Forside

Programmering og udvikling af små systemer samt databaser (BINTO1610D)
1.Semester

Skriftligt produkt

Eksamensnummer: S145325
Afleveres på Digital Eksamen d. 11. december 2020 senest kl. 12:00

Antal anslag og antal sider: 27 sider

Indholdsfortegnelse

1.Introduktion

1.1 Indledning

I denne rapport beskrives implementerings-og udviklingsforløbet for en objektorienteret dating-app. Målet med appen er at udvikle et delprodukt af en ny dating-app, som i fremtiden skal blive videreudviklet og forhåbentligt blive en konkurrent til Tinder på længere sigt. Med udgangspunkt i ovenstående indeholder og beskriver denne rapport udviklingsprocessen for sådan et implementeringsforløb. Kravspecifikationerne tydeliggøres som start-pointe, og rapporten tager afsæt heri - hvorledes analyse, design, implementering, test og dokumentation er anvendt til at nå disse mål. Dertil test, refleksioner og vurderinger af, hvor vidt de valgte implementeringer er optimale. Undervejs diskuteres det løbende, hvorledes implementeringerne kunne være gjort med alternative metoder, og hvad der evt. kunne have været gjort anderledes. Afslutningsvist en perspektivering til den virkelige verden, og programmets fremtidige udviklingsmuligheder.

1.2 Problemområde

Jeg har i denne eksamensopgave konstrueret et administrativt system, der gør det muligt for en bruger at date andre brugere fra en online dating platform. Platformen gør det muligt for brugerne at matche og møde nye mennesker, men stadigvæk holde sig relativ anonym. Systemet tilbyder først og fremmest muligheden for at oprette sig som bruger i databasesystemet og dertil få adgang til følgende funktioner: at se en fuld profil for et potentielt match, derfra klikke for at dislike eller like den forslåede profil, som man muligvis finder interessant. Systemet giver desuden en notifikation, hvis den forslåede profil gengælder interessen. Dertil findes en fuld liste over brugerens aktuelle matches, hvorfra brugeren har mulighed for at fjerne et match igen. Systemet giver selvfølgelig også brugeren mulighed for at slette sin profil igen, hvis dette ønskes.

1.2 Kravsspecifikationer

Herunder stilles der skarpt på kravsspecifikationerne til programmet som en dating-app samt programmets vidde og udstrækning til det formål, at der opstår en dybere forståelse for, hvad der forventes af programmet og dets kunnen. Analysedelen har til forhold at sætte systemets opbygning i et større perspektiv til dets brugere, omgivelser og endelige klare mål.

Dette system er udviklet til at have en primær aktør, som i dette tilfælde er en bruger. Idet programmet udvikles ud fra en primær aktør, betyder dette at alle brugere, der interagerer med programmet behandles ens i systemet. Derudover kræver det selvfølgelig, at der er mindst to brugere gemt i databasen for at programmet kan opfylde alle 11 nedenstående krav.

I dette program skal følgende være muligt for brugeren:

Brugeren skal kunne oprette en profil, hvis ikke denne allerede er tilmeldt, samt tillade en bruger, hvis brugeren allerede er logget ind at forblive logget ind. En tidligere bruger af systemet skal blot kunne logge ind. Brugeren skal af programmet foreslås potentielle match, som brugeren skal kunne like eller dislike. Brugeren skal kunne se fulde profiler for potentielle matches både før og efter der klikkes like eller dislike. Brugeren skal modtage en notifikation, hvis man har et nyt match. Desuden

skal brugeren have retten til at fjerne et match igen, hvis der fortrydes. Brugeren skal ligeledes kunne tilgå en liste over allerede eksisterende matches. Brugeren skal kunne opdatere sin egen profil, herunder have muligheden for at slette sin bruger. Brugeren skal til slut have muligheden for at logge ud af programmet.

2.Arkitektur, design & analyse

Et vigtigt aspekt for at komme godt i gang med implementeringen af et nyt system er et grundigt forarbejde. I nedenstående afsnit tages der udgangspunkt i arkitektur, design og analyse. Først fastlægges metoderne, hvorpå der i sådan en implementeringsproces optimalt burde tages udgangspunkt i. Her forklares vigtigheden for Three Tier modellen og MVC-datastrukturen for netop denne eksamensopgave. I analysedelen dannes der et overblik over, hvad programmets funktionelt skal kunne udrette. Ud fra ovenstående kravsspecifikationer uddybes disse indholdsmæssige krav, som programmet skal indeholde. Dette gøres b.la. igennem use-cases, der tydeliggør mulige interaktioner mellem bruger og programmet. Efterfølgende vises et konstrueret UML-klassediagram, hvis formål er at beskrive programmets klasser, hvor det vil vise sig at UML-diagrammet krævede forskellige versioner før jeg nåede frem til en optimal præsentation og inddeling af klasser.

2.1 MVC[1]

MVC står for Model-View-Controller og er et designmønster, der bruges til at organisere den overordnede kodekonstruktion. MVC står for Model-View-Controller. Et designmønster, der bruges til at strukturere ens kode, samtidig med at det gør den mere læsbar og overskuelig. Det er et populært værktøj, og det mest brugte design pattern/framework til webapplikationer, fordi det er nemmere at spore og løse problemer, når det er opdelt i funktionalitet, samtidig med det sørger for ændringer ikke kommer til at påvirke de andre komponenter i de andre mapper.
En anden metode end MVC kunne f.eks. være at udarbejde hele koden i et samlet dokument. Dette kan dog risikere at medføre, at programmøren har svært ved at finde overblik i projektet.
Den første del, Model, er den del, der indeholder de centrale komponenter. Her findes de dynamiske komponenter af applikationens datastruktur. Her ligger ansvaret for datamanagement, altså hvordan den data vi får ind skal ændres, opdateres og ”manipuleres” med, samt kommunikere med vores controllers. Model er alle mine objekter med forskellige variabler.
Den næste del, View, er min frontend, som har ansvar for præsentation, altså det der bliver set. Her ses alle visuelle repræsentationer af data, diagrammer, tables, charts, html osv. Handlingerne fra controlleren ”invokes” her.
Den sidste del, Controller, er ansvarlig for inputs, hvis opgave er at acceptere disse inputs og konvertere dem til kommandoer til Viewet, men oftest til modellen. Denne del bliver altså brugt til at kontrollere det funktionelle altså ”logikken” i kodebasen. Controlleren importerer modeller, og bruger dertil sin funktionalitet til at slette og udfylde data. Fordi jeg bruger en MVC-datastruktur laver mine Controllers requests og responses til de forskellige Models.

[1] Rapporten godkendelsesopgave 3

2.2 Three Tier Modellen & Datastrukturen

Three Tier Modellen er et client-server software arkitektur-mønster. Ideen med modellen er at inddele implementering af et system i tre forskellige tiers. Disse tiers kan opdrages eller udskiftes selvstændigt, da de kører på tre forskellige fysiske instanser. De agerer altså uafhængigt og selvstændigt, hvilket gør det muligt, at gå ind og lave ændringer i det specifikke tier, hvis der opstår et problem f.eks. data tier er for langsomt eller ikke stort nok.
De tre tiers indeholder som det første et presentation tier, som præsenterer front-end, der altid er rendered af browseren. Browseren tager HTML, CSS osv., og laver det om til en statisk eller dynamisk hjemmeside. Med andre ord, hvad der vises på klientens desktop.
Andet tier er logic tier, som er applikationsserveren. Den centrale server, der vil kommunikere ned til de forskellige presentations tiers. I dette tilfælde bruges node.js med express som middleware.
Det sidste lag er data tier, som er databaseserveren. Dette er storage delen, hvor simpel fil storage, local storage osv. er koblet op. I dette tilfælde local storage og server-side storage. Dette uddybes i afsnit 3.4.
De tre dele er forbundet og trækker data fra hinanden, hvilket giver brugeren mulighed for at interagere med andre klienter, som sender en forespørgsel til serveren, dertil henter og opdatere data i databasen. Hvordan datastrukturen er sat op, uddyber jeg under afsnittet ”Konstruktion & Implementering”.

2.2.1API & Node JS

Jeg har valgt at benytte Node.js for at kunne konstruere det, der i ovenstående kaldes logic tier. Jeg har valgt Node.js pga. dens asynkrone udførelse. Node.js er det man kalder enkelt trådet, hvilket betyder at systemet ikke bruger mange ressourcer på at køre flere funktioner på samme applikationsserver. Desuden indeholder Node muligheden for at have en vandret skalering, netop fordi den både kan skalere på en enkelt eller flere desktops.
Den asynkrone kode giver brugeren af systemet et væsentlig hurtigere respons. Når koden er asynkron, betyder det at den har evnen til at køre flere ting samtidig. Altså at systemet ikke venter på den første ”proces” er kørt færdig inden den starter på den næste i rækken. Dette gøres der b.la. brug af, når brugeren ønsker at se fuld liste over alle sine samlede matches. Hvis koden ikke var asynkron, ville tingene i stedet blive vist trinvist, som de blev kørt af systemet.

Desuden er Node.js et effektivt værktøj til at opdele kode mellem browseren og serveren, fordi jeg i denne implementering både gør brug af: JSON, Express og Node js, hvor express optræder som middleware, dvs. en bro mellem databasen og applikationen. Disse gør det muligt at kommunikere og anvende en database til systemet.

2.3 UML-diagrammer som redskab

Det allerførste, der blev udarbejdet i dette eksamensprojekt, var en fremstilling af diverse UML-diagrammer herunder klassediagrammer og use-case diagrammer. Disse udarbejdes efter ovenstående kravsspecifikationer.

