# Programmering og udvikling af små systemer samt databaser

# Godkendelsesopgave 3 – API til datingapp

## Dato: 18-10-2020

## Studienummer: 126997

## Antal sider: 5 inkl. forside

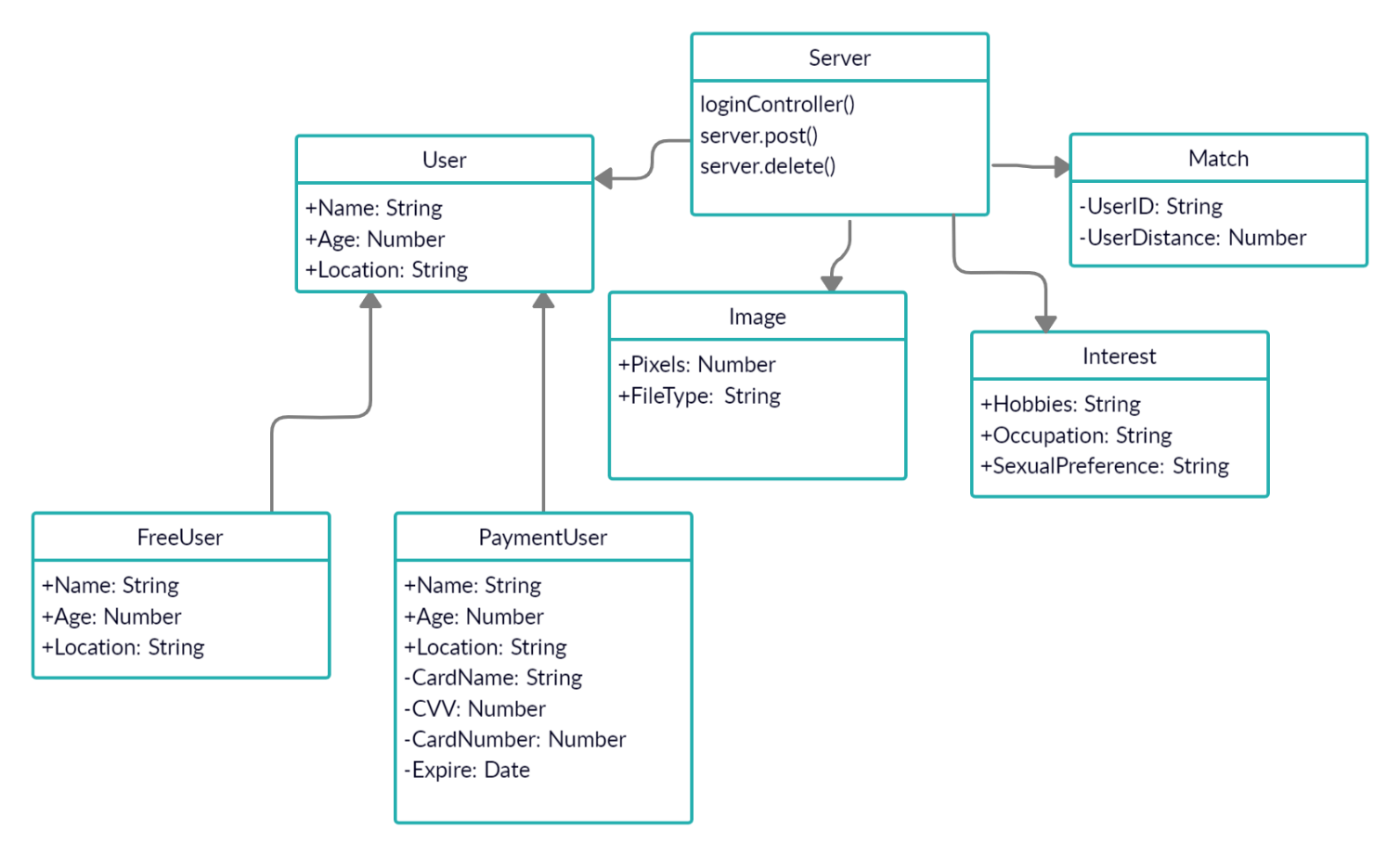
## Antal tegn: 4578 inkl. mellemrum

## Klassediagram til oversigt over projektet

Objektorienterede teknikker fungerer godt i situationer, hvor komplicerede informationssystemer gennemgår kontinuerlig vedligeholdelse, redesign og tilpasning til det foranderlige forretningsmiljø.   
Unified Modeling Language (UML) er kort beskrevet modellering af objektorienterede systemer vha. diagrammer, der giver udvikleren mulighed for at visualisere konstruktionen af et objektorienteret system.  
Dette værktøj kan forbedre kvaliteten af design og proces og derved hjælpe med at skabe informationssystemer af højere kvalitet.

For at tilrette- og planlægge mit API projekt startede jeg ud med at tegne et klassediagram, som gav et overblik over konstruktionen og sammensætningen af klasser og server og de tilhørende funktioner og attributter.

Klassediagrammet, der følger UML-notationen, ser således ud:

Hermed kunne jeg gå ud fra at starte med at udvikle serveren, som taler sammen med alle klasser i programmet. User er en af disse, men klassen har desuden tilkoblet to nedarvende klasser, FreeUser og PaymentUser. Pilen indikerer, at der er tale om inheritance relationship dvs., at de to subklasser nedarver Users egenskaber. Dog skiller PaymentUser sig ud ved at have kreditkortdetaljer som private attributter, da denne betalende bruger oplever flere fordele såsom at kunne superlike og have et ubegrænset antal swipes dagligt. Disse fordele er dog ikke programmeret ind i dette projekt.

