Rapport - Uge 12 - To Do

Indledning

I denne rapport vil vi komme nærmere ind på processen i det projekt vi er blevet stillet, i forbindelse med undervisningen og vores projektuge. I projektet skal vi lave en webapplikation omhandlende en todo-liste. På websiden skal det være muligt for en person at oprette en bruger for at få adgang til en privat todo-liste vha. En login funktionalitet. Brugeren skal først få adgang til login funktionaliteten når en administrator har godkendt oprettelsen. Når brugeren opretter en bruger, skal koden hashes når den bliver tilføjet til databasen for at implementere noget datasikkerhed. Todo-listen til hver bruger skal have nogle specifikke funktionaliteter, bl.a. Muligheden for at markere en opgave når den ikke længere er relevant, hvis den fx. Er fuldført eller at deadline er overskredet. Derudover skal alle opgaver gemmes i databasen, og på brugerens side, så man altid har mulighed for at gå tilbage i tiden og se tidligere opgaver. Til sidst skal det være muligt for brugeren at dele sin todo-liste med en anden person, som kan få adgang til indholdet ved at gøre brug af en adgangstoken.

Formålet med projektet går ud på at vi som studerende kan anvende de færdigheder vi har tilegnet os i fagene, på en praktisk måde, i en problemstilling som ikke er kendt af os i forvejen.

I projektet vil vi primært gøre brug af Node.js og Express sammen med MongoDB, men også JavaScript og andre teknologier vi har lært eller gjort brug af i forbindelse med undervisningen. Vi vil også gøre brug af teori fra vores undervisningen, og fra relevante bøger eller forums på nettet.

Vi vil i løbet af rapporten komme ind på hvilke metodeovervejelser vi har gjort os, på baggrund af den opgave vi har fået stillet. Derudover vil vi forklare hvordan vi har researchet for at få løst opgaven, og i vores tilfælde vil denne del mest gå med at forklare det vi har lært i undervisningen, da det er udgangspunktet for den stillede opgave, og vores research har dermed baseret sig på emner relevant for dette.

I analysefasen vil vi forsøge at besvare de spørgsmål vi har stillet os selv i vores problemformulering, for at analysere på hvordan vi har prøvet at løse dem.

I konstruktionsfasen vil vi gå mere i dybden med opbygningen af nogle af de forskellige elementer som udgør webapplikationen, og vores primære fokus vil her være på Express og MongoDB, som er de læringselementer opgaven fokuserer på.

Til sidst i rapporten vil vi evaluere hele processen, og komme ind på, hvilke dele af opgaven der har været mest udfordrende, samt hvilke dele var lettere at håndtere. Derudover vil vi komme ind på om der er noget vi ville have gjort anderledes, eller prioriteret på en anden måde hvis vi skulle gøre det igen, for til sidst at afslutte med en konklusion.

Problemformulering

Vi vil i denne opgave udvikle en web-applikation, der agerer som en To Do-liste. Applikation skal have en admin der skal godkende andre brugere når de har oprettet sig med mail og password. Det skal ikke være muligt at kunne komme ind på andre sider hvis brugeren ikke er logget ind.

To do-listen skal ikke slette fuldførte opgaver, men skal i stedet fremvises som “fuldført”. Brugere skal kunne levere deres To do-liste til andre brugere men skal være godkendt for at kunne se andres (adgangstoken).

Hvordan fremviser vi opgaverne som fuldført og igangværende?
Hvordan skal brugeren tilføje nye opgaver?
Hvordan skal brugeren markere en opgave som fuldført?
Hvordan kan vi designe en brugeroplevelse, der er intuitiv og logisk, så det vil være et praktisk værktøj i brugernes hverdag?

Metodeovervejelser

Når der bliver oprettet en bruger valgte vi at give brugeren en status der hedder “awaiting” dette gjorde vi for at holde styr på hvilket brugere en admin skal godkende eller afvise til at kunne bruge webapplikationen.
For at fremvise vores tasks brugte vi et each loop der kører igennem hver task i databasen og fremviser dem på en overskuelig måde til brugeren.

Vi valgte desuden at bruge Pug som template, da dette er ligetil at arbejde med, og det er det, vi har benyttet i undervisningen. Derudover var det som tidligere nævnt forudbestemt,
at vi skulle benytte Node.js og Express.

Research

I vores researchfase til dette projekt, som vi lavede før vi påbegyndte kodningen, gennemgik vi undervisningsmaterialet på dkexit.eu samt læste relevante dele af bogen “Get Programming With Node.js”. Målet var at repetere de dele af undervisningsmaterialet, der vil forberede os bedst muligt på de emner og problemstillinger, som vi eventuelt vil løbe ind i.

