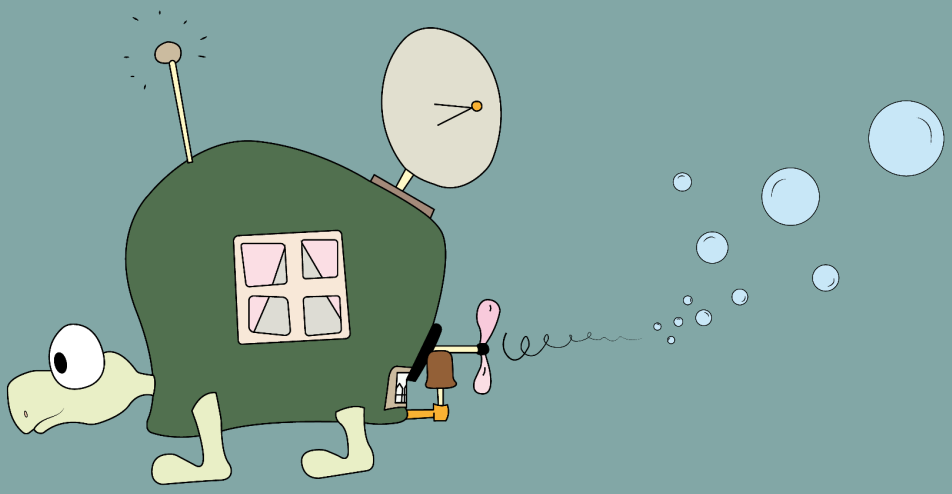
2. SEMESTER – PROJEKT 1



Gruppe 3 - Anna, Dalena, Kasper & Marco

INDHOLDSFORTEGNELSE

INDLEDNING

Når man har en hjemmeside, der kræver et login, indeholder det typisk specifikke informationer, som kan være sensitive. Det er derfor vigtigt at have styr på sikkerheden, så der ikke er potentielle hackere, der kan bryde ind i databasen og snuppe alle de sensitive informationer. I dette projekt kommer arbejder vi med logins, hvor dataen som brugerne taster ind, bliver gemt i en database og der bliver brugt metoder som hashing og salting til at sikre brugerens data. Daten består i todo-lister, som skal indeholde forskellige opgaver, og hele dette system er udviklet i node.js og express.js.

PROBLEMFORMULERING

Hvordan benytter man Express og NodeJS til at lave en ‘to do liste’ der har en database?

- Hvordan er dertilhørende login oplysninger sikre og beskyttet mod hacking med hashing og salting?
- Hvordan gør vi en lokal database public?

AFGRÆSNING

I løbet at opbygningen har vi tænkt på, at en custom state kunne være en god feature, hvor brugeren selv kan gå ind og tilføje, hvilke ‘state’ som ‘todo’, ‘inprogress’ eller ‘done’. Vi tænkte at dette kunne være en smart funktionen, da det giver brugeren flere funktioner og muligheder, dog har vi begrænset tid, og der ikke være tid til at implementere funktionen. Vi har derfor valgt at afgrænse custom states og blot lave nogle prædefinerede states.

Derudover har vi tænkt, at et list view kunne være måde at få et overblik over opgaverne som en liste. Det er en god funktion at have, hvor brugeren kan skifte mellem den almindelig overview, hvor man kigger efter ‘states’ og en liste med opgaverne, at efter hvilken oversigt man helst vil have. Her har vi dog valgt at lægge fokus på et overview, hvor opgaverne er inddelt efter states.

Drag and drop er ydermere en feature vi har overvejet, som gør det lettere for brugeren at skulle skifte ‘states’. Det er en funktion som mange kalendere og to do lister har. Vi ved at

den vil være bøvlet og vil kræve mere tid at implementere, og derfor har vi valgt at afgrænse funktionen grundet mangel på tid.

Caesar og Viginere kryptering er forskellige metoder til at kryptere passwords. Vi har valgt at afgrænse disse metoder, da vi tænker hashing og salting ville tage tid at implementere, da dette er helt nyt for os. Vi har derfor valgt at fokusere på det.