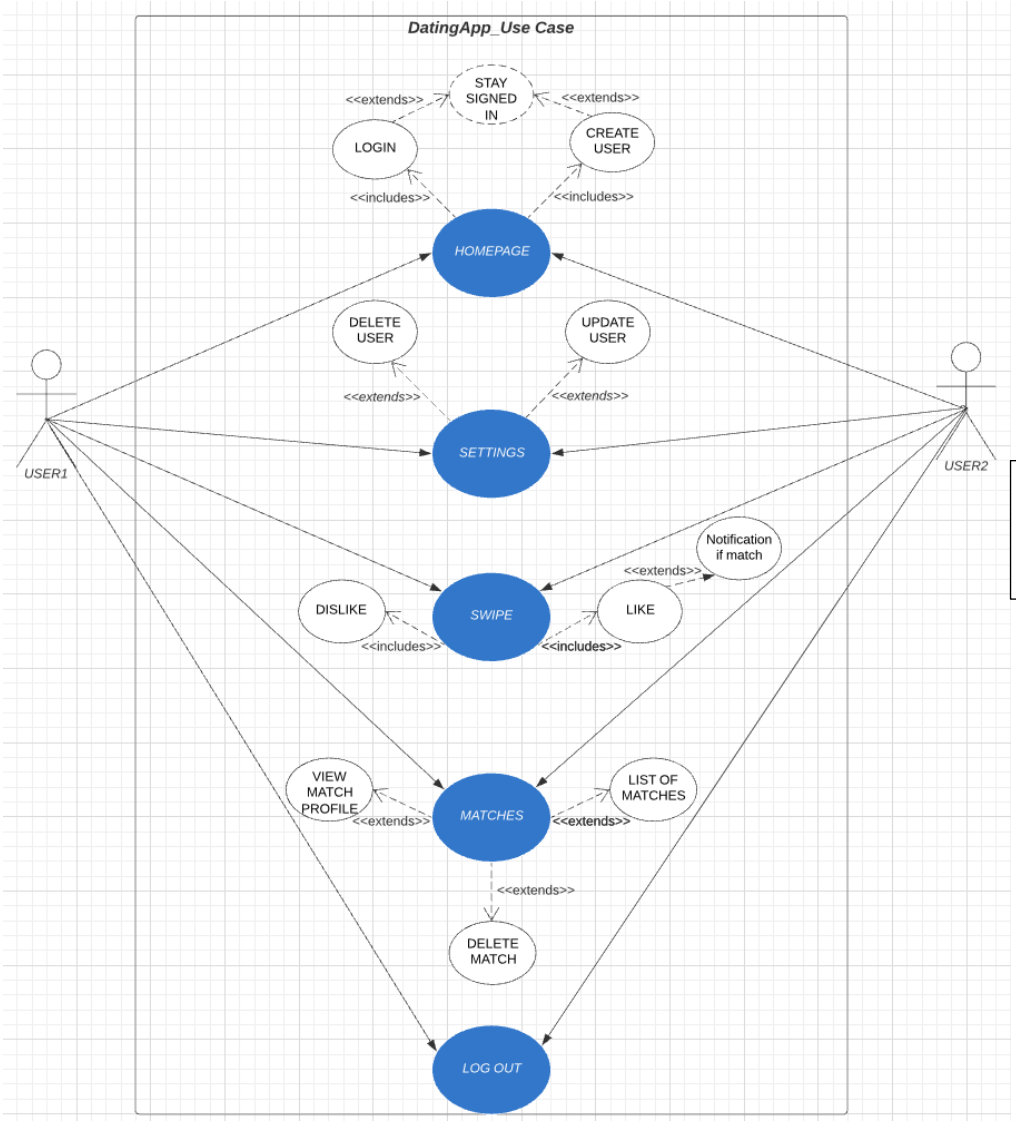
UML-diagrammernes formål er at formulere en måde, hvor på et programs arkitektur kan blive visualiseret. Man kan altså sige, at UML-notation er derfor et slags sprog, der bruges til at beskrive struktur og sammensætning af forskellige komponenter. Med andre ord tillader UML mig at specificere min programstruktur, og hjælper mig dermed at visualisere. De giver mig en template at bygge systemet efter, og hjælper mig med at forstå et kompleks system del for del. UML-diagrammerne klargør og fastlægger de mål, der arbejdes hen imod under implementeringen af dating-systemet, samt skaber overblik over hele processen og agere som dokumentation for vores system
Store komplekse applikationer bliver bygget af mange mennesker på tværs af organisationer, derfor er kommunikation vigtigt. Oftest vil end-users ikke forstå koden, forretningsfolk forstår oftest ikke koden, der kan altså spares meget tid, da UML giver et "high level" overblik over applikationen også for folk, der ikke forstår kodning.
Man skelner ligeledes mellem to typer af UML-diagrammer, der er knyttet til objektorienteret programmering. Strukturelle diagrammer, der bruges til at indkapsle statiske strukturer i et system, altså de fysiske og konceptuelle elementer som f.eks. class, der er et sæt af objekter, som har ligeligt ansvar, eller interface set af operationer, som specificerer en klasses ansvar. Disse statiske strukturer beskrives f.eks. i klassediagram, der beskriver kode strukturen. Det andet er adfærdsdiagrammer, der bruges til at beskrive dynamiske strukturer i systemer f.eks. er interaktion en opførsel, der består af en besked eller en gruppe af beskeder, som bliver udvekslet mellem elementer, når en handling er udført. Disse ses f.eks. i use-case diagrammer eller state-diagrammer, hvor states kan opdateres dynamisk.

Klassediagrammer er som nævnt i ovenstående et strukturelt diagram. Det er således et rebskab, der går ind og illustrerer, hvordan et systems klasser ser ud med metoder attributter, og hvad deres relationer er. De visualiserer, beskriver og dokumenterer. Fordelen er klart, at de kan oversættes direkte til kode. Desuden beskriver de begrænsninger, sammenslutninger, interfaces, kollektion af klasser og associeringer.
Formålet er analyse og design af den statiske del af systemet, og beskrive ansvaret af et system i forhold til dets komponenter. Med andre ord hjælper de med at designe selve systemet, herunder giver et overblik over, hvordan funktioner og krav skal håndteres i systemer, da klasserne i et klassediagram hver især er en slags skabelon for de objekterne, der skal fremgå i systemet.

Use-case diagrammet giver derimod en forståelse for, hvilke forhold/relationer, der er mellem mulige brugere og deres handlinger. Fordelen ved use-case diagrammet er, at det danner et overblik over kravene for systemet og over systemet set udefra. Det identificerer de eksterne og interne faktorer, som spiller en rolle i systemet, og dermed viser interaktion og kravene mellem komponenter og aktører. Use-case diagrammet går altså derfor ind og beskriver, hvordan en bruger interagerer med et system, hvilket giver en klare forståelse for, hvordan systemet skal designes og indrettes på den mest optimale måde ift. denne interageren. Dvs. et "high-level" diagram, som danner overblik over de forskellige scenarier og muligheder en bruger kan gøre brug af i systemet.

Da kravene til denne udarbejdelse af en dating-app er relativ indviklet og kompleks med mange funktioner, objekter og mange både store og små implementeringer, er det fordelagtigt at starte med konstrueringen af UML-diagrammer. Som nævnt skaber de klare målsætninger, og hjælper således med

at skabe klare rammer og forenkle selve processen. Derudover sætter de klasser og objekter i mindre kasser, der gør det mere overskueligt og simplificeret, så en udefra kommende også ville kunne forstå essensen af programmet blot ved at kigge på disse diagrammer. UML-diagrammerne hjælper altså med at visualisere de usynlige dele af programmet, netop fordi funktionerne i et sådant program kun kan anvendes, men altså ikke ses synligt af brugeren. Netop vigtigheden af dette uddybes mere under punkt 8.procesevaluering.
Udviklingen af klassediagrammerne og use-cases uddybes nedenstående.[2]



Figur 1. – use-case diagram

2.3.1 Use-cases

Ovenstående UML-diagram viser mit use-case diagram for denne implementering af en dating-app. Diagrammet viser, hvordan aktører kan finde rundt på brugerplatformen og interagere med systemet. I nedenstående tabel er der kreeret en oversigt ud fra det udarbejdede use-case diagram. Tabellen indeholder derfor de mulige interaktioner for brugerne.

[2] Afsnittet om UML-diagrammer er skrevet med udgangspunkt i forlæsningsvideoen i uge 00000101

USE-CASES	**Muligt scenarie**
USE-CASE 1	En aktør ankommer til startsiden. Aktøren er allerede oprettet som bruger og logger derfor ind med sine oplysninger. Brugeren har muligheden for at forblive logget ind.
USE-CASE 2	En aktør ankommer til startsiden. Aktøren er ikke tidligere oprettet i vores system, og tilmelder sig derefter som bruger. Brugeren har muligheden for at forblive logget ind.
USE-CASE 3	Brugeren er logget ind og ønsker at ændre i sine tidligere indtastede brugeroplysninger og klikker på ”Din profil”.
USE-CASE 4	Brugeren er logget ind, og ønsker at slette sig som bruger fra dating-appen. Brugeren klikker på ”Din profil” og klikker på ”Slet bruger”.
USE-CASE 5	Brugeren er logget ind for at finde nye matches. Brugeren klikker for at like et potentielt match og får derefter en notifikation om, at de to brugere nu er et match.
USE-CASE 6	Brugeren er logget ind for at finde nye matches. Brugeren swiper forkert og matcher med uønsket aktør. Brugeren vil nu gerne fjerne dette match igen.
USE-CASE 7	Brugeren har swipet nok for i dag og logger ud af sin bruger ved at klikke ”Din profil” også klikke ”Log af”.

