funktioner eller når jeg har skulle hente data fra databasen. Formålet ved det, har været at være sikker på at funktionen returnerer det rigtige, så jeg ikke begynder at arbejde videre med de forkerte resultater.

Når jeg har sku arbejde med data fra databasen, har jeg tit sat en console.log(), efter jeg har prøvet at hente informationer fra databasen. Det har sikret, at jeg for fat i de rigtige oplysninger.

I nedenstående eksempel requester jeg to ID fra databasen. For at sikre mig at de rigtige ID, indsætter jeg en console.log

```
138     var firstId = req.params.id; // First user = den person der liker
139     var secondId = req.body.id; // Second user = den person der bliver liked
140
141     console.log(firstId, secondId);
```

*userRoutes.js linje 138*

Der er dog nogle begrænsninger ved brugen af console.log. console.log kan kun teste et output ad gangen. Den kan heller ikke hjælpe med at sige, hvor i funktionen er der fejl kun, at der er en.

Eftersom ens program bliver større, opstår der højt sandsynligt også flere funktioner og andre ting der skal testes. Derfor skal man huske at fjerne dem undervejs, så terminalen ikke bliver spammed med alle de console logs man ikke bruger mere, hvilket kan være meget tidskrævende.

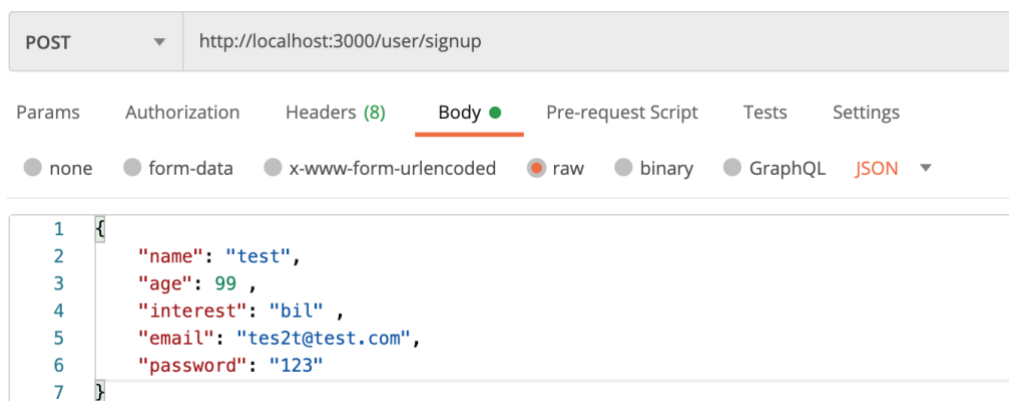
## Postman

Udover console.log metoden har jeg også gjort meget brug af Postman. Postman har hjulpet med at teste min API og dens CRUD endpoints.

Postman er et API testing værktøj, der gør det nemmere for udviklere at teste og videre udvikle deres API 'er.

Postman gør det muligt at håndtere mange forskellige slags request, hvilket browseren ikke har mulighed for at gøre.

Mit primære formål med brugen af Postman var at kontrollere at mine CRUD endpoints fungerede, som de skulle. Mit første punkt var at sikre mig at API 'en var sat op korrekt. Derfor oprettede jeg en simpel GET request, der skulle sende beskeden "Hello World". Mit næste punkt var opsætningen af min første POST request, hvilket vil være mit endpoint for at brugeren kan oprette en ny profil. Da jeg ikke havde oprettet en frontend endnu, blev jeg nødt til at hardcode den nye bruger. Postman giver også værktøjerne til dette. Da jeg sender rå JSON-tekst, er det vigtigt, at brugeren er opstillet som et JSON-objekt.



Udskrift af Postman

Test med Postman bidrog med at give mig en mere dynamisk testmetode, hvilket resulterede i hurtigere og mere effektiv udvikling af programmet. Samtidig havde det også en effekt på mig, der var med til at give mig en bedre forståelse for funktionaliteten af de forskellige endpoints.

