
analysis
pictureready
illustration
designwebsite
web
technology
typography
photometal
tag
CSS
Express
www
concept
development
http
software
HTML
traffic
product
description
computer
term
TODO
PUG
artworkgallery
project
rank
floating
process
send
internet
hang
webtag
write
PHP
webdesign
page
src
online
conten
seo
quality
layout
JS
list
testing
cloud
word
agency
metal
media
webpage
engineering
router
hypertext
design

Indholdsfortegnelse

Indledning

I dag er manges kalendere præget af huskelister og små opgaver, man ikke må glemme. Derfor er “to do”-lister meget aktuelle, og det gennemgående tema for dette projekt. Vi skal udarbejdet et website, hvor man kan oprette brugere med forskellige roller, indskrive opgaver og lignende, som man vil huske, med tilhørende dato og tidspunkt. Man skal også have muligheden for at gemme det, bl.a. via JSON eller XML, så brugere kan se hinandens “to do”-lister. Dette er fx brugbart på en arbejdsplads.

Problemformulering

“Hvordan kan vi lave en brugervenlig “to do” applikation, hvor man kan oprette en bruger med tilhørende bruger-id og et password?”

Metodeovervejelser

Endnu engang har en af vores primære værktøjer været Microsoft Teams, hvor vi har snakket sammen hver eneste dag. Vi har været gode til at strukturere vores arbejdstider, også selvom det er online. Teams har nogle besværligheder til tider, men brugen af møde-funktionen i programmet har gjort det meget let for os at arbejde sammen, når vi nu ikke kan være fysisk til stede i samme lokale. Derudover har vi til selve teamwork-delen også brugt Trello som altid, da den er god til at give os et overblik over både de opgaver, der skal udføres, samt en fornemmelse for en realistisk tidsplan.

Til selve projektet har vi mest benyttet os af metoder, vi i forvejen har skabt kendskab til - mere om dette i research afsnittet herefter. Vi er blevet glade for at bruge pug som en template til vores websites, så lige præcis den har vi igen valgt at bruge til vores projekt, ligesom sidste gang.

Research

For at komme igang med projektet, har vi først kigget på, hvad vi har lavet i det forrige projekt og hvad, der har givet mening at genbruge derfra, så vi ikke starter helt forfra. Derefter har vi sat os til at undersøge, hvordan vi på den smarteste måde kan lave en database, som skal være udgangspunkt for de gemte “to do”-lister på vores website.

Funktionaliteten ved at oprette en bruger og log in er noget af det, vi har valgt at genbruge fra det tidligere biblioteks-projekt, men som noget nyt har vi arbejdet med at give brugere roller. Derfor har vi naturligvis også været ude og undersøge, hvordan man kan opnå en mulighed for, at en bruger med administratorrettigheder

kan godkende eller afvise andre brugere. Det er nemlig noget ret nyt, og vi har ikke kunne komme i tanke om noget, vi før har arbejdet med, som er i samme stil. Vi har derfor brugt en god del af vores research og tests på user roles, hvilket der også bliver forklaret mere om i konstruktions-afsnittet.

Analyse

Normalt vil vi måske planlægge og skitsere, hvordan vores website skal se ud, men idet dette projekt er kommet kort efter det forrige, har vi i stedet fokuseret på bare at springe ud i det og begynde med noget kode, vi allerede har lavet. Selve opbygningen og strukturen af vores filer er bygget op ved hjælp af Express, og så bruger vi igen pug som en template til vores side.
Det har givet os en del besparelse af tid at have muligheden for at tage udgangspunkt fra vores projekt fra ugen før, og derfor har vi simpelthen også bare valgt, at selve vores layout på siden er meget ligesom ved det tidligere projekt, da design ikke har været så meget i fokus i denne omgang. Vi har brugt næsten al vores energi på funktionaliteten, som vi har opdelt i punkter og mindre opgaver løbende. Vi skal som sagt lave et website, hvis formål er at give brugere en liste, hvorpå man kan skrive opgaver og noter, man skal huske i løbet af dagen, ugen eller hvad tidspunkt, man ønsker. Derudover skal der oprettes forskellige roller, som har forskellige rettigheder, såsom at godkende eller slette andre brugere, fx hvis en bruger ikke opfører sig ordentligt eller ikke bruger websitet hensynsfuldt.