De nedenstående tabeller er en uddybelse af udvalgte use-cases. Jeg har valgt at uddybe use-case 1,2 og 6 fra ovenstående oversigt over alle use-cases. Jeg gennemgår oplevelsen af dating-appen fra brugerens synsvinkel samt processerne i systemet.

USE CASE NR.	Use-case 1: Oprettelse af bruger på dating-appen
AKTØRER	Bruger
BESKRIVELSE	Brugeren klikker på ”Opret bruger” og udfylder derefter felterne, for at oprette sig selv som bruger i systemet.
PRE-BETINGELSER	Brugeren er ikke tidligere oprettet i vores system
POST-BETINGELSER	Kodeord skal indeholde mindst 6 karakterer, 1 stort bogstav og mindst 1 tal. E-mail og password bliver gemt i databasen, og passwordet bliver hashet af bccrypt. Brugeren forbliver logget ind vha. local storage.
MULIGT SCENARIE	En mulig bruger ankommer til startsiden og beslutter sig for gerne at ville oprette en bruger på datingplatformen. Brugeren klikker på ”Opret bruger” og udfylder derefter felterne. Brugeren klikker ”Opret” og bliver sendt til login siden. Brugeren logger ind og bliver sendt til sin profil.
ALTERNATIVT SCENARIE	En bruger forsøger at logge ind i databasen uden at være oprettet som bruger før.

UNDTAGELSER	Fejlmeddelelse: - Brugeren indtaster ikke et password. Ej heller et password med overensstemmelse til kravene. - Brugeren skriver ikke en e-mailadresse. - Brugeren gentagelse af password stemmer ikke overens. - Brugernavnet findes allerede i databasen.

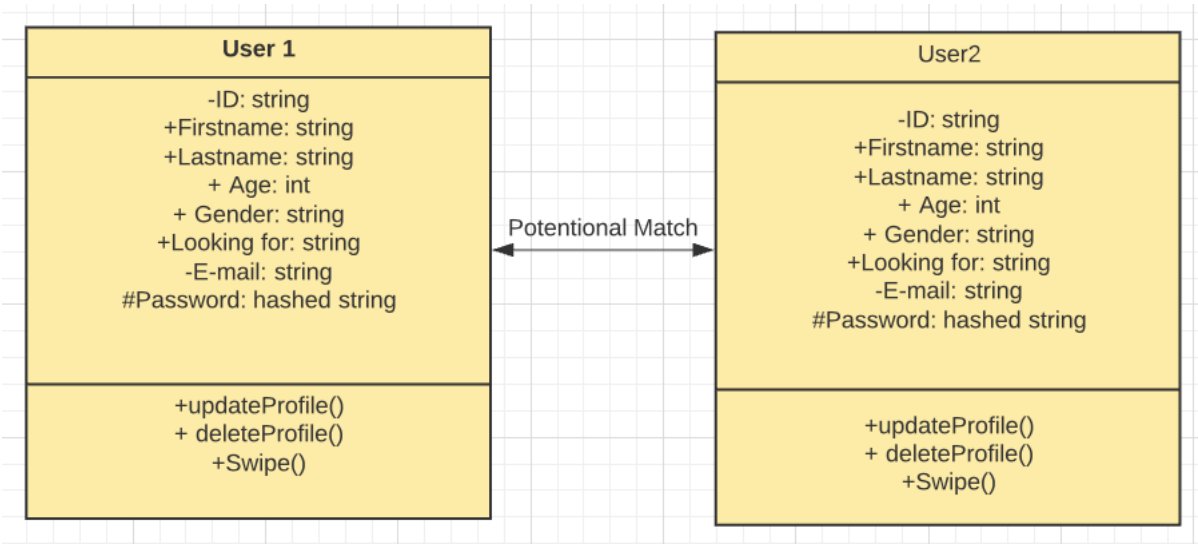
USE CASE NR.	Use-case 2: Login i systemet
AKTØRER	Bruger
BESKRIVELSE	Brugeren ankommer til startsiden. Bruger trykker på 'Login', hvorefter denne vil blive omdirigeret til deres respektive loginside. Her kan brugeren indtaste sin e-mail og password, hvorefter denne kan logge sig ind i systemet.
PRE-BETINGELSER	Brugeren skal tidligere være oprettet i systemet.
POST-BETINGELSER	E-mailen bliver søgt efter i databasen, sammenlignes og holdes op mod passwordet.
MULIGT SCENARIE	En tidligere bruger ankommer til startsiden og logger ind med sine oplysninger.
ALTERNATIVT SCENARIE	%
UNDTAGELSER	Fejlmeddelelse: - Brugeren indtaster ikke et password - Brugeren skriver ikke et e-mail - Brugernavnet ikke findes i databasen - Brugernavnet og passwordet ikke stemmer overens.

USE CASE NR.	Use-case 6: Fjern et match igen
AKTØRER	Bruger
BESKRIVELSE	Brugeren klikker for forkert og matcher med en uønsket profil. Brugeren ønsker at slette matchet igen.
PRE-BETINGELSER	Brugerne skal være oprettet i systemet. Matchet skal også have liket brugeren.
POST-BETINGELSER	Loopet køres igennem for at finde Id'et i arrayet for matchet, som der ønskes fjernet.
MULIGT SCENARIE	Brugeren ankommer til startsiden og starter med at like og dislike. Brugeren liker ved en fejl ten uønsket bruger – de bliver et match. Brugeren

	ønsker at slette matchet igen. Brugeren klikker på matchet og klikker "fjern match".
ALTERNATIVT SCENARIE	Brugeren klikker like og får et ønsket match og beholder matchet. Matchet ligger nu under listen over eksisterende matches.
UNDTAGELSER	Fejlmeddelelse: - Brugerens Id kan ikke findes i arrayet over matches.

2.3.2 Klassediagram

I nedenstående figur ses mit klassediagram, der er udarbejdet i denne implementering af dating-appen, som danner rammerne for, hvordan strukturen i programmet er opbygget. Diagrammet viser to klasser af mulige brugere, og hvad deres relation til hinanden er. Der er kun etableret to klasser, da det er minimumskravet for antal af brugere, hvis appen skal kunne opfylde alle kravspecifikationerne dvs. b.la. kravet om at kunne like eller dislike en anden bruger.
De etablerede klassers, attributter og metoder vises i bunden af klassen. User indeholder ID, first-name, lastname, age, gender, looking for (hvilket køn søger man), e-mail & password.
Dette klassediagram kunne udvides, hvis man vælger at videreudvikle programmet. Dette kommer jeg tilbage til under punktet perspektivering.



Figur 2. – klassediagram

Delkonklusion

Dette afsnit redegjorde med udgangspunkt i implementering af en objektorienteret app, hvordan strukturen er udført i denne udvikling af et delprodukt for denne dating-app.
Dette gjordes først ved hjælp af en specificering af kravspecifikationer til at fastlægge, hvad programmet skulle indeholde, og dertil UML-diagrammer til at illustrere programmets funktioner i forhold til brugeren, samt deres interaktioner med hinanden. Dertil blev der klargjort, hvilke mulige scenarier og problemer, jeg kunne løbe ind i under implementeringen. Der blev altså analyseret på, hvordan

programmets opbygning og struktur skulle udføres optimalt. Betydningen for dette afsnit uddybes senere hen under punkt 8. Procesevaluering.

4.Konstruktion & implementering

I den næste del af rapporten tages der udgangspunkt i pointerne fra ovenstående redskaber og værktøjer, og forklares hvordan jeg har fundet frem til at implementere og konstruere systemet mest optimalt på. De væsentligste dele og elementer af systemet bliver forklaret uddybende, og en diskussion for hvad der ligger til grund for måden at implementere disse metoder, funktioner og elementer vil blive diskuteret løbende nedenfor.

4.1 Presentation Tier // Simpelt Brugerinterface

Til at designe og implementere et simpelt brugerinterface bruges HTML og CSS til at opbygge en frontend, der indeholder kravspecifikationerne mulighed for at indtræde. Jeg har valgt at opbygge min frontend/HTML-delen på EJS-filer, som er en view engine i Express, der i interaktion med JavaScript gør det muligt at inspicere og interagere med HTML-strukturen.
Express er som tidligere nævnt min middleware, der er broen mellem de tre tiers. Jeg gør altså brug af det, der kaldes server-side rendering, der er en metode, hvor man kan skrive en form for scripts ind i HTML-delen og dertil indsætte dynamisk data fra backend.
Jeg har valgt at opdele presentation tier i flere underdokumenter for at holde overblikket f.eks. i header mappen har jeg implementeret de filer, som jeg sætter i HTML <meta> tags. Desuden er der oprettet en main.css fil, der indeholder det CSS, jeg ønsker at have i alle HTML-filerne. En smart shortcut er at linke mainMeta.ejs til main.css filen, så jeg inde i mine ejs filer kan inkludere denne mainMeta.ejs, så behøver jeg ikke at linke til min main.css fil hver gang jeg har at gøre med en ny HTML side.

Grundet enormt tidspres har jeg valgt at prioritere de to sidst nævnte tiers nemlig logic tier og database tier, og dermed nedprioriteret selve designet af HTML-filerne, hvilket tydeligt kan se ses på hjemmesidens interface. Den har tydelige mangler både brugervenlighedsmæssigt og på styling siden, da fokusset altså har ligget på at få programmets funktioner til at virke optimalt. Med andre ord trænger frontend til en kærlig hånd, hvis dating-appen skal videreudvikles, da brugerinterface er en kæmpe faktor for at tiltrække nye brugere til programmet, særligt da der er stor konkurrence på datingapp området i forvejen. Dette ville sågar gå ind og blive en forretningsmæssig afgrænsning.