## Forretningsmæssige begrænsninger

I følgende afsnit vil jeg pointere hvilke forretningsmæssige begrænsninger programmet har i forhold til et færdig lavet program. Jeg vil derfor komme ind på mulige features og udfordringerne, ved implementeringen af dem.

### Potentielle matches

Lige nu er visningen af potentielle matches meget simpel og statisk. Alle brugere i systemet bliver vist på nær personen der er logget ind. Man kunne til fordel gøre det lidt mere dynamisk. Ved at sortere listen af potentielle matches efter alder, interesser og ikke mindst køn. Dette kunne gøres ved hjælp af en algoritme.

Jeg har lavet en begrænsning i forhold til om kan vælge køn. Det vil være fordelagtigt ikke at gøre i det færdige produkt, da køn spiller en ret stor rolle i "datingverdenen". Det vil medføre at alle brugere skal have to nye property 's: deres eget køn og det køn de er interesseret i. Løsningen til dette kunne være at give "userSchema 'et" de to nye property 's. Det medfører dog kun at alle ny oprettet brugere får dem.

Brugerne der allerede er i databasen, vil derfor ikke have dem. Det vil man kunne løse ved at lave en funktion med nested for loops. Første loop skal loope igennem et array af alle brugerne. Det andet loop skal loope igennem hvert enkelt brugers property 's. Efterfuldt af et if statement, der tjekker om de har en property ved navn "køn" og "fortrunket køn". Hvis de ikke har det, skal man bruge "Object.assign()". Den tilføjer en ny property til givet objekt, og værdi hvis det er angivet.

### Betalende brugere

Virksomheden, som udvikler dette program, vil gerne kunne tjene penge på det. Der er to muligheder: reklamer eller brugerne skal betale for brugen af app 'en. I dette punkt vil jeg have fokus på muligheden for betalende brugere.

Hvis tilføjer betalende brugere, skal de have nogle ekstra features eller de gratis bruger skal begrænses. Da gratis brugere allerede nu har mulighed for alt, er den bedste løsning begrænsning af gratis brugere.

For at lave en betalende bruger, skal der oprettes en ny collection i MongoDB.

Dokumenterne i den nye collection skal have en ny property: Credit Card som objekt. Så det bliver muligt at tilføje: Kortnummer, udløbsdato, CVV-nummer. Dvs. det bliver et nested objekt. Der skal derfor oprettet et helt nyt Schema for den collection. Hvilken collection de nye brugere bliver oprettet i, kommer an på valget om brugeren vil have en gratis profil eller ej. Da alle brugerne nu er delt op i to forskellige collection. Giver det en hel ny struktur i databasen, men også i selve programmet. Man skal tilbage i routerne for hver kravsspecifikation, og omstrukturere dem. Således det er muligt for begge collection at tilgå de forskellige features.

## Procesevalueringen

Min kode proces har forbedret sig meget i projektforsløbet.

En vigtig ting jeg har lært, er at dele mine problemstillinger op i mindre bider. Jeg har tit startet mine problemstillinger lige på, ved at gå i gang med at løse hele problemet med det samme. Det resulterede ofte i kode, der var uoverskueligt og som ikke fungerede.

Jeg begyndte derfor at ændre mine tilgang til selve kode delen af opgaven.

Jeg startede derfor med at dele problemstillinger op i mindre punkter og løse dem en efter en. Det gik op for mig, den her fremgangsmåde hjælpe mig meget med at få et bedre overblik over strukturen og funktionaliteten.

Jeg begyndte at skrive de mindre punkter ned på papir. Og skrive hvad hvert punkt krævede af kode, for at det kunne løses.

Ovenstående tog jeg til mig, efter jeg var startet på at lave like funktionaliteten i programmet. Den funktion voldte mig store problemer, da der var mange ting, jeg skulle tage stilling til. Jeg havde fremgangsmåden til problemet, men var ikke klar over hvordan det skulle løses. Det havnede i meget ud kommenteret kode, som jeg endte med ikke at gøre brug af alligevel.

Jeg startede derfor helt forfra med den omtalte teknik. Senere i forløbet blev jeg ved med at gøre brug af den teknik. Det gav mig en meget bedre kode skik og struktur i min opgave.