## Server og klasser

Jeg startede som nævnt med at opbygge min server ved at importere express og sætte den til at køre på og lytte til port 3000 (jeg kunne også have valgt port 4000, 5000 osv.).

Herefter programmerede jeg mine klasser, som alle skulle ende med at tale sammen med serveren for at muliggøre CRUD-endpoints. CRUD, som står for Create, Read, Update og Delete, indeholder fire funktioner, som vi kender fra de fleste internetsider og apps. I dette tilfælde vil de f.eks. konkret gøre det muligt for brugeren af vores datingapp at opdatere beskrivelsen af sig selv på sin profil og slette matches, som han/hun fortrød at have liket.

Herefter kodede jeg alle klasserne, som projektet skulle indeholde: User, FreeUser, PaymentUser, Interest, Image og Match. Hver klasse oprettes i en separat JavaScript-fil og sluttes af med module.exports = (x klasse), så klassens kode eksporteres til vores server. Hvis vi lod alle klasser være i samme fil, kunne module.exports have svært ved at læse alle klasser, som ligger efter den første.

* User:

For at gøre det så simpelt som muligt, valgte jeg kun at give moderklassen, User, tre attributter; navn, alder, lokation. Jeg kunne have tilføjet mange flere relevante detaljer såsom køn og login egenskaber, men dette projekt har blot til formål, at vise hvordan serveren snakker sammen med klasserne ved CRUD-endpoints, og senere vil vi så gøre datingappen mere realistisk.   
Den betalende bruger, PaymentUser, nedarver ligesom FreeUser parent klassen, Users egenskaber samt har kreditkortoplysninger som endnu nogle attributter, der i den senere udvikling af systemet vil kunne give denne bruger fordele.

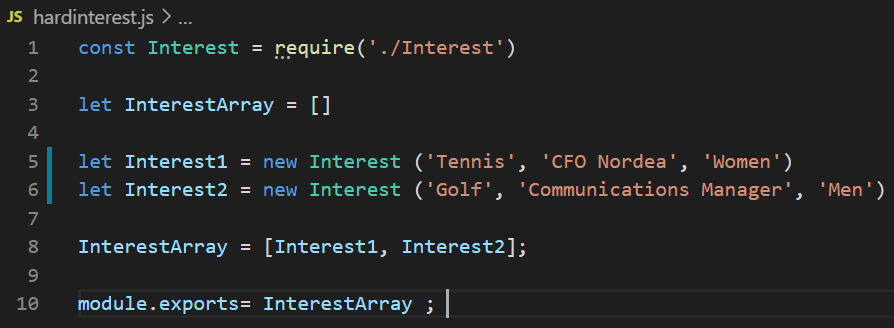
* Interest, Image og Match:

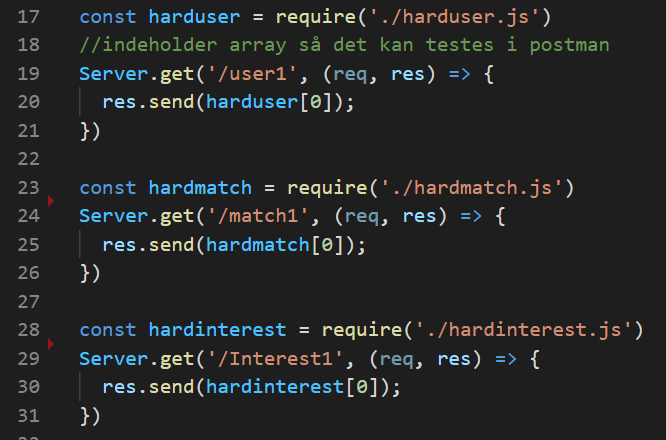
Iblandt overvejelserne ved udviklingen af projektet og klassediagrammet var, hvorvidt jeg skulle lave Interest, Image og Match som klasser, attributter eller funktioner på de forskellige users. Sidstnævnte kunne fx være en funktion på User, der muliggjorde billede-upload, hvor vi så også skulle tilføje validering af filen i vores server.

Jeg endte dog med at lave de tre som klasser, da det for mig ville hænge bedre sammen med struktureringen af CRUD-endpoints i serveren, som i forvejen skulle på klassen, User.

## CRUD-endpoints

## Efter oprettelsen af klasserne gemte jeg dem i serveren og satte den til at require de filer, som klasserne ligger i: Herefter lavede jeg CRUD-endpoints til User, Interest og Match ved brug af post og delete funktionerne, som giver brugeren en bestemt besked, når han/hun benytter funktionerne såsom at slette et match:

Nu hvor projektet ikke indeholder en front-end såsom en HTML-fil, som vi kan teste vores endpoints i, hardcodede jeg specifikke værdier til User, Interest og Match i tre nye filer, som så kunne testes i Postman.   
Det så således ud for klassen Interest:  


Efter at have lavet arrays med værdier til klassernes forskellige konstruktører, koblede jeg de tre nye filer til min server. Jeg kodede herefter dét, som Postman skulle ”lede i” (filerne fx hardinterest.js) og hente (let variable fx Interest1 array), når jeg lavede GET requests i programmet.   
  
Jeg sikrede mig til slut, at serveren virkede og hentede de rette oplysninger fra de hardcodede brugere i Postman:  
