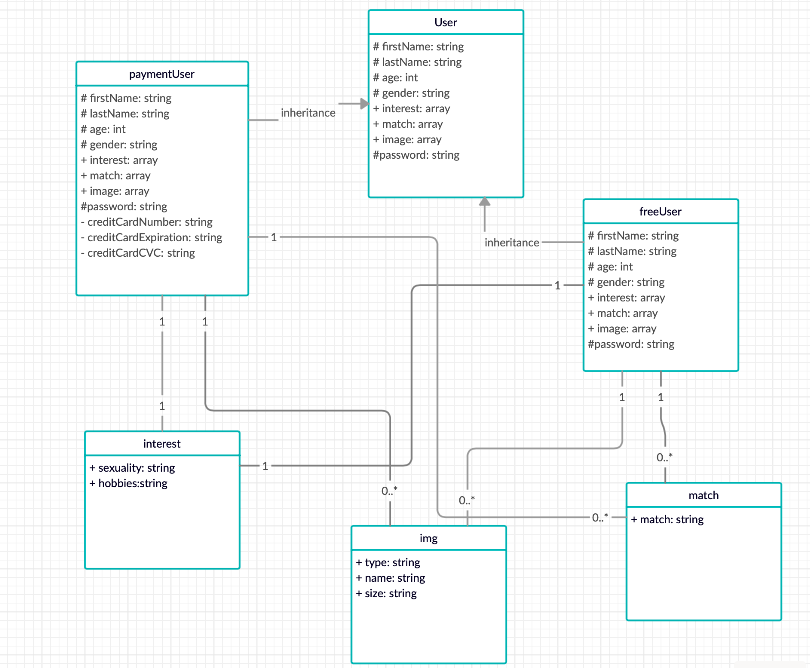
Godkendelsesopgave 3

Programmering og udvikling af små systemer samt databaser

Rapporten indeholder i alt 5 sider, eksklusiv forsiden



I denne godkendelsesopgave har jeg benyttet MVC modellen til at strukturere min kode så den er mere overskuelig. Fordi vi ikke skulle lave frontend i denne opgave, har jeg dog ikke benyttet mig af modellens V: view. Jeg har derfor kun lavet to mapper; ”controller” og ”model”. Min ”model” mappe indeholder filer med alle mine forskellige klasser. Det er altså her jeg håndterer den data, som jeg kan sende videre til min controller. Min ”controller” mappe fungerer således som min API, da den sørger for at hente den korrekte data fra min ”model”, for at videresende denne data til den server, som efterspørger den. ”Controller” mappen modtager og sender således requests fra serveren, og fungerer dermed som bindeledet imellem server og data.

Jeg kunne godt have undladt at dele mine filer op på denne måde. I stedet kunne jeg have haft alle mine klasser, min hardcodede data til de pågældende klasser, og mine requests i en og samme fil. Men dette havde ikke blot skabt et stort rod, det havde også gjort det umuligt at eksportere den korrekte data, som jeg efterspørger i mine forskellige endpoints for users, interests og match. Det er derfor væsentligt, at man i hvert fald deler de ønskede endpoints op i hver sin fil. Denne fil kunne godt indeholde klasserne og den hardcodede data, og stadig levere det ønskede endpoint.

OBS: mine hardcodede filer ligger inde i min ”controller” mappe, da jeg ikke kunne få det til at fungere hvis de lå i min model mappe, hvor jeg egentlig gerne ville have haft dem jævnfør MVC modellen.

Fordi koden er delt op i så mange filer og mapper, har jeg benyttet ”module.exports” og ”require” for at få filerne til at være forbundet.

 Her eksporterer jeg min klasse ”User”, så jeg kan hente den i ”freeUser” og ”paymentUser” filerne. I disse filer nedaver jeg således User klassen til mine to underklasser ved at require det eksporterede element fra User.js filen, altså User. Fordi filerne ligger i samme mappe, er stien til filen bare (”./user”). Havde den ligget i en anden mappe, skulle denne mappe angives først i stien.



Nu kan jeg extende User klassen til mine to underklasser ”paymentUser” og ”freeUser”.

I mine underklasser laver jeg først en ny constructor med de nye attributter som gør sig gældende for underklassen, samt de gamle fra den klasse jeg nedarver fra. Dernæst bruger jeg ”super” til at specificere hvilke attributter jeg nedarver. Dette er smart, for så slipper jeg for at skulle initialisere alle de gamle atributter igen. I min ”paymentUser” klasse tilføjer jeg ”creditCardNumber”, ”creditCardExpiration” og ”creditCardCVC”. Jeg gjorde mig mange overvejelser omkring hvordan jeg skulle inkorporere denne ”betalings del”. Jeg ville egentlig gerne have benyttet JWT tokins til at beskytte kortoplysningerne, men fordi det ikke er en reel datingapp med frontend, så valgte jeg den ovenstående beskrevne løsning, for at holde det mere simpelt. Havde jeg brugt tokins, ville jeg have lavet en separat klasse for ”creditCard”, for at disse kunne gøres hemmelige for uvedkommende. Jeg havde desuden lavet en funktion, som kunne kontrollere at kortoplysningerne var korrekte. I min hardcoding for ”paymentUser” har jeg derfor i user4 skrevet kortoplysningerne ind som \*\*\*\*, i stedet for tal, for at illustrere, at de skal være hemmelige.



Udover ”User”, ”paymentUser” og ”freeUser” har jeg lavet klasserne match, interest og img.