Det gik op for mig hvor betydning, struktur har for udarbejdelsen af et projekt.

Under forelæsningen om struktur og MVC, var det ikke gået op for mig, hvor relevant det vil være. Strukturering har ikke haft særlig meget relevans i tidligere godkendelsesopgaver. De har været meget mindre i form af funktionalitet og antal filer der skal bruges. Jeg har derfor ikke haft den store forståelse for, hvor vigtigt det er.

Brugen af MVC i opgaven har været til stor gavn for mig. Jeg har ofte svært ved at se sammenhængen mellem alle komponenterne i et program. Og hvordan det hele er struktureret.

Min fremgangs måde til implementering af MVC. Begyndte da jeg startede på projektet. Jeg startede med at oprette filer og mapper, jeg tænkte, kunne være relevant i forhold til projektet. Jeg gav dem en nogen lunde MVC-struktur. Dette blev selvfølgelig rettet undervejs. I takt med at der blev lavet flere filer og mapper. Det medførte derfor en stille og rolig implementering af MVC.

## Konklusion

I dette projektforsøg har jeg udarbejdet et "Minimum Viable Product" af en dating app. Der har basisfunktionaliteten af en almen dating app. Programmets opbygning er bygget på baggrund af objektorienteret programmering.

For udarbejdelse af programmet har struktur spillet en stor rolle. Det er blevet struktureret efter MVC – framework. Der har medført et bedre overblik og struktur over projektet.

Det har samtidig medført en bedre udviklingsproces.

Udover MVC, er programmet udviklet med henblik på 3- tier arkitekturen. Der indebærer klienten, storage og API 'en.

Klienten er API 'ens frontend. Klienten består af alt, som brugeren har interaktion med. Her har brugeren mulighed for at se data og sende request videre til kontrolleren. Storage har været en stor del af opgaven. Her bliver alt data behandlet og placeret. Brugen af en database har givet programmet en masse muligheder, som ikke var mulige uden. Det har ligeledes givet API 'en potentielle til at skalere. På baggrund af de funktionelle krav til opgaven. Har der ikke været behov for en database. Jeg vil mene brugen af database, har givet en mere negativ effekt fremfor positiv.

Det centrale i programmet er API 'en. Her har jeg mine controllers placeret. Det er her alt logikken bag programmet ligger. Her bliver data manipuleret, såsom når brugeren skal opdatere sin profil eller like en anden profil. Routes 'ene er kodet med asynkron programmering, for at give dem bedre udførelse. Det gør det muligt for funktionerne at køre parallelt med hinanden, for at forhindre reaktionstid.

For at finde og løse fejl i programmet har jeg gjort brug nogle test former. Unit testing har jeg anvendt til test routeren for oprettelse af en bruger. Console.log har jeg brugt til at teste om funktionerne har det korrekte output og om variabler og objekter har de rigtige værdier. Da der er tale om et "Minimum Viable Product", er der nogle forretningsmæssige begrænsninger ved dette. Programmet har ikke alle de nødvendige features, hvis skal bruges ude i den virkelige verden. Med det i tankerne, er der stadig mulig for et skalerbart program.

## Litteraturliste

Academind. (2020, October 25). Building a RESTful API with Node.js [Video file].

Retrieved from [https://www.youtube.com/playlist?list=PL55RiY5tL51q4D-](https://www.youtube.com/playlist?list=PL55RiY5tL51q4D-B63KBnygU6opNPfk_q)

[B63KBnygU6opNPfk\\_q](https://www.youtube.com/playlist?list=PL55RiY5tL51q4D-B63KBnygU6opNPfk_q)

Academind (2020 November 22) Node.js + Express - Tutorial - Update and Delete Data with MongoDB [Video file]. Retrieved from

<https://www.youtube.com/watch?v=-JcgwLIh0Z4&t=5s>

Chris On Code. (2020, November 6) How To Use EJS to Template Your Node Application

Display mongo data with EJS. Retrieved from

<https://www.digitalocean.com/community/tutorials/how-to-use-ejs-to-template-your-node-application>