4.2 Logic Tier // Server

Logic tier er som tidligere nævnt den logiske del af systemet, der indeholder de såkaldte controllers. Serveren er i denne implementeringsopgave udformet som et API i framework Express i Node.js, hvor Express som tidligere nævnt fungerer som middleware, altså en såkaldt bro eller bindeled mellem de forskellige lag i systemet. Systemet indeholder to API'er. Det første API er web.js, som varetager funktionerne, som afsender HTML-filer til brugeren. Det næste API er api.js, der har til opgave at holde styr på databasens input, dvs. gemme og læse data fra databasen. De to API'er taler sammen

igennem brugerens interaktioner på hjemmesiden. De kommunikerer ved, at web.js sender den HTML side som klienten ønsker at hente, hvor api.js modtager brugerens request's igennem en form, hvorefter api.js, sender brugeren tilbage til den side, som web.js varetager.
Mappen med controllers er kaldt routes og indeholder de to API'er. Disse kunne jeg godt have haft ude i stam mappen sammen med server.js, men jeg har valgt at holde dem adskilt for at holde vores mapper og filer overskuelige. Jeg har desuden valgt at opdele det i separate mapper, så filerne ikke bliver uoverskueligt lange.
Dataudvekslingsformatet er JSON-struktur – dennes betydning vender jeg tilbage til under nedenstående punkt, database tier // locale storage & server-side storage.

4.3 Database Tier // Local Storage & Server-side Storage

Undervejs når en bruger indsætter sine oplysninger og til- og fravælger options på platformen, skal dette kunne huskes og gemmes i databasen. Til dette formål er der gjort brug af server-side storage og local storage. Server-side opbevaring bruges i delproduktet til at gemme brugers indtastede oplysninger, men hvis vigtigste formål er at validere en bruger. I dette system er valideringen meget simpel, netop fordi brugeren "bare" skal oprette sig før den kan logge ind i systemet, og dertil kommer at ingen har det samme fornavn eller efternavn som dem.
Denne server-side storage kan ses under mappen database i filen users.json. Nedenstående kodestykke viser til venstre, hvordan en brugers oplysninger gemmes i databasen ved oprettelse og kodestykket til højre igen slettes fra databasen, hvis brugeren deaktiveres/sletter sin profil.

```
[
  {
    "id": "77043a79-fa02-486f-a970-341130ceaade",
    "firstName": "Lucas",
    "lastName": "Gransbak",
    "email": "Lucasgransbak@outmail.dk",
    "birthDay": "04/09-1997",
    "sex": "mand",
    "lookingFor": "kvinde",
    "password": "$2b$10$L.tF7Bkcno6nGA.asaydieLXLwJ3jt0pWg7YnXzSW792c/Ezb3XH
    "matches": [],
    "likes": [],
    "dislikes": []
  },
  {
    "id": "255f460b-5e96-4204-a787-4c0e959689b4",
    "firstName": "Sofie",
    "lastName": "Knudsen",
    "email": "SofieKnud@gmail.com",
    "birthDay": "03/03-1999",
    "sex": "kvinde",
    "lookingFor": "mand",
    "password": "$2b$10$0LL7E2DIYkj7Y.Nu.MjBjuiutWrak9qwuQQuFEHSqvmDolyee9Pr
    "matches": [],
    "likes": [],
    "dislikes": []
  }
]
```

Kodestykke1

database > {} users.json > ...

```
[
  {
    "id": "77043a79-fa02-486f-a970-341130ceaade",
    "firstName": "Lucas",
    "lastName": "Gransbak",
    "email": "Lucasgransbak@outmail.dk",
    "birthDay": "04/09-1997",
    "sex": "mand",
    "lookingFor": "kvinde",
    "password": "$2b$10$L.tF7Bkcno6nGA.asaydieLXLwJ3jt0pWg7YnXzSW792c/Ezb3XHq",
    "matches": [],
    "likes": [],
    "dislikes": []
  }
]
```

Kodestykke 2

Databasen er opbygget af JSON filer, da dette system kun er et delprodukt af en dating-app. Det betyder, at det er meget nemmere at arbejde med samt teste og videreudvikle for mulige videreudviklere af appen, netop fordi JSON-filer er en tekstrepræsentation af et objekt og indeholder, derfor ikke metoder elle funktioner.
Ude til højre ses skabelonen for users.json, dvs. hvordan alle brugerens profiler ser ud, og er beskrevet i vores database. Alle brugere, der tilmelder sig hentes ind i dette array. Det skal noteres, at databasen gemmer brugerens password, men at koden er hashet vha. bcrypt. Dvs., at databasen ikke gemmer adgangskoden direkte og er synlig for enhver programmør, der sidder med koden, men bliver gemt som f.eks. "723yehieukhfy7fiwkhdf7we3f" og ikke f.eks. ”Fodbold123”, der er koden uden den blev hashet.

```
{
    "id": "",
    "firstName": "",
    "lastName": "",
    "email": "",
    "birthDay": "",
    "sex": "",
    "lookingFor": "",
    "password": "",
    "matches": [],
    "likes": [],
    "dislikes": []
}
```

Skabelon for users.json

Local storage er det stik modsatte af server side-storage, da det er opbevaring af data på browseren. Local storage er dog ikke specielt anerkendt pga. manglende sikkerhedsforanstaltninger, da man lettere kan få adgang til den følsomme data. Local storage går nemlig ind og gemmer lokalt på computeren i ubegrænset tid og slettes som udgangspunkt ikke - medmindre brugeren går direkte ind og sletter sig som bruger i programmet, eller man decideret beder JavaScript om at slette indholdet i local Storage. Jeg bruger b.la. local storage ved login, så brugerens desktop husker e-mail og password og til at forblive logget ind på siden. Jeg har valgt at bruge local storage, fordi det er yderst relevant, at brugerens oplysninger skal gemmes permanent, da det er en app, hvor man har sin personlig profil, og derfor kræver at app’en skal kunne huske de brugere, der højst sandsynligt bruger appen dagligt. Det er desuden relevant at anvende local storage, hvis app’en engang skulle videreudvikles til både at indeholde betalende bruger og gratis brugere. I dette tilfælde vil app’en gå ind og gemme den betalende brugers kortoplysninger, så den måned for måned ved hvilken konto beløbet skal trækkes fra. Hvorimod, hvis det blot var en JustEat-app af en slags ville den locale storage ikke være relevant, men det ville session storage derimod. Her ville indholdet i session Storage blive slettet, når vinduet lukkes ned igen og browsersessionen dermed afsluttes. Dog ville dette kræve en større sikkerhed i storage, når det har med betaling og opbevaring af fortrolige oplysninger at gøre, netop som opbevaringen af kortoplysninger er. Desuden er det også en forretningsmæssig begrænsning, da der ikke er nogen validering af e-mailadressen, dvs. at vi rent faktisk ikke ved om denne findes. Disse skulle klart prioriteres under en mulig optimering og videreudvikling af programmet.

Et alternativ til local storage kunne som nævnt ovenfor være session storage, hvor når vinduet lukkes ned og browsersessionen afsluttes slettes data fra databasen. Netop af denne grund fravalgte jeg at bruge session storage, da det ikke er en ønskelig egenskab for databasen i denne sammenhæng af systemets opbygning og mål.

4.4 Datastruktur – Array, Loops & if….else statements

Jeg har valgt at bruge arrays, som datastruktur til at gemme, opbevaring og manipulering af data. Array er inden for programmering betegnelsen for en variabel, der indeholder en række af værdier,

associeret med en nøgle. Nøglen er typisk et heltal, men nogle programmeringssprog tillader brugen af strenge og andre datatyper til identificering og sortering.
Arrays anvendes altså til at gemme lister af variabler samt deres værdier. Man kan så efterfølgende trække disse værdier ud af sit array ved at anvende og specificere index nr. for den værdi man ønsker at finde[3].

Arrays gøres der blandt andet brug af i implementering af kravet om en liste over tidligere matches. Her gemmes, opbevares og struktureres alle de Id'er på brugere, som profilen har matchet med. Det er et array, fordi vi skal gemme mere end en værdi i dette profil objekt. Fordelen ved at anvende arrays er, at vi kan udnytte deres mutabilitet[4], hvorved vi kan gøre brug af array-metoder til at eksempelvis at tilføje nye elementer til arrayet, eller fjerne dem igen. At tilføje nye elementer til arrayet bliver brugt, når der opstår et nyt match. Dette match ryger dertil ind under arrayet for listen over tidligere matches.
Desuden har jeg også gjort brug af arrays i implementeringen af like og dislike funktion, både til at gemme alle de Id'er på brugere som profilen har liket og disliket, men også fordi arrays er super brugbare og nemme at køre loops igennem. I datalogiens verden er loops en programmeringsstruktur, der gentager en række instruktioner, indtil en bestemt betingelse er opfyldt, hvor de b.la. bliver brugt til "at bladre gennem" værdier, tilføje numre af tal, gentage funktioner og mange andre ting. Loops har jeg også brugt til kravet og rettigheden for en bruger at slette et match igen.

```
450   // Slette brugeren fra personens matches og likes
451   if (matchUser.matches.includes(loggedInUserObj.id) === true) {
452
453     // Looper igennem brugerens array med matches og finder det id på den person vi vil fjerne
454     for (let i = 0; i < matchUser.matches.length; i++) {
455       // Hvis den plads i arrayen er = personens id
456       if (matchUser.matches[i] === loggedInUserObj.id) {
457         // "i" er den plads den skal slette, og i + 1 for at sige den skal stoppe ved en større end i - Den sletter ikke den selve pladsen
458         matchUser.matches.splice(i,i + 1);
459       }
460     }
461     // Her kører vi igennem brugerens likes for også at fjerne den person der fra.
462     for (let i = 0; i < matchUser.likes.length; i++) {
463       // Hvis den plads i arrayen er ligmed personens id
464       if (matchUser.likes[i] === loggedInUserObj.id) {
465         // "i" er den plads den skal slette, og i + 1 for at sige den skal stoppe ved en større end i - Den sletter ikke den selve pladsen
466         matchUser.likes.splice(i,i + 1);
467       }
468     }
469
470   } else { // Hvis vi ikke finder personen i matches ej heller i likes
471     message = "Du blev ikke fundet i " + matchUser.firstName + " matches"
472   }
473
474   message = "Du har nu fjeret " + matchUser.firstName + " fra dine matches";
```