Konstruktion

Vi bruger node express med Pug som viewtemplate til at starte vores projekt. Til det laver vi en database med to kollektioner. Selve vores projektmappe som bliver oprettet kommer til at have forskellige mapper: Views, routes, node-modules, models, public, bin. Efter vi har lavet en mongorestore kommer vores dump mappe ind i projektmappen.

I vores database har vi som sagt to kollektioner, som vi navngiver users og tasks. Vi efterligner det i vores models mappe, for at skabe et mere overskueligt overblik. I models har vi vores forskellige wrappers og skemaer til mongoose. Alle vores routere ligger inde i index.js. Det er bevidst, at de ikke er delt ud over flere, da antallet af routere og kodelinjer var overskuelig nok for os i en fil. Vores app.js ligger i rodmappen, og det er som udgangspunkt ikke en vi rører så meget ved. Vi har været inde i app.js og skrive to kodelinjer til express session, så vi kunne benytte os af session baseret authentication system.

User roles

En af de problemer, vi er stødt ind i, er i forbindelse med, at vi vil udarbejde en funktionalitet, som gør det muligt for en bruger med administratortilladelse at

godkende eller afvise andre brugere, der oprettes. Til at fremvise dette har vi bl.a. arbejdet med følgende form:

```
form(id='reguser' action='/approveuser' method='post')
    select
        option(value="verify") verify
        option(value="unverify") unverify
        option(value="admin") verify admin
    input(type="submit" value="Go")
```

Vi har dog haft en del problemer med at få fat i vores options i vores *handleUser.js* - men vi har Googlet meget om det og er nået frem til, at det er et relativt normalt problem, når man arbejder med pug. Men det har ikke fået os til at opgive - tværtimod har vi derefter brugt mere og mere tid på at prøve at løse netop dette problem, hvilket set i bakspejlet måske både var godt og skidt, da vi i sidste ende har brugt meget tid på det her ene problem.

Vi har dog fået lidt fremgang ved at tildele en *userrole* til vores *else if*-sætninger, som vist herunder, og derefter tilføjede vi også *userrole* til vores *form*. Før havde vores *select* ikke et navn tildelt, hvilket formentlig har været en af grundene til, at vi ikke har kunne få fat i den ordentligt. Derudover har vi også tilføjet *email* til *req.body* også i vores *else if*-sætninger, så der burde kunne findes et match vha. indtastede emails.

```
if (req.body.role = "unverify") { // If unverified, delete user
User.deleteOne({
    email: req.body.email,
    password: req.body.password,
    firstname: req.body.firstname,
    middlename: req.body.middlename,
    lastname: req.body.lastname,
    role: req.body.userrole});          // create object in db-format

} else if(req.body.userrole = "verify") {
User.updateOne({role: "verified", email:req.body.email});

} else if(req.body.userrole = "admin") {
User.updateOne({role: "admin", email:req.body.email});
}
```

```
form(id='reguser' action='/approveuser' method='post')
    select(name="userrole")
        option(value="verify") verify
        option(value="unverify") unverify
        option(value="admin") admin
    input(type="submit" value="Go")
```

I sidste ende, efter mange timers frustration, er vi dog kommet frem til en endelig løsning, som ser ret anderledes ud. Det ses udelukkende i vores kode indtil nu, men som en del af arbejdsprocessen vil vi også bare gerne fremvise lidt af, hvordan koden så ud på et tidligere tidspunkt, derfor dette eksempel med vores user roles.