Devhints. (2020 November 30) Chai cheat sheet. Retrieved from

<https://devhints.io/chai>

Express. (2020 December 8) Routing. Retrieved from

<https://expressjs.com/en/guide/routing.html>

Jacobsen, N. F. (2020 December 7) Asynkron programmering i JavaScript med promises

[Video file]. Retrieved from

<https://www.youtube.com/watch?v=snfdulYn1BA&feature=youtu.be>

Jacob Sharir. (Last Updated: Mar 19 2019). How to Use Postman to Manage and Execute Your APIs. Retrieved from

<https://www.blazemeter.com/blog/how-use-postman-manage-and-execute-your-apis>

jay. (2020 November 30) Unit Testing Express : Unit Test Express Route. Retrieved from <https://codehandbook.org/unit-test-express-route/>

Kevin. (2020, November 10) Display Mongo Collection With ExpressJS Using EJS Templates [Video file]. Retrieved from

[https://www.youtube.com/watch?v=0uaSi8v5CHQ&t=1032s&fbclid=IwAR1fQPjRNauyRE3E1jI0iGelyVDOI1ZKG4dMWE0eutAhCn\\_06GpZ5Aitq9g](https://www.youtube.com/watch?v=0uaSi8v5CHQ&t=1032s&fbclid=IwAR1fQPjRNauyRE3E1jI0iGelyVDOI1ZKG4dMWE0eutAhCn_06GpZ5Aitq9g)

Lauren Schaefer (2020 December 4). What is NoSQL? Retrieved from

<https://www.mongodb.com/nosql-explained>

MDN contributors (Last updated: Aug 10, 2020) Array.prototype.splice(). Retrieved from

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Array/splice](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/splice)

MongoDB. (2020 November 22) db.collection.update(). Retrieved from

<https://docs.mongodb.com/manual/reference/method/db.collection.update/>



MongoDB. (2020 November 26) Insert Documents¶. Retrieved from

<https://docs.mongodb.com/manual/tutorial/insert-documents/>

MongoDB (2020 November 27) \$set (aggregation). Retrieved from

<https://docs.mongodb.com/manual/reference/operator/aggregation/set/>

MongoDB (2020 November 28) db.collection.insert(). Retrieved from

<https://docs.mongodb.com/manual/reference/method/db.collection.insert/>

MongoDB (2020 November 29) db.collection.deleteOne(). Retrieved from

<https://docs.mongodb.com/manual/reference/method/db.collection.deleteOne/>

MongoDB (2020 December 3) \$pull. Retrieved from

<https://docs.mongodb.com/manual/reference/operator/update/pull/>

MongoDB(2020 December 3) db.collection.find(). Retrieved from

<https://docs.mongodb.com/manual/reference/method/db.collection.find/>

MongoDB (2020 December 8) \$addToSet. Retrieved from

<https://docs.mongodb.com/manual/reference/operator/update/addToSet/>

Mongoose. (2020, December 6) Models. Retrieved from

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

NPM (2020, December 5) nodemon. Retrieved from

<https://www.npmjs.com/package/nodemon>

Parikshit Hooda. (Last Updated: 14-06-2018) Model-View-Controller (MVC) architecture for Node applications. Retrieved from

<https://www.geeksforgeeks.org/model-view-controllermvc-architecture-for-node-applications/>

Satish. (2020, October 29). Inserting Data into MongoDB: Node.js [Video fil]. Retrieved from