Kodestykke 3

Metoden med loops ses i ovenstående kodestykke 3, hvor den bruges for deleteMatchUserId(routes-api-417). Kodestykket viser et for loop, der køres for at finde det specifikke Id på den bruger, som der ønskes at blive fjernet fra matches. Med en if……else statement undersøges hver bruger, om deres id passer sammen med det specifikke "brugerId", der ønskes fjernet. Når loopet kommer til denne bruger i arrayet, hvis Id stemmer overens, anvendes metoden splice til at fjerne den pågældende

[3] https://www.nemprogrammering.dk/Tutorials/c-sharp/21-arrays.php
[4] https://medium.com/swlh/javascript-array-mutability-immutability-93d366c90751

bruger fra matches. Else statementet beskriver, hvis personen ikke blev fundet i hverken arrayet med listen over matches og ej heller i arrayet med tidligere likes.
Local storage anvendes nu til at kunne gemme vores opdateringerne arrays, så data kan tilgås senere hen. Fordi local storage kun har mulighed for at opbevare elementer af typen "string" udføres der til allersidst en serialisering, der konverterer til sådan et format. Der gøres nu brug af konverteringsformatet JSON. JSON er som tidligere nævnt en tekstrepræsentation af et objekt, som man lettere vil kunne sende over netværk. Man kan lave et objekt om til JSON ved at bruge funktionen JSON.stringify til at kunne gemme det opdaterede array i local storage. Hvis disse elementer senere skal tilgås, bruges JSON.parse(). Nedenstående kodestykke viser, hvordan dette er blevet brugt i praksis.

```
475
476     // Så laver vi filen tilbage til rå data
477     var rawData = JSON.stringify(dbUsers, null, 2);
478     // Så gemmer vi de nye ændreing
479     fs.writeFile(dbUserPath, rawData, function (err) {
480       if(err) throw err;
481     });
482
483     res.status(200).json({ status: "OK", message: message });
484     res.end();
485   });
```

Kodestykke 4

Der findes mange forskellige alternativer til arrays, som både har fordele og ulemper sammenlignet med arrays som datastruktur. F.eks. kunne man have anvendt Maps, som et muligt alternativt. Maps gør lige det ligesom arrays muligt at gemme elementer knyttet til en bestemt nøgle. Denne egenskab ville det gøre nemt at tilføje elementer til et Map.
Desuden indebærer maps en datastruktur, der hurtigt kan lede igennem og manipulere et stort map med mange værdier. Jeg fravalgte maps har to årsager: for det første føler jeg mig bedre hjemme i håndteringen af arrays, da vi har brugt det løbende på semesteret, hvor maps ville være en ny metode for mig at tage i brug. For det andet har Maps den kæmpe ulempe, at de modsat arrays ikke lige så nemt og hurtigt kan omformateres til en string, så local storage forstår vores opdatering.

Delkonklusion

I ovenstående afsnit er datastrukturen blevet uddybet samt udvalgte dele af de metoder og disses fordele og ulemper, der er blevet anvendt i udarbejdelsen af programmet. F.eks. er der blevet forklaret, hvordan brugen af loops og if….elses statements er en kæmpe del at opbygningen af koden bag programmet. Samtidig hvordan arrays som datastruktur går ind og spiller en afgørende rolle b.la. i forhold til databasens opbygning. Dertil er det blevet udpeget, hvorfor valget er faldet på at anvende arrays som datastruktur fremfor maps eksempelvis, fordi arrays fordelagtigt kan omformateres til JSON-format, så local storage forstår indholdet. Desuden har jeg prøvet på bedst muligvis at diskutere alternative løsningsmuligheder på disse metoder og finde mulige måder at optimere programmets funktionalitet i fremtiden.

5. Test

En vigtig del af udviklingen af et program er testning både løbende tests, men særligt en afsluttende test. Formålet med dette er at finde ud af om programmet virker optimalt og dertil inspicere programmets virken og indfange de fejl, der nu end måtte forekomme. For at undersøge og teste programmet

er der blevet anvendt forskellige testing- og error handling metoder. Jeg har så vidt muligt forsøgt, at lave test parallelt med implementeringen af mit program, for løbende at finde fejl og bugs, der evt. stod i vejen for kodens funktionalitet. Nedenstående vil en beskrivelse af disse test-metoder blive gennemgået.

5.1 Error handling

Error handlinger har klare og tydelige formål, nemlig at opfange, forhindre og forebygge mulige fejl. Dette gøres indenfor to felter: de fejl, der er forårsaget af programmøren selv, og fejl der er forårsaget af en mulig bruger af programmet. At opspore fejlene, der er forårsaget af programmøren har til formål at finde frem til den specifikke fejl i koden, og få disse fejl løst og optimeret så programmet kører som planlagt. Hvor at finde fejlen forårsaget af brugeren, bunder i, hvordan og hvornår brugerens interaktioner med programmet fejler f.eks. at brugeren undlader at udfylde felter, når de opretter sig som bruger, eller at den e-mail de skriver, faktisk ikke er en e-mail. Den sidst nævnte type fejl, fejl hos brugeren, kan undgås ved at opbygge ”hjælpefunktioner” i koden, som tjekker for forskellige typer af fejl f.eks. som den indtastede e-mail indeholder et @, hvorefter f.eks. at give brugeren en notifikation, der fortæller, hvor fejlen er opstået.

5.1.1 Unit-testing

Til testing af programmets funktioner er der gjort flittigt brug af unit-testing. Unit-testing er en metode indenfor testing, hvor man går ind og tester en unit dvs. enhed af gangen. Units kan f.eks. være en klasse, en funktion eller et helt modul. Formålet med dette er at isolere og identificere mulige problemer og fejl, og køre dem igennem en selvstændig test case. Man kan evt. kombinere tests for at dække større units.

5.1.1.1 Postman

Til unit-testing har jeg gjort brug af Postman. Postman er det der kaldes test harnesses, dvs. et eksternt stykke software, der hjælper med at teste units. Postman kan bruges til mere end blot testing, men formålet med at anvende Postman i mit tilfælde, har været Postmans egenskab til at kunne unit teste dele af min programkode. Med andre ord er største delen af min kode altså blevet kørt igennem Postman for at tjekke om deres funktionalitet overhovedet virkede og kørte som ønsket.

5.1.1.2 Console.log()

Flere gange undervejs i udviklingen af programmet har jeg gentagne gange indskrevet en console.log-statement. Denne strategi har jeg valgt at bruge som en del af min unit-testing for at tjekke om enkelte stykker kode kørte og virkede optimalt og som ønsket. Console.log() kører kodestykket igennem i terminalen, og kontrollerer dermed om programmet overhovedet virker og giver besked herefter. Hvis kodens funktion ikke virker som ønsket, vil der i terminalen ikke blive spyttet det ønskede og forventede output ud. Dette betyder blot, at der er en bug i programkoden, som skal ændres for det ønskede resultat, kan opstå. Console.log() har været utrolig nyttigt, hjælpsomt og givet hurtigt svar under testingen af programmets virken.

5.2 Sluttest

5.2.1 Use-case testing

For endeligt at slutteste om programmet indeholder kravspecifikationerne udførtes der til alle sidst i processen use-case tests, der testede hele programmet af herunder de forskellige mulige use-case scenarier. Disse havde til formål at opdage og identificerer fejl og mangler i programmet. Disse fejl og mangler burde afvikles og optimeres for at stå med et tilfredsstillende resultat. En lille, men væsentlig pointe at nævne er, at use-case tests har den ulempe at være mere undersøgende af det funktionelle fremfor objektorienterede, da den går mere ind og vurdere programmets reelle funktioner, og ser bort fra det indvendige nemlig kommunikationen mellem objekter og klasser. Den klare fordel er dog at use-case tests opfanger implikationer fra brugerens synsvinkel og reflekterer over, hvordan det kunne give en endnu bedre brugeroplevelse.

5.2.2 Black box testing

Som en del af den afsluttende testing af programmet har jeg benyttet mig af black box testing. Black box testing bliver udført ved hjælp af en ekstern person. Med andre ord er Black box testing, når testeren ikke ved, hvordan applikationen fungerer ”indeni”. Testeren ved altså ikke noget om arkitekturen, eller hvordan komponenter spiller sammen eller kodebasen. Testpersonen gennemgår de forskellige use-cases, som testpersonen var en mulig bruger af programmet. Ud over at teste use-cases, så giver en black box testing også en tilbagemelding omkring brugervenlighed. Fordelen ved denne type test er, at man hurtigt får en tilbagemelding på, om brugeren rent faktisk formår at bruge de funktioner, som programmet burde være i stand til at understøtte. Ulempen er derimod at de er ressourcekrævende, og typisk er vurderet meget subjektivt i forhold til testerens normale præferencer og generelle meninger. Denne ulempe er dog mulig at mindske ved anvendelse af flere black box brugertests.
Testen viser, at programmet egentlig virker relativt optimalt og kun mindre fejl. Disse fejl var jeg dog allerede opmærksom på forinden. Alligevel har black box testen været en rar måde at få feedback, samt at få svar på ens forestillinger om programmets virken holdt stik.