Interests har jeg lavet som sin egen klasse, så jeg kunne lave et endpoint hvor jeg udelukkende henter denne data. Man kunne godt bare have skrevet interesserne ind som et array i den hardcodede data for mine users. I stedet har lavet et array med de forskellige users interesser, og required det i min ”hardUser” fil. Her har jeg sat dataen ind ved at kalde på den plads i mit importerede userInterestArray jeg vil have for den pågældende user. 

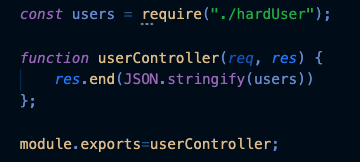
Samme fremgangsmåde har jeg benyttet for klasserne ”match” og ”img”.

Img har jeg altså også lavet som en klasse, fordi det ikke er muligt at indsætte et billede, uden frontend. Klassen består således af attributterne ”type”, ”name” og ”size”, for at angive at der er tale om en .PNG, altså et billede, samt hvor stort det er og hvad jeg har kaldt filen. Havde vi lavet frontend, ville jeg lave en funktion, som tilladte brugeren at indsætte et eller flere billeder. De skulle også kunne slette dem, og rette i dem.

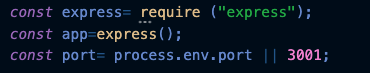
Ligeledes for klassen ”match”, har jeg forsimplet den en del pga. mangel på frontend. Jeg har altså bare lavet en klasse med attributten ”match”. Jeg har dernæst hardcoded nogen strings, som indeholder en besked om hvem usersne har matchet med. Jeg ville gerne have lavet en funktion, som kunne sammenligne køn, alder, sexualitet og interesser, og dermed finde ud af hvilke brugere der kunne være et match. Alternativt, hvis vi lavede frontend, ville jeg lave en funktion som gjorde, at brugerne selv kunne gå ind og ”like” andre brugere. Hvis begge ”likede” hinanden blev de et match. De skulle dernæst kunne skrive sammen over appen. Desuden ville jeg implementere særlige funktioner for ”paymentUser”, så de fx kunne se hvor mange der har ”liket” dem mm.

Jeg har valgt at lave tre filer i min ”controller” mappe: ”userController”, ”matcController” og ”interestController”. Disse filer henter data fra de hardcodede filer. Da jeg eksporterer dataen fra mine hardcodede filer som arrays, bruger jeg JSON.stringify i min userController funktion, for at endpointsne kan vise dataen for mine users som strings. Min userController funktion har til formål at angive det response som min get request skal bruge når jeg kalder det i min app.js. Jeg ville gerne have lavet en funktion som kunne hente dataen for den enkelte user. Dette ville jeg gøre ved at kalde deres userID, men jeg kunne ikke få det til at virke, hvorfor jeg nu nøjes med at hente dataen for hele mit userArray.

Samme fremgangsmåde har jeg også brugt i mine match- og interest controllers.



Fra mine controller filer eksporterer jeg således den data mine endpoints kan hente. Jeg henter dataen i min app.js fil. Her kan jeg aktivere de forskellige CRUD endpoints jeg vil tilgå.



Det gør jeg ved at bruge pakken express. Denne tillader mig bla at bruge http request.

Desuden angiver jeg den port, som min server lytter på.



App.use er et get request, som viser dataen for mine users. Jeg henter først dataen fra mine controller filer og angiver dernæst den route denne data kan findes på. I dette tilfælde kan dataen for mine users altså findes ved at skrive localhost:3001/users i søgefeltet.

Jeg vurderede, at det ikke gav mening, uden en frontend, at have andre CRUD endpoints end Read, altså et get-request, når der ikke er nogen reel brugerinteraktion.

Mangel på frontend har altså gjort, at koden er blevet forsimplet en del. Det har betydet, at dataen skulle hardcodes og at jeg har lavet færre interagerende elementer og funktioner.

Dette har også en betydning på CRUD endpointsne.



Jeg har dog stadig lavet de andre endpoints, men det eneste de reelt gør er, at komme med en besked om hvad endpointet skulle have gjort. Hvis der havde været en frontend, ville app.delete reelt kunne slette en bruger eller andet. App.put kan bruges til at opdatere dataen og app.post kan bruges når ny data skal tilføjes af brugeren.

Jeg valgte at undlade at implementere JWT tokins, da det blev for teknisk i forhold til hvad vi på nuværende tidspunkt har lært samt det øvrige omfang af opgaven. Desuden vurderede jeg, at det ikke var så væsentligt at bruge tokins, når vi ikke skulle lave en frontend hvor faktiske brugere skulle kunne logge ind. Dette medvirker derfor også, at jeg ikke har lavet en loginfunktion, og at jeg ikke kan hemmeliggøre hverken kortoplysninger eller password for mine brugere. Jeg kan sagtens se formålet med at implementere JWT tokins i forbindelse med en login function, når vi skal lave en dating app med frontend hvor brugeren selvfølgelig kun skal have adgang hvis den pågældende bruger er logget ind.

Jeg har brugt Git til løbende at gemme mit projekt. Dette er nyttigt da jeg så kan tilgå gamle versioner af min kode, hvis noget af det nye jeg implementerer skulle ødelægge det forhenværende jeg har lavet. Jeg har derfor to branches, den ene, main branch, bruges til at gemme den udgave af programmet som jeg ved virker. Den anden branch, develop, bruger jeg til at arbejde videre på projektet. Når den nye udgave virker, kan jeg så ”merge” min nye udgave over på den gamle. Desuden er GIT meget nyttigt når man skal lave gruppeprojekter, for at andre derigennem kan tilgå og videreudvikle ens kode.