<https://www.youtube.com/watch?v=uZqwHfNlf8M&t=2s>

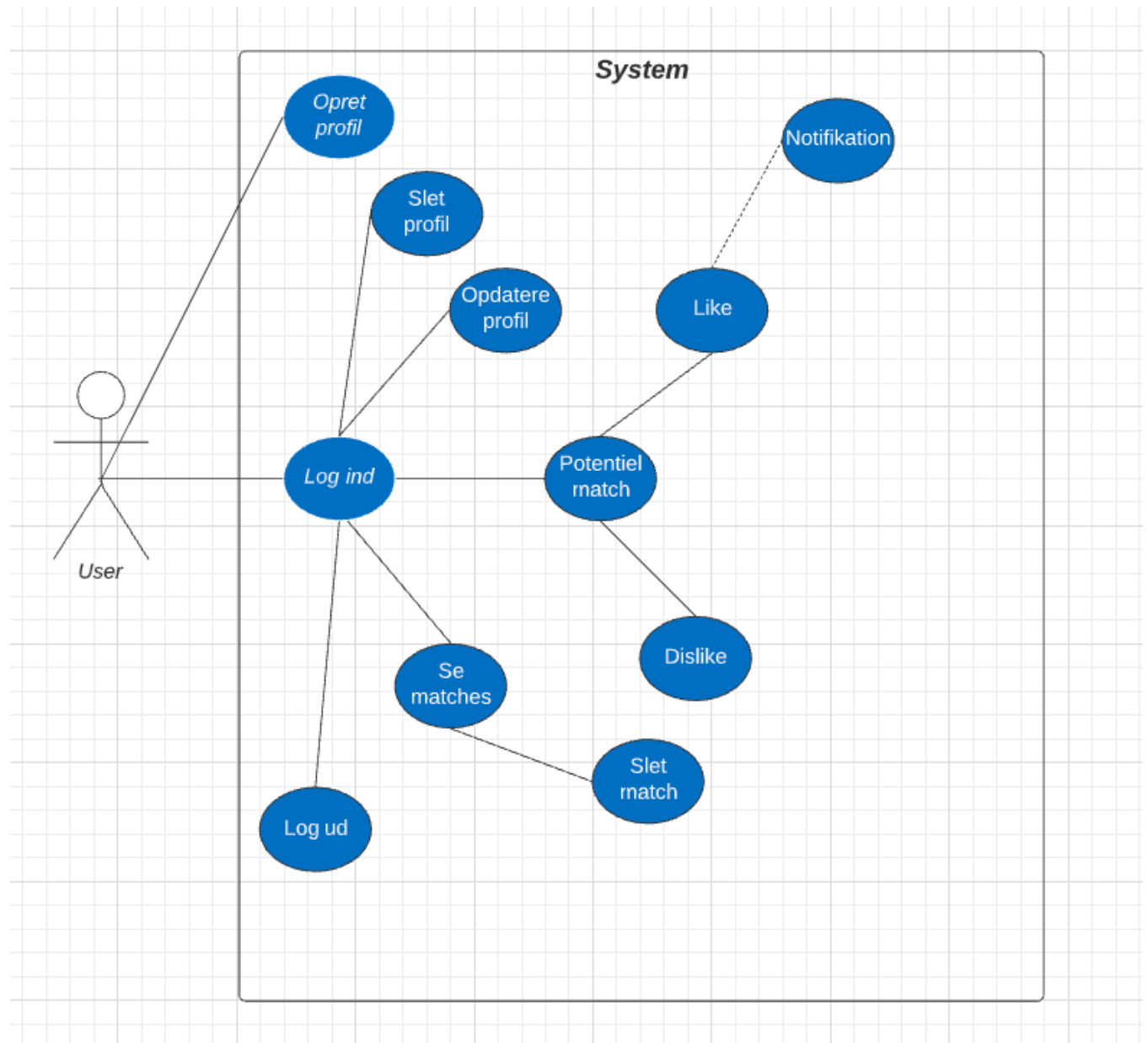
The net Ninja. (2020, October 23). REST API Tutorial (Node, Express & Mongo) #15 - Creating a Front-end [Video file]. Retrieved form <https://www.youtube.com/watch?v=fGQFeV32nwE>