METODER

Desk research

I starten af projektet brugte vi metoden desk research, som er en research metode, hvor man går ind og kigger på eksisterende applikationer og får inspiration til, hvad der fungere godt fra dem. (Polak 2020)

Datasikkerhed

Hashing

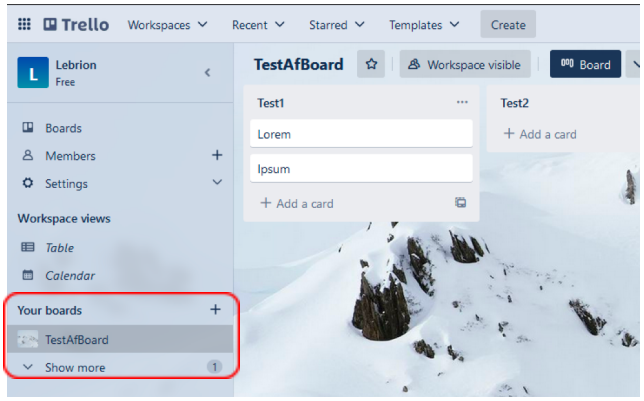
Hashing er en envejsproces der omsætter data, i dette projekt brugernavn og password, til en string af forskellige værdier. Denne string har dog en fast længde alt efter, hvilken algoritme man bruger til hashing. Input dataen fra brugeren gemmes dermed ikke i dens oprindelig form, men i stedet den hashede form i databasen. Når brugeren logger ind, matcher den inputtet med hash værdien, og bekræfter eller afkræfter om inputtet er rigtigt. Hvis databasen bliver hacket, vil hackeren kun have den hashede værdi af inputtet, men ikke det oprindelige input. Hashing i sig selv er ikke den mest sikre løsning, da hackeren kan bruge andre redskaber, som rainbow tables til at knække den hashede string, derfor tilføjer man som regel salting til ens hashing. (Lake 2018)

Salting

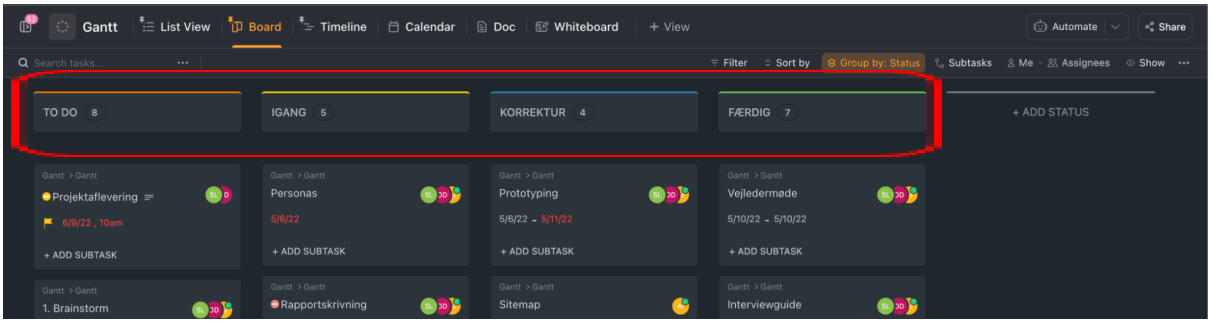
Salting er en tilføjelse til hashing, hvor inputtet får en tilfældig unik string tilknyttet inden den bliver hashet, hvilket gør at hackeren ikke kan gøre brug af rainbow tables, da hver string er unik. (Lake 2018)

RESEARCH

Som research er der lavet en desk research for at se på, hvordan andre har lavet et system til håndtering af opgaver opdelt i forskellige lister. Vi har i denne research kigget på to sider, som kan dette - Trello og ClickUp. Her har vi kigget på, hvordan siden er opbygget i forhold til visualisering af lister med dertilhørende todo's. Overordnet ligner siderne meget hinanden, da de kan det samme.
Ved Trello vises en overskuelig liste over ens todo-lister ude i siden, hvilket fungerer godt og giver nemt overblik over de lister man har, og man kan nemt skifte imellem dem. Se markering på billedet herunder.



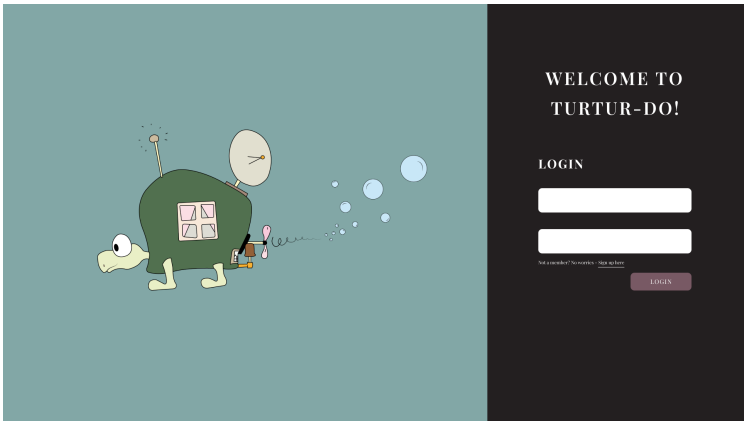
Hos ClickUp vises det på en overskuelig måde på ens liste, hvilke opgaver man er i gang med, udført og endnu ikke startet. Dette fungerer også super godt, da man kan flytte ens opgaver mellem de forskellige stadier