Use case	Formål	Data	Forventet scenarie	Kommentar	OK/ Virker ikke	Fejlbeskrivelse
Opret profil	Oprette en ny bruger i systemet via. sign-up controlleren.	E-mail: Mille.holm@gmail.dk Password: Buller0912	Brugeren oprettes succesfuldt i systemet og sendes til log ind siden.	Måske at man bliver logget direkte ind og ikke skal en "omvej" hvor man skal logge ind igen.	OK i Chrome	Hjemmesiden kan kun tilgås i Chrome
Log ind	Brugeren logger ind via login controlleren, der sammenligne e-mail og password i vores database. Brugeren forbliver logget ind.	E-mail: Mille.holm@gmail.dk Password: Buller0912	Brugeren bliver logget in dog sendt til profilens forside.	-	Halv	Den logger fint ind, og forbliver logget ind selvom jeg lukker computeren ned.
Liste over matches	Brugeren er logget ind og ønsker at se listen over matches.	Brugeren leder efter muligheden for at se listen over matches	Brugeren klikker på "Dine matches".	-	OK	-
Like / dislike potentielt match	Brugeren disliker og liker brugere for at få nye matches	Brugeren leder efter nye matches og får en notifikation, når de har et nyt match	Brugeren liker en bruger og får en notifikation med et nyt match	Kunne måske gøres lidt mere ud af dette i forhold til notifikationens udseende	OK	
Slette match	Brugeren ønsker at slette et match vha. arraydatastrukturen fjernes et uønsket match	Brugeren leder efter muligheden for at fjerne et match	Brugeren får succesfuldt slettet matchet uden problemer.	Kunne evt. godt gøres anderledes.	OK	
Redigere sin profil	Brugeren ønsker at redigere i tidligere indtastede oplysninger.	Brugeren leder efter muligheden for at opdatere sin profil. Klikker "din profil" og indtaster redigeringen.	Brugeren redigerer sin bruger succesfuldt.	Mulighed for at ændre mere end blot sit navn og efternavn.	OK	
Slette sin profil	Brugeren har mulighed for at slette sin profil vha. deleteprofile controlleren.	Brugeren leder efter muligheden for at slette sin profil.	Brugeren klikker "din profil" og klikker "slet bruger"		OK	

5.3 Optimering

Undervejs i implementeringen har optimering hele tiden stået i fokus at teste ideer og kodestykker, der kunne gøre funktionaliteten af programmet bedre og mere optimalt. Optimeringen af kodens funktionalitet har konstant været i fokus. F.eks. optimering af koden, så den er simpel og ikke indeholder kodestykker, der er unødvendige med manglende funktion i forhold til kravsspecifikationerne. Dertil optimerer kodens eksekverbare tid ved at anvende nye teknikker.

Efter den sidste sluttest står det klart, at programmet stadigvæk mangler nogle væsentlige aspekter og kunne gøre gavn af meget mere optimering, som det også fremgår ovenover. Optimeringen ville både indeholde optimering af funktioner, sikkerhed, brugervenlighed, styling, men også selve måden at håndtere indkommende fejl på med en udvidelse af unit testing og flere kompetancer. Der er altså både en del programmeringsmæssige- og forretningsmæssige afgrænsninger. Hvis koden skulle optimeres i fremtiden, ville jeg sætte fokus på 3 ting: Først optimering af allerede eksisterende kode, så f.eks. at det tillades at åbne app'en i Safari og ikke blot i Chrome, eller at kunne swipe rigtigt og ikke blot klikke for at like og dislike. Dernæst implementerer nye metoder, funktioner og sikkerhedsforanstaltninger, der f.eks. hjælper brugeren, hvis brugeren har glemt sit password eller validerer at den indtastede e-mail eksisterer. Til allersidst mangler appen som tidligere beskrevet en del på frontend og styling siden. Disse punkter ville jeg lægge fokus på i en fremtidig optimering af programmet.

Delkonklusion

Testing har dermed været et vigtigt aspekt i implementeringen af systemet. Afsnittet forklarer, hvor hjælpsomt error handlinger har været på at opfange og løse potentielle programmør og slutbruger fejl. Jeg har i test-afsnittet vist, hvordan jeg gennem error handling kan afhjælpe potentielle brugerfejl med hhv. Derunder hvordan unit testing har været væsentligt til at identificere, hvor fejlen i koden ligger gemt, hvor brugen af Postman som en test harnesses, samt den strategiske console.log testmetode, der er blevet brugt til at forsikre mig om, at programmets små enheder fungerer og dele virker optimalt. De løbende test har haft stor betydning for det endelige resultat af programmet, der opfylder alle kravspecifikationerne med undtagelse af en notifikation om muligheden for en bruger at forblive logget ind. Til allersidst en sluttest i form af black box test, hvor programmet blev testet af en ekstern person, der hjalp med at tydeliggøre manglerne, samt give inspiration til en mulig optimering af programmet.

7. Dokumentation

For selve implementeringsprocessen har det været væsentligt at dokumentere koden undervejs. Dokumentationens formål er at kommunikerer klart og tydeligt, hvad den specifikke kode har af betydning, og hvilke formål den har til grunde. Uden dokumentation mister man hurtigt overblik i mange siders lang kodning. Hjælp til dokumentering er der i denne eksamensopgave blevet brugt værktøjerne GitHunb, Git og kommentering. Disse redskaber har til sammen givet et større overblik og bedre strukturering af koden og dens implementering. For ikke at glemme en bedre forståelse for kodens

enkelte komponenter, men også i det større hele ved at holde styr på forskellige versioner af koderne. I nedenstående afsnit forklares disse værktøjer og brugen af dem.

7.1 GitHub & Git

Før jeg kunne komme i gang med selve opgaven, blev der oprettet et repository på GitHub. GitHub er en webbaseret platform, der opbevarer koder i cloud-based storage. GitHub gør det muligt for flere personer at arbejde på samme projekt, og se hinandens redigereringer i sekundet, de indskrives. Men under dette forløb ønskes brugen af GitHub til at holde styr på de forskellige versioner af mit program. Dertil integreres Git, så disse to platforme kan samarbejde. Git er en lokal version software control for udviklere, hvilket betyder at platformen gemmer forskellige versioner af projektet undervejs i de forskellige faser. Dette gør det nemmere at holde styr på, hvad der i udviklingsprocessen allerede er etableret, og eventuelt vende tilbage til tidligere faser for at redigere i disse, hvilket gør det lettere at rette og løse fejl, der kan opstå under udviklingen. Ved indtastning af "git add" stager man filen/flytter den til staging area, altså koden gemmes i terminalen, derefter "git commit -m ", så det ønskede comittes til den lokale git repository database og til sidst "git push" for endeligt at sende beskeden.[5]

Jeg løb ind i kæmpe problemer med GitHub og Git under implementeringen af mit program. Jeg ville i så fald have lavet 3 forskellige mapper, der skulle hjælpe med at sortere mine forskellige kodedele og testdele. Ved Git branching divergerer man fra "hovedlinjen" (typisk kaldet master) af ens filer. Dette gøres, så man kan lave ændringer, der ikke påvirker den allerede fungerende kode. En branch er altså en pointer, vi kan flytte rundt på, som peger på andre snapshots. Disse branches skulle ligeledes være inddelt i Testbranch, Onebranch og Master. Onebranch skulle indeholde forskellige kode dele. Disse kodedele skulle testes i testbranch for mangler, fejl og optimering. Hvis koden blev godkendt til brug, ville den blive committet til Master. Master bliver altså kun brugt som en placeholder for de færdige og kontrollerede kodedele. Formålet med dette var at teste fejlene løbende og trinvist, samt at holde tingene adskilt og give et større overblik over mangler og færdigheder.
Jeg forstår, hvordan Git bruges og integreres, men jeg havde store problemer med det i min terminal. Min terminal ville overhovedet ikke anerkende mine Git push-beskeder. Jeg formåede dog at lave en masterbranch og developmentbranch, men terminalen ville derfra ikke genkende, at der var kommentarer til stede. I stedet endte jeg med at lave det hele samlet på min computer. Dette blev utroligt rodet og indviklet at arbejde med. Det blev svært at holde styr på, hvad der virkede og hvad der ikke gjorde, samt adskille de ting, der fungerede i processen fra den resterende kode.

7.2 Kommentarer som dokumentation

I programmeringssproget er det muligt at indsætte kommentarer, som ignoreres af computeren, når programmet udføres. Dette kan være hensigtsmæssigt af flere grunde, og måske endda være næsten lige så vigtig som selve koden. Kommentarer i koden skaber klarhed og tydeliggør, hvad intentionen bag koden har været. Dette er brugbart af flere grunde. Oftest arbejder flere mennesker sammen om at udvikle et program. Derfor er det vigtigt, at man supplerer sin kode med kommentarer, så andre

[5] Rapporten godkendelsesopgave 3

forstår hensigten med koden[6]. Det er ikke altid nemt at gennemskue et program, hvis man ikke selv har skrevet det. Ligeledes er det et væsentligt redskab for personen, der programmerer koden, da det er en hjælpende hånd til at holde overblikket, holde tungen lige i munden og forstå de små som større aspekter af koden.
Kommenteringen er i denne eksamensopgave ligeledes blev brugt som et værktøj til at deaktivere enkelte dele af koden. Dette gøres for at teste om enkelt dele af programmet virker optimalt. Udkommentering er en valid måde at teste programmet på. Udkommentering forekommer med en "//" i både begyndelsen og slutningen på, hvad der ønskes udkommenteret, hvis vi skriver i JavaScript. Enkelt linjers kommentarer skrives blot med en "//" i starten, hvorved den efterfølgende tekst ikke indskrives i selve programmet, men blot som en slags note. Kommenteringen er særligt til gavn, hvis man sidder med en kode i en længere periode. De hjælper altså med at huske, hvad ens tidligere code-session har af specifikke funktioner og betydninger.
Eksempel på kommentering:

```
302     // Så hasher vi adgangskoden, også selvom der ikke er skrevet nogle ny adgangskode = vi ikke behøver at have flere if statements end et.
303     // Vi skal have alt koden inde i dette scope, for at alt føler med i bcrypt's tempo og for at vi kan bruge hash callbacked
```

Kodestykke 5

Delkonklusion

Ovenstående afsnit viser altså, hvor relevant og væsentligt det er at inkludere dokumentation i sin implementering og konstruering af et program. Det står klart beskrevet, at dokumentationen undervejs i projektet ikke har været helt som ønsket, og kunne klart forbedres. Brugen af Git og GitHub var langt fra ideelt, og havde jeg formået at få dette til at virke, så havde der klart været et bedre overblik undervejs i procesudviklingen. At fravælge disse to redskaber er langt fra ideelt, men var sådan resultat blev efter for meget bøvl med redskaberne. Dog var anvendelsen af kommentarer en ren succes fra start af, da det gav en meget større forståelse af, hvad der foregik i koden, og hvad de forskellige koder havde til formål. Erfaringen af brugen af dokumentering vil klart blive taget med mig videre.