w3schools. (2020 November 11) Node.js MongoDB Delete. Retrieved from

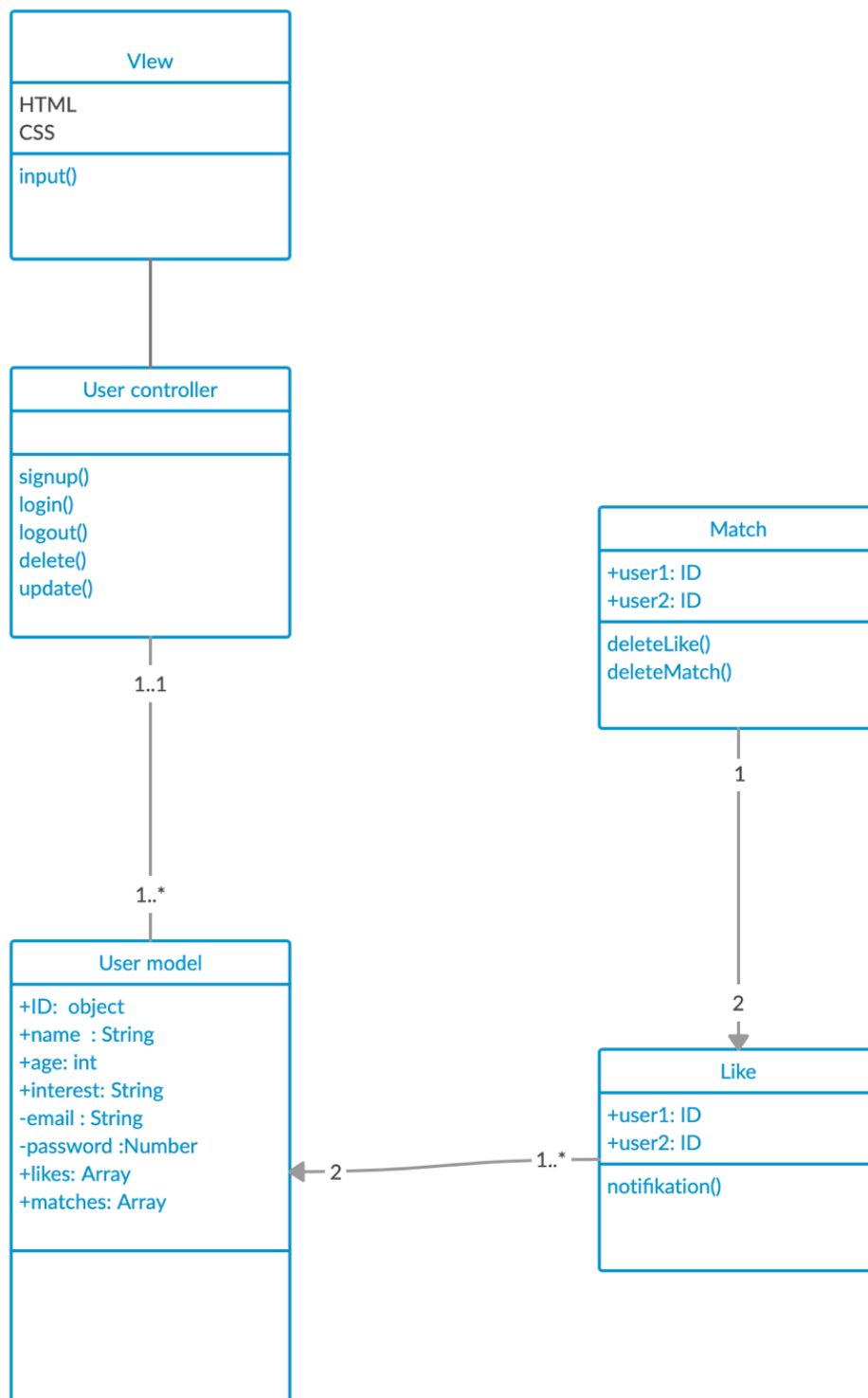
[https://www.w3schools.com/nodejs/nodejs\\_mongodb\\_delete.asp](https://www.w3schools.com/nodejs/nodejs_mongodb_delete.asp)

## Bilag

## 1. Use case diagram



## 2. Klasse diagram



### 3. Gitlog

Følgende billeder er udklip fra Github

Commits on Dec 10, 2020

- Sidste rettelse (ordenlig kode skik) [e1dfdd2](#)   
MathiasN8 committed 23 seconds ago

Commits on Dec 7, 2020

- Rettelser af kommentarer og syntax [7bb66e3](#)   
MathiasN8 committed 3 days ago