Vi har for eksempel brugt kapitel 21 på dkexit til at få styr på, hvordan vi bedst skriver vores views-dokumenter i pug, som er den template engine, vi igen har valgt at bruge. Det er også et nyttigt kapitel til at repetere, hvordan en ideel mappestruktur bør se ud, hvilket er nødvendigt igen i dette projekt, da størrelsen af det kræver mange undermapper.

Vi gennemgik kapitel 3 i bogen, som handler om opsætningen af en mongodb database. Det var selvfølgelig en god måde at forberede os på opgaven, da vi selv skulle oprette mongodb-databaser manuelt i denne uge. Vi brugte det til at opbevare data om vores To-Do-listes brugere, blandt andet deres mails, passwords, navne mm. Kapitlet i bogen hjalp os med selve opsætningen, men udvidede også vores

generelle forståelse for de muligheder, vi har i et projekt med brugen af mongodb til databaser.

Researchfasen fandt sted i løbet af de første par dage af projektets forløb, og blev primært lavet individuelt, hvorefter vi samlede os og evaluerede, hvad vi havde lært. Vi lavede også en tidsplan, som var baseret på vores viden forud for projektet og efterhånden ændret for at tilpasse vores estimat for, hvad vi skal lave og hvor hurtigt vi kunne dette. Tidsplanen findes under referencer.

Analyse

I vores problemformulering har vi spurgt hvordan vi sikrer os, at vores løsning indeholder funktionaliteter som både kan bruges af en administrator, men også af en registreret bruger. Hvis man kigger på vores konstruktionsdel, og vores løsning, og sammenligner det med vores problemformulering, er vi næsten nået helt i mål.

For at sikre at det ikke er muligt at se en todo-liste uden en konto, og at man samtidig ikke kan se en todo-liste tilhørende en anden person, har vi lavet en login funktion som giver adgang til ekstra sider tilhørende sin egen liste. Derudover har administratoren en ekstra side, med mulighed for at godkende oprettede brugere, så disse får lov til at kunne logge ind.

Hvis man som bruger gerne vil kunne se de opgaver der ikke længere er relevante, har de også mulighed for at gå ind på en side hvor al historikken vises. Så i øjeblikket bliver listen vist på to forskellige sider, alt efter om det er opgaver der stadig er relevante eller ej. Ved at inddele det i flere sider, har vi prøvet at gøre vores løsning forholdsvist logisk, ift. Hvordan man kan se de forskellige trin i todo-listen. Hvis det skulle være helt optimalt og mere brugervenligt, skulle vi have en enkelt side til alle funktionaliteter i todo-listen, men på nuværende tidspunkt valgte vi at gå med opdelingen af funktionaliteter til hver deres side.

Konstruktion

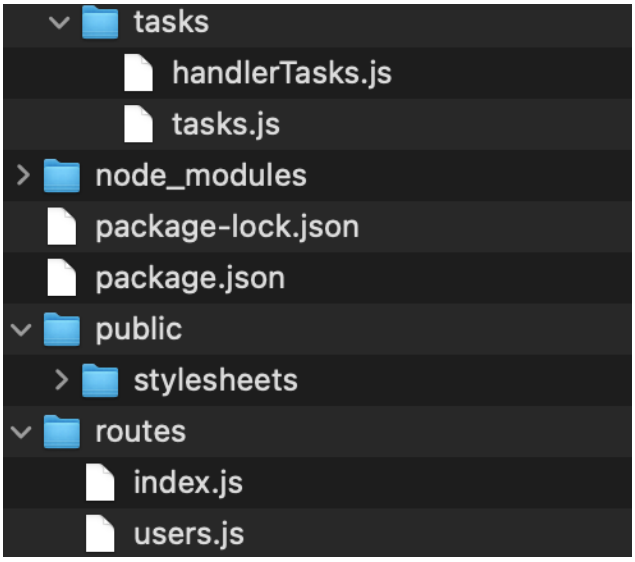
På de tre første billeder ses mappestrukturen for vores To-Do-listes kode. Vi har delt vores filer op i mapper efter funktionalitetsmæssig sammenhæng for at skabe overskuelighed blandt de mange filer, som projektet forlanger. Yderst i mappen har vi tre filer, heriblandt app.js, som blandt andet indeholder errorhandlers og requirements til diverse node-moduler, som vi skal bruge i vores filer. Dernæst er der Package.json og Package-lock.json. De to JSON-filer bruger vi til at liste projektets scripts, dependencies og fastgøre disse til bestemte versionsnumre.