8. Procesevaluering

Diverse steps til at designe, konstruere, implementere og udvikle dette datingprogram, har ikke været lige nemt hver gang. I afsnittene nedenstående forklares, hvilke problemer jeg har stødt på i udviklingsprocessen, samt hvordan jeg har prøvet at løse disse. Dertil er der en kortere forklaring af, hvad jeg føler, jeg har lært i under udarbejdelsen af eksamensprojektet.

8.1 Problemer undervejs

8.1.1 Axios

Et af de største problemer jeg støtte på under udviklingen af dette system, var da jeg skulle indføre den funktion, der skulle gøre det muligt for brugeren at like eller dislike en forslået bruger. Applikationen blev ved med at genindlæse siden gang på gang, når en bruger klikkede like eller dislike. Det viste sig, at funktionen var blevet udarbejdet i en form, der fik serveren til at genindlæse hver evig eneste gang, der af brugeren blev klikket like eller dislike. Dette problem kom jeg til livs ved at oprette en ny slags form, der brugte Axios. Da jeg først fik installeret Axios, der er en package ligesom

[6] https://programmering.systime.dk/?id=202

express. Axios er et redskab til at lave nemme http request via. JS i client. Denne egenskab fik jeg udnyttet, og jeg fik oprettet to post requests til API'en, som gjorde det muligt for en bruger at like og dislike et potentielt match uden at siden genindlæste.
Et alternativt kunne være at bruge Vanilla JavaScript, der også gør det muligt at sende http reqeusts. Axios er dog nemmere og mere forståeligt at arbejde med, derfor valgte jeg denne til løsning af mit problem.

8.1.2 NPM

I implementeringen har koden skulle NPM pakken bruges. NPM står for node package manager.
NPM er pakkehåndtering for Node JavaScript-platformen. Pakken har det formål at automatisk opdatere de ændringer programmøren har udført og de ting han installeret. NPM sætter moduler på plads, så noden kan finde dem, og styrer afhængighedskonflikter intelligent. Det er ekstremt konfigurerbart til at understøtte en lang række brugssager. Oftest bruges det til at udgive, opdage, installere og udvikle nodeprogrammer[7].
Jeg havde store problemer med at installere NPM. Jeg prøvede gentagne gange at installere pakken i terminalen, men gang på gang spyttede den en error ud. Jeg løste problemet, ved at lukke terminalen ned ved control c, også kørte jeg den igen ved at skrive "npm install", hvor jeg nu i stedet skrev "sudo npm install nodemon -g" bagefter, og derefter "npm start" igen, hvorefter tingene kørte som de skulle.

8.1.3 Jævnfør GitHub & Git

Der henvises til punkt 7.1 om GitHub & Git.

8.2 Læring i forløbet

I dette forløb har jeg lært utrolig meget omkring det at opbygge, designe, konstruere og implementere et system. Jeg startede egentlig med at begynde at kode uden at have lavet mine UML-diagrammet, og i det hele taget uden at have tænkt over, hvordan jeg ville strukturere opgaven og udviklingsprocessen. Til trods for jeg egentlig godt vidste, at det var den forkerte måde at angribe en implementering på. Grunden til dette var vel, at jeg følte, at der var et kæmpe forarbejde inden jeg reelt kunne komme rigtigt i gang med at kode og kan se en udvikling og fremgang i procesudviklingen. Jeg ønskede blot at komme ud over stepperne og få krydset det ene kravsspecifikationerne af efter det andet. Dette viste sig dog hurtigt at være en superdum og ineffektiv måde at håndtere kodeudviklingen, fordi jeg for det første ikke havde et reelt overblik eller nogle faste mål og konstruktionsrammer at gå efter. Jeg faldt derfor ned i et sort hul, hvor jeg ikke kunne få tingene til at arbejde eller kommunikere sammen. Resultatet af dette blev, at jeg besluttede mig for at starte helt forfra, og denne gang gøre tingene i den rigtige rækkefølge. Selvom det var en hård beslutning at kassere alt, hvad jeg hidtil havde fået formået at kodet sammen, så viste det sig hurtigt, at det var den rigtige beslutning. Processen kørte i andet forsøg meget mere glat, da der nu var styr på tankerne bag interaktionerne mellem users, og hvordan klasserne skulle snakke sammen, samtidig med der var lagt en reel plan for, hvordan processen for systemets og dertil kodens udvikling skulle foregå.

[7] https://www.npmjs.com

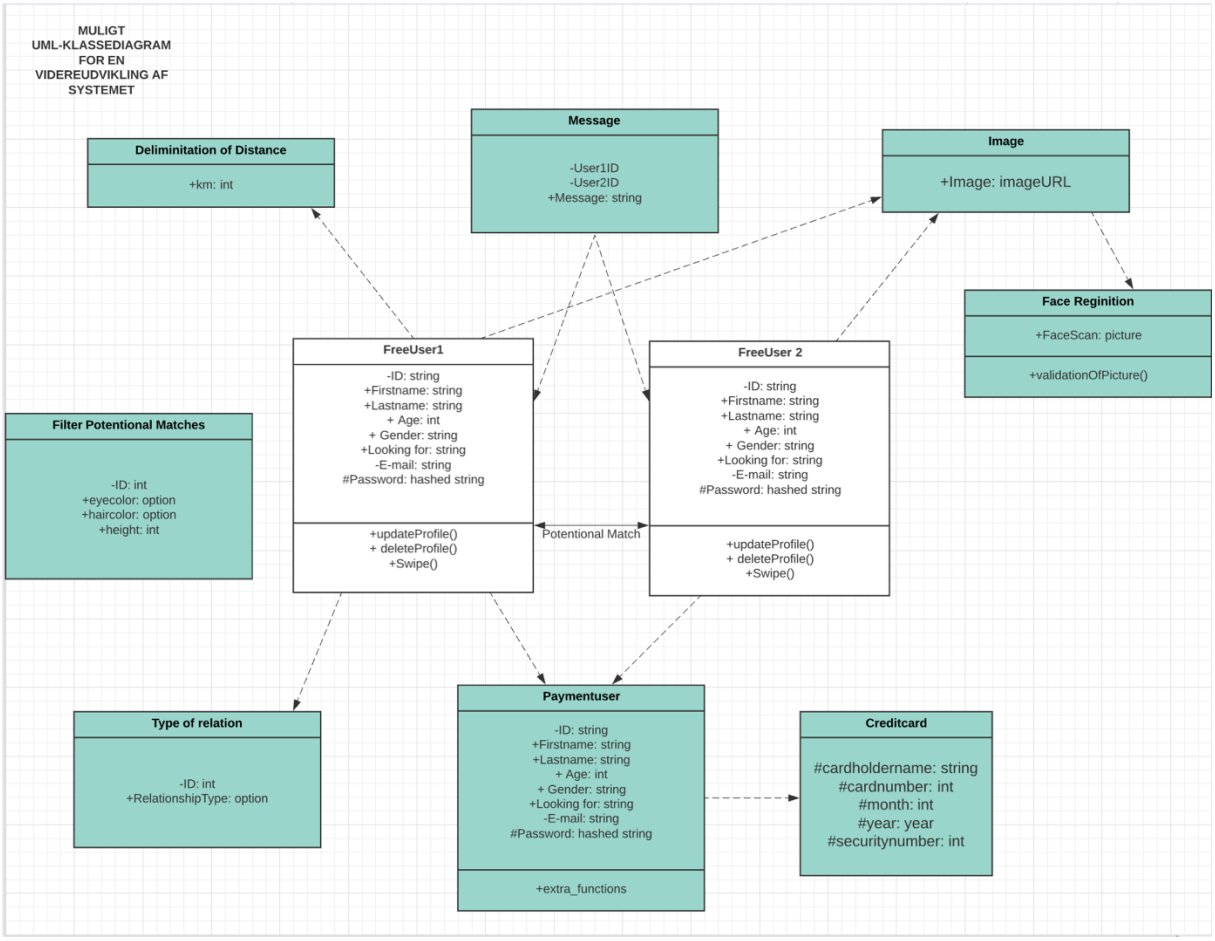
Jeg føler desuden selv, at jeg er blevet langt mere sikker på at kode, fordi forløbet har været så intenst, og man har siddet med koden intensivt så mange dage i træk. Jeg har fået en bedre forståelse for et API's opbygning, dertil fået Three Tier model rigtigt ind under huden og få brugt det i praksis. Jeg synes særligt delen med local storage og server-side storage, og hele processen i, hvordan databasen skulle gemme og opdatere info har været spændende og fangende.
Men det jeg nok har lært allermest i dette forløb, er nok at Rom ikke blev bygget på en dag. Det kræver tid, og måske tusinde fejl før man får en rigtig. Det er både det sjove, men også det totalt frustrerende ved at kode og programmere.