Commits on Dec 5, 2020

- test [4f2d8c0](#)   
MathiasN8 committed 5 days ago

Commits on Dec 4, 2020

- rettelser [38c73bb](#)   
MathiasN8 committed 6 days ago

Commits on Dec 3, 2020

- Hår lavet test - mangler dog at rette det sidste på unit testen [e044f15](#)   
MathiasN8 committed 7 days ago
- Brugeren kan nu slette sine matches [eb82dc6](#)   
MathiasN8 committed 7 days ago

Commits on Dec 2, 2020

- Viser alle matches nu [7d8d87c](#)   
MathiasN8 committed 8 days ago

Commits on Dec 1, 2020

- Brugerne kan like hianden og der bliver lavet et match [d2f5b3a](#)   
MathiasN8 committed 9 days ago

Commits on Nov 30, 2020

- unit testing start [e52a941](#)   
MathiasN8 committed 10 days ago

Commits on Nov 29, 2020

- Lavet local storage for login info [2903778](#)   
MathiasN8 committed 11 days ago

Commits on Nov 28, 2020

- Lavet en side for alle match og gjort det muligt at slette dem [c5401f5](#)   
MathiasN8 committed 12 days ago

Commits on Nov 26, 2020

- Opretet så den viser matches på ny side ved tryk på knap [f9b110e](#)   
MathiasN8 committed 14 days ago
- Opretet like og dislike knapper + små rettelser [2e735cc](#)   
MathiasN8 committed 14 days ago

Commits on Nov 24, 2020

- Rettet mine kommentarer [df8807e](#)   
MathiasN8 committed 16 days ago







Commits on Nov 22, 2020

- Lidt CSS på update input felterne [dd128e8](#)   
MathiasN8 committed 18 days ago
- Kan opdatere mine brugere, dog opstår der fejl beskeder [6a069dc](#)   
MathiasN8 committed 18 days ago










Commits on Nov 21, 2020

- Brugeren kan se liste over potentielle matches [f043546](#)   
MathiasN8 committed 19 days ago










## Commits on Nov 13, 2020

test	 MathiasN8 committed 27 days ago	 183d85a	
forside	 MathiasN8 committed 27 days ago	 509dda5	







## Commits on Nov 12, 2020

ret	 MathiasN8 committed 28 days ago	 498284f	
Rettelser af css og html	 MathiasN8 committed 28 days ago	 aeaf4c9	
CSS amok	 MathiasN8 committed 28 days ago	 7e4245f	










## Commits on Nov 11, 2020

oprettet lidt flere divs og knapper	 MathiasN8 committed 29 days ago	 fd7e4d8	
User kan slette sin egen profil og logge ud	 MathiasN8 committed 29 days ago	 2798fa3	
Admin kan nu logge ind og se liste over users	 MathiasN8 committed 29 days ago	 9fbc961	










## Commits on Nov 10, 2020

Oprettet admin side med user liste	 MathiasN8 committed on 10 Nov	 0071ca4	
Oprettet EJS og den indlæser HTML og CSS filerne	 MathiasN8 committed on 10 Nov	 96596bc	

## Commits on Nov 6, 2020

Hentet EJS	 MathiasN8 committed on 6 Nov	 eb20031	
Færdig med login og signup	 MathiasN8 committed on 6 Nov	 9c43b9e	
Oprettet admin routes og user signup route	 MathiasN8 committed on 6 Nov	 688501e	






## Commits on Oct 29, 2020

o	 MathiasN8 committed on 29 Oct	 e1acd7b	
Rettelse	 MathiasN8 committed on 29 Oct	 98acc90	
Oprettet userModel og userRoutes (i gang med implementere mongoose)	 MathiasN8 committed on 29 Oct	 7c97d54	

## Commits on Oct 25, 2020

Oprettet routes og lavet et frontend	 MathiasN8 committed on 25 Oct	 6ce3b73	
--------------------------------------	---	---	---

## Commits on Oct 23, 2020

Json og Server	 MathiasN8 committed on 23 Oct	 76b32f6	
synlighed af mapper	 MathiasN8 committed on 23 Oct	 48d22a7	