Login

Håndteringen af login fra inputs på forsiden foregår i login.js. I det store hele er der ikke meget ændret fra vores fremgangsmåde i sidste projekt. Der er dog nogle vigtige forskelle.
I den if-sætning, hvor vi også checker password med bcrypt.compare, har vi lavet et check på om brugeren er blevet verified af en admin. Hvis brugeren ikke er verified bliver success sat til false selvom password er et match. Udover at session objektet styrer authentication så har vi også sendt nogle data om brugeren med som vi bruger blandt andet til at godkende brugere, at gøre forskel på hvad en admin kan se og hvad en godkendt bruger kan se, samt at gøre sådan at en bruger kan kan se tasks som denne har oprettet.

```
success = await bcrypt.compare(req.body.password, user.password);
if (user.role === "unverified") {
return !success; // Hvis bruger er unverified = forhindrer login
} else if (success) {
    req.session.authenticated = true;
    req.session.role = user.role;
    req.session.email = user.email;
    req.session.userid = user._id;
    console.log(req.session.role);
    console.log(req.session.userid);
} else {
    req.session.destroy(); //Kan bruges til logout
}
return success;
```

Registrering af ny task

På det view der hedder taskform kan brugeren indtaste en nye task. Ligesom med library projektet foregår det med en post funktion hvor et nyt objekt bliver lavet med titel, beskrivelse, udløbsdato, et id på den bruger som er logget ind, prioritet og en status. Det der er interessant her er at objektet har fået tilføjet id'et på den bruger der

er logget ind og det har den fået fra req.session.userid som vi satte da vi udførte login.

Fremvisning af tasks

Når vi viser task frem har vi 2 kolonner, en med to do tasks og en med doing tasks. Disse reagere på hvilken status de er sat til og det er udført ved hjælp af at definere variabler hvor der er sat de betingelser at den givne tasks pid skal svare til den bruger som er logget ind og at de har den status givet ved det regular expression der hedder *equal to* henholdsvis *do* eller *doing.*

```
let pid = req.session.userid;
let tasks = await handleTasks.getTask({pid: pid, status:{$eq:"do"}}, {sort: {title: 1}}); //Sc
let dtasks = await handleTasks.getTask({pid: pid, status:{$eq:"doing"}}, {sort: {title: 1}});
```

På samme side er det også muligt at ændre status på en task eller at slette den helt. Det er gjort med 3 links hvor den givne tasks id er tilføjet som en del af linket. Dette gøre det muligt via af routeren at fange tasks id med req.params som referere til det link som bliver udført i browserens adresselinje. Dette id sendes derfra videre til funktioner i handleTasks.js blandt andre changeTaskDone hvor det bliver brugt som betingelse i mongoose metoden findByIdAndUpdate for at finde den rette task og opdatere med en ny status.

```
const taskId = req.params._id.toString().trim();
await Task.findByIdAndUpdate(taskId, {status: "done"})
.then(function(){console.log("Status changed");})
.catch(function(error){console.log(error); // Failure
})
```

Hashing med Bcryptjs

Hashing er en envejsbillet til datakryptering. Vi brugt bcrypt til at hash brugeradgangskode og derefter gemme den i databasen. På denne måde gemmer vi ikke adgangskoder i almindelig tekst i databasen, og selvom nogen kan få adgang til en hashadgangskode, kan de ikke logge ind.

```
'use strict';
const User = require("./usersschema");
const bcrypt = require('bcryptjs');
const session = require('express-session');
const mongoose = require('mongoose');
const { compileClientWithDependenciesTracked } = require("pug");
const dbName = "todo";
const CONSTR = `mongodb://localhost:27017/${dbName}`;
const CONPARAM = {useNewUrlParser:true, useUnifiedTopology: true};
```

Saltet hashing - generering af tilfældige bytes og kombination af det med adgangskoden før hashing skaber unikke hashes på tværs af hver bruger adgangskode. Når brugeren forsøger at logge ind, kontrolleres hash af den almindelige tekst adgangskode nogensinde skrevet til harddisken.
Hvis hashene stemmer overens, får brugeren adgang. Hvis ikke, får brugeren ikke adgang.
Bcrypt giver mulighed for at vælge værdien af saltRounds.