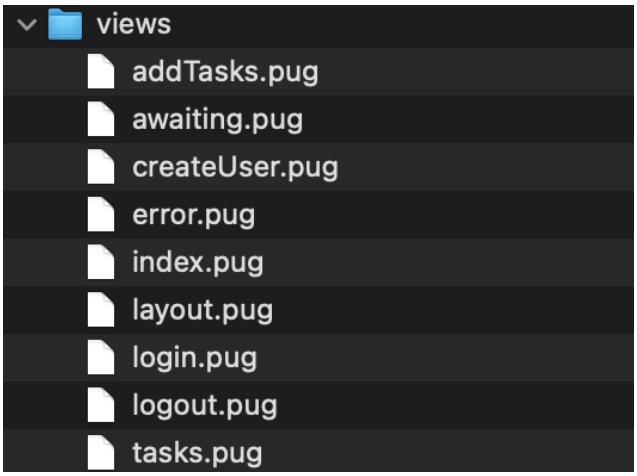
Vores bin-mappe indeholder www-filen, der finder laver selve serveren og finder porten mm. Dump-mappen indeholder vores databaser med accounts og tasks, som bliver fyldt op via input felterne af brugerne. Models.mappen indeholder fem js-dokumenter. Det er et object data modeling library, indenfor Mongodb, som blandt andet holder styr på sammenhængen mellem de datasæt, vi får fra input felterne i vores løsning. Altså de forskellige account og tasks der oprettes.

Node_modules har en lang række node.js-moduler i sig, hvoraf vi implementerer nogle i vores projekt. De node-moduler vi bruger kan ses i de yderstliggende JSON-filer. Public-mappen derunder indeholder vores CSS-stylesheets, som vi bruger til at ændre brugergrænsefladens udseende i håb om at styrke brugervenlighed. Vores index.js og user.js i route-mappen kan ses lidt som en rutevejledning, da deres primære formål er at redirecte brugere til forskellige URL'er baseret på deres inputs. Nederst i vores mappestruktur har vi views, som indeholder diverse pug-filer. De svarer til hver af de undersider, vores løsning har. Pug-filernes indhold er altså det man vil se, når man går ind på en af de forskellige sider. Vi har valgt at holde os til én template engine, pug, da den virker intuitiv og bliver brugt til løsningsforslag i undervisningsmaterialet, så den er nemmere at finde relevant hjælp til.

På det fjerde og femte billede har vi vores routes/user og views/createUser. Selvom filerne er delt op i mapper efter deres typer og formål, så er de selvfølgelig stadig forbundet på kryds og tværs heraf. Som det kan ses øverst i vores router, user.js, så bruger vi "const = require" til at introducere brugbare metoder og objekter mm. fra

vores filer i modelsmappen. Vi har også GET- og POST requests. Post requesten fra linje 18-24, for eksempel, submitter data som brugeren har indtastet i Pug-filens inputfelter, ind i vores database, hvorved en ny bruger bliver oprettet.





routes > JS users.js > ...

```js
var express = require("express");
var router = express.Router();
const account = require("../models/accounts/handlerAccounts");
const tasks = require("../models/tasks/handlerTasks");
const Tasks = require("../models/tasks/tasks");
const Account = require("../models/accounts/accounts");
const accounts = require("../models/accounts/accounts");

/* GET users listing. */
router.get("/", function (req, res, next) {
  res.send("respond with a resource");
});

router.get("/createUser", function (req, res, next) {
  res.render("createUser", { title: "Home Page" });
});

router.post("/createUser", async function (req, res, next) {
  let result = await account.createAccount(req);
  res.render("createUser", {
    title: "Create a new user",
    account: result,
  });
});
```

```
extends layout

block content
  h1= title
  p Welcome to #{title}
  h2 Register a new user:
  form(action="/users/createUser" method="post")
    input(type="text" name="firstname" placeholder="First name" required)
    input(type="text" name="lastname" placeholder="Last name")
    input(type="email" name="email" placeholder="Email" required)
    input(type="password" name="password" placeholder="Password" required)
    //- input(type="text" name="rights" placeholder="rights")
    input(type="submit" value="Create User")
```

Evaluering af proces

Vi har i dette projekt valgt at samarbejde om skrivelsen af kode. Dette giver os som gruppe et større overblik over hele koden, hvilket gør det hurtigere hvis et problem opstår og vi skal samarbejde om at løse det. Vi formår at nå vores deadline uden at uddelegering af arbejdsopgaverne, vi har haft en god planlægning af hvad der skulle laves først. Til dette har vi anvendt et Trello board. Vi startede projektet med et opstartsmøde, hvor vi skrevet ind alt hvad vi skulle når og til hvornår det skulle være klar på ugen. Trello boarded er opdelt i (“To do”, “Doing” og “Done”). Dette gav os en meget bedre struktur under arbejdsugen, og samtidig hjalp med at holde styr på hvad vi skulle have med i web-applikationen.