9. Perspektivering

Formålet med denne eksamen har været at udvikle et delprodukt ift., hvad der rigtigt indgår på en virkelig dating-platform. Dvs., at der er specifikke forretnings-og programmeringsafgrænsninger og mangler ved mit program, som adskiller sig fra en rigtig dating hjemmeside. Disse afgrænsninger og mangler skulle optimeres, hvis denne app i fremtiden skulle blive en reel konkurrent til Tinder. Hvis man skulle kigge lidt på, hvad konkurrenterne inden dating-apps indeholder af funktioner og metoder, så mangler dette delprodukt for det første muligheden for at indsætte et billede af sig selv, så man "ved hvad man liker". Dernæst muligheden for at kommunikere med et match på platformen, hvilket er en rimelig væsentlig faktor for en dating-app, hvis et match skal kunne udvikle sig til mere mellem brugerne end blot at bare at være et match.
Samtidig har appen ingen afgrænsninger i forhold til afstanden mellem potentielle matches. Altså har brugerne ingen mulighed for at vide, hvor i verden det potentielle match er fra/befinder sig. Dette ville i så fald være en relevant funktion at have med, hvis denne app skulle videre udvikles. Det samme gælder muligheden for afgrænsninger på alder, netop hvis en bruger kun ønsker potentielle matches forslået af programmet, hvis disse opfylder brugerens alderskriterier f.eks. en aldersgrænse mellem 22-27 år. Dertil måske en valgmulighed for at tilvælge, hvilket slags forhold man søger.
Man kunne desuden forestille sig at der blev oprettet en ekstra user klasse, der kunne være en betalende user. En betalende user kunne f.eks. have fordele i form af at kunne se en liste over, hvem der havde liket brugeren. Dette ville dog kræve en større sikkerhed i storage, når det har med betaling og opbevaring af fortrolige oplysninger at gøre, netop som opbevaring af kortoplysninger er.

Hvis man skulle lege med nogle nye tanker om forbedringer og optimering indenfor datingapp-branchen, kunne det være relevant at implementere helt nye features, der ikke er set før. Når man i dag sidder og swiper er der reelt ingen garanti for, at personen man matcher med, rent faktisk er den som personen udgiver sig for at være, dvs. der er ingen validering af, om personen faktisk eksisterer. Det kunne derfor være en ide at indføre en form for ansigtsgenkendelse. Hvor programmet går ind og scanner brugerens ansigt og sammenligner med brugerens billeder på profilen. I så fald ville det også blive en mere "sikker" app, hvor man ikke kan snyde med sin identitet. Dette har dog nogle programmeringsmæssige begrænsninger, hvilket henvender sig til programmøren, da det kræver store programmeringsmæssige kompetence at implementerer sådan en ansigtsgenkendelsesfunktion i programmet. Endvidere ville en optimering af disse begrænsninger kræve en stor optimering af allerede eksisterende kode plus kræve en stort ekstra indsats for at få disse funktioner til at spille sammen.

Ligeledes kunne det have en værdi for brugeren at konstruere en funktion, der rummer et slags ”filter”, så brugerne kan filtrere andre brugere i sin søgning efter potentielle match på app’en. Et filter, der indeholder hårfarve, øjenfarve eller højde på matchet man søger efter, så f.eks. at programmet kun foreslår mænd med brunt hår, blå øjne og over 180cm.
Jeg har nedenstående forsøgt at lave et UML-klassediagram, der ville beskrive design-strukturen for et videreudviklet program.



10. Konklusion

Som konklusion på procesudviklingen og implementeringen af et administrativt objektorienteret program, har jeg altså igennem en kravsspecifikation fået kreeret og designet UML-diagrammer herunder klasse-diagram og use-case diagram, der skulle sætte rammerne og give en struktur for denne implementering af programmet, samt både give et større forståelse for programmets funktionalitet og indretning, men også give en form for retningslinje at gå efter. UML-notationen viste sig at have en enorm indvirkning på procesudvikling, da den strukturerede programmet og på en måde ”holdt mig i hånden undervejs”, når jeg kom lidt i vildredde. Uden UML havde implementeringen af systemet været meget mere rodet og uoverskuelig.
Desuden blev der redegjort for, hvordan datastrukturen er opbygget igennem Three Tier modellen og MVC-mappe struktur til at holde de forskellige kodedele adskilt og holde overblikket over de forskellige instanser i udviklingen. Der redegjordes ligeledes for de enkelte tiers i systemet, og vist at

jeg kan formåede at implementere nogle af de allervæsentligste javaScript-funktionaliteter til relevante formål i vores program som f.eks. array som datastruktur. Samtidig blev der diskuteret fordele og ulemper for de forskellige anvendte metoder, og dertil opstillet mulige alternativer til disse b.la. local storage's fordele fremfor session storage.

Der blev fremvist forskellige metoder indenfor testing af programmet både løbende i implementeringen, men også sluttesting af programmet. Det blev tydeliggjort, hvordan error handling kan være et hjælpsomt redskab i kampen om at opspore potentielle brugerfejl, men også de mulige fejl jeg som programmør har fejlagtigt har lavet i koden. Der blev dertil forklaret om brugen af unit testing både ved brugen af console.log() som et strategisk testredskab af små kodestykker, men også udkommentering som et værktøj til at opfange fejl. Der blev desuden forklaret om brugen af Postman som et testharness, hvis formål var at isolere problemer og køre hver unit test case selvstændigt.
Der blev ligeledes i forlængelse af dette også redegjort for en sluttest i form af en black box test, der samlede op om programmets funktionalitet og brugerens interaktioner med denne, samt generel feedback om siden og mulig optimering.

Gennem udviklingsprocessen blev det hurtigt klart at dokumentering var en vigtig del af implementeringen af et program. GitHub og Git var et kæmpe nederlag at anerkende ikke at kunne få til at køre som ønsket, og hvor kommentarer i koden blev en central spiller i at holde overblikket for koderne og funktionaliteten af de forskellige metoder og funktioner. Til erfaring vil der i en fremtidig programimplementering lægges flere kræfter i GitHub og Git, da jeg savnede de mange fordele disse værktøjer har under en sådan programimplementering.
Igennem hele rapporten bliver der løbende stillet skarpt på løsningsovervejelserne i processen og stillet alternativer op til disse, hvortil der blev udarbejdet en procesevaluering, der forklarede 3 større problemer jeg havde undervejs i udviklingen. I procesevaluering blev der desuden forklaret, hvad jeg har lært i løbet af denne udviklingsproces, og hvad jeg vil tage med mig videre.

Afslutningsvist blev der i perspektiveringen kigget på mulighederne for en fremtidig videreudvikling af programmet. Der blev dertil kigget på, hvilke mulige funktioner og elementer, der kunne optimeres og implementeres for at blive en skarp konkurrent til andre DatingApp's. Disse nye tiltag ville dog kræve en større omstrukturering og nyt design af UML-diagrammer, og der blev derfor vist et muligt UML-klassediagram af en optimeret og videreudviklet datingApp.

11. Litteraturliste

Online bøger:
https://eloquentjavascript.net , Marijn Haverbeke, third edition, 2018. – senest besøgt 10/12/2020

Links:
https://masteringjs.io/tutorials/express/redirect - senest besøgt 05/12/2020

https://jwt.io - senest besøgt 05/12/2020

https://stackoverflow.com/questions/1027224/how-can-i-test-if-a-letter-in-a-string-is-uppercase-or-lowercase-using-javascrip - senest besøgt 05/12/2020

https://stackoverflow.com/questions/27978868/destroy-cookie-nodejs - senest besøgt 05/12/2020

https://en.wikipedia.org/wiki/List_of_HTTP_status_codes - senest besøgt 05/12/2020

https://expressjs.com/en/4x/api.html - senest besøgt 05/12/2020

https://nodejs.org/en/docs/ - senest besøgt 05/12/2020

https://ejs.co - senest besøgt 06/12/2020

https://developer.mozilla.org/en-US/ - senest besøgt 03/12/2020

https://www.w3schools.com - senest besøgt 10/12/2020

https://www.npmjs.com/package/axios - senest besøgt 05/12/2020

https://www.w3docs.com/snippets/javascript/how-to-get-url-parameters.html - senest besøgt 05/12/2020

https://www.lampdocs.com/how-to-reload-a-page-only-once-using-javascript/ - senest besøgt 05/12/2020

https://www.nemprogrammering.dk/Tutorials/c-sharp/21-arrays.php - senest besøgt 09/12/2020

https://medium.com/swlh/javascript-array-mutability-immutability-93d366c90751- - senest besøgt 09/12/2020
https://programmering.systime.dk/?id=202 senest besøgt 07/12/2020

https://www.npmjs.com senest besøgt 10/12/2020

https://arkitektur.digst.dk – senest besøgt 08/12/2020

Videoer:
https://www.youtube.com/watch?v=_EP2qCmLzSE&t=1147s – senest besøgt 03/12/2020

https://www.youtube.com/watch?v=6iZiqQZBQJY - senest besøgt 03/12/2020

https://www.youtube.com/watch?v=f-2jDPgh_Ng&list=PL4cU-xeGkcC9iqqESP8335DA5cRFp8loyp&index=1 - senest besøgt 05/12/2020

https://www.youtube.com/watch?v=0D5EEKH97NA – senest besøgt 01/12/2020