Godkendelse af nye brugere

Som en del af opgavebeskrivelsen, skal vi lave funktionaliteter, der gør det muligt for en admin bruger at gå ind og godkende nye brugere som enten almindelig eller admin bruger. Vi har også lavet en funktion, der gør det muligt for admin at slette en afventende bruger, hvis man mener, at de ikke skal have tilgang til web-appen. Vi bruger vores session authentication system til at gå ind og se, om det er en admin, der er logget ind. Det er kun admin, der har adgang til viewet med approveusers. Web-appen bruger authentication systemet til at kigge efter, om en bruger har rollen admin eller ej.

```
    users,
    admin: req.session.role == "admin" ? true : false});
});
```

Inde i layout har vi så tilføjet en if admin, som er den, der afgør om man ser viewet eller ej i vores menu.
De afventende brugere bliver udskrevet på et view, som vi kalder for approveusers. Her kan admin ved brug af tre links nemt klikke på, om de skal godkendes eller ej. Users kommer fra vores database og ved hjælp af vores wrapper getusers og vores router, kender viewet users fra users kollektionen. Vores godkendelse bliver lavet ved, at man sender url for det link, som admin klikker på. Det bliver sendt videre til hvad end router, som passer med linket. Den tager så den email, som den afventende bruger har, samtidig med linket bliver sendt. Derfor går den videre til router og videre til den tilhørende wrapper.

```
console.log(req.params.email);
await User.findOneAndUpdate({email:req.params.email}, {role: "verified"})
.then(function(){console.log("Data deleted");})
```

Vi kan så, ved hjælp af en metode fra mongoose, opdatere den afventendes brugers rolle, eller den kan slette hele brugeren.

Tildele tasks til en bestemt user

En anden stor problemstilling, vi er stødt på i vores projekt, er opstået efter, at vi endelig fik både user roles og oprettelse af tasks til at fungere - nemlig at tildele tasks til en bestemt brugers id.

Vi har tildelt følgende roller:

- admin: kan gøre hvad som helst, som at oprette og slette en bruger.
- bruger: kan oprette, se og slette tasks

Evaluering af proces

I denne omgang har vi været gode til at opdatere vores Trello i løbet af hele projektugen, vi har været aktive meste af dagen og natten på Teams sammen for at løse problemer vi stødt i. Vi har været igennem to hårde uger med projekter, men vi har fået en masse erfaring ud af det i sidste ende. Vi har lagt mærk til at vi ikke har været så presset end forrige projekt.

Konklusion

Vi har formået at lave en web-app, der lader en bruger administrere forskellige opgaver eller tasks, så brugeren kan have en digital huskeliste. Det er muligt for en bruger at gå ind og oprette sig i systemet, men brugeren bliver ikke godkendt, før en admin har været inde og godkende. Web-appen gør således forskel på, hvilken rolle en bruger har. De oprettede opgaver bliver vist overskueligt i to kolonner på en side, der hedder showtasks. Det er kun de opgaver, som den indloggede bruger laver, der vises for brugeren. Er brugeren færdig med en opgave, så kan personen trykke på done, og de færdige opgaver vises så på en anden side. Vi har derfor besvaret vores problemformulering. Projektbeskrivelsen ønskede en udskrift af opgaver til JSON og XML, dette er ikke besvaret grundet tidsmangel.

Referencer

Durmus, S. (2020) *Making a ToDo List with HTML, CSS and Javascript* [online] available from <https://medium.com/clarusway/making-a-todo-list-with-html-css-and-javascript-154839b770b6> [22 March 2021]

Styling Lists - Learn Web Development | MDN (n.d.) available from <https://developer.mozilla.org/en-US/docs/Learn/CSS/Styling_text/Styling_lists> [22 March 2021]

Todo List App Using HTML CSS & JavaScript (n.d.) available from <https://dev.to/codingnepal/todo-list-app-using-html-css-javascript-5e7p> [22 March 2021]

Trello like Todo List (n.d.) available from <https://codepen.io/n4r4/details/JRNzqA> [22 March 2021]

Vue - Trello Smooth Drag and Drop (n.d.) available from <https://codepen.io/l-portet/details/jObbRYJ>