Konklusion

Passwords bliver opbevaret sin en hashet værdi, og er derfor sikre. Eftersom bcrypt har muligheden for at sammenligne input-password i plaintext, har vi valgt at bruge den funktion. Dette kunne dog optimeres bedre ved at hashe input og så sammenlign.

Siden fremviser brugers tasks i en overskuelig liste, hvor brugeren så kan trykke om selve tasken er completed eller slette den hvis han/hun giver op. Det er derefter muligt at se den færdige eller slettede task i en underside, hvor man så kan se completed date og/eller deleted date.

Brugeren har ikke mulighed for at komme ind på andre sider end create account og login, hvis de ikke er logget ind. Dette hjælper på sidens sikkerhed da ingen andre en folk som admin har godkendt kan navigere rundt i to-do listen.

Når brugeren er logget ind, kan de se hvilket tasks de har tilføjet, her har de muligheden for at færdiggøre dem ved at delete eller complete dem. Når en bruger vil tiløje en ny task, skal de skrive start dato og deadline. Til hvis man vil starte senere end da man laver den.

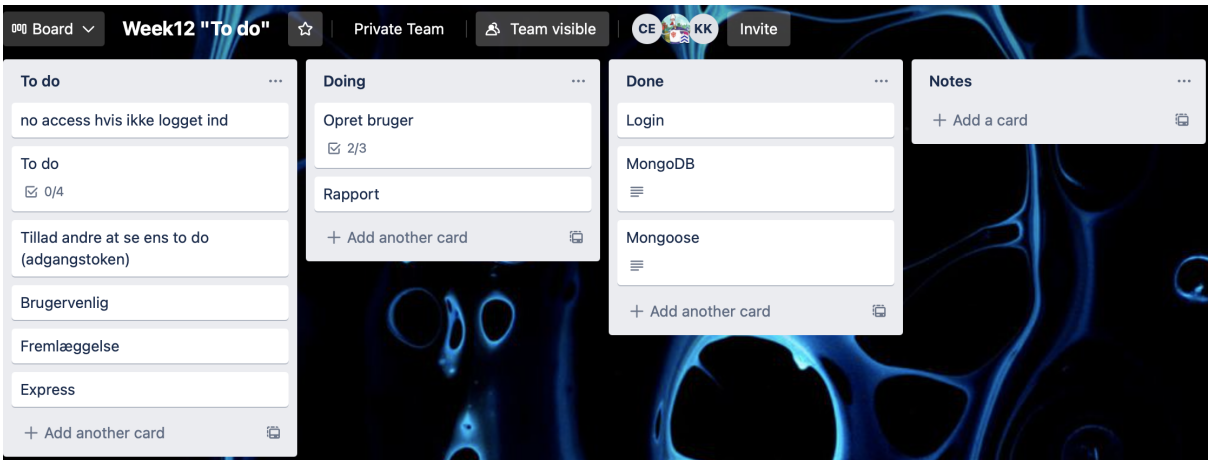
Hvis en admin er logget ind kan se se en side der hedder awaiting, hvor de kan gå ind og se alle admins, brugere og awaiting accounts. Her kan de så vælge at approve eller disapprove alle brugerne og give dem tilladelse eller fjerne deres rettigheder for at bruge siden.

Pug håndtere om hvis det er en user der er logget ind kan de ikke se awaiting siden, hvorpå den vil vise den side hvis det er en admin der er logget ind.

Applikationen indeholder samtlige funktioner, der er krævet i projektbeskrivelsen, og den er velfungerende. Hvis projektet havde haft en længere varighed, kunne man med fordel have brugt tid på at optimere brugervenligheden ift. styling samt flere funktioner.

Referencer

1. Tidsplan:



2. http://dkexit.eu/webdev/site/index.html (n.d.) *Web Development* [online] available from <http://dkexit.eu/webdev/site/index.html> [14 March 2021]
3. Wexler, Y. (2019) *Get Programming With Node.Js*. Shelter Island, NY: Manning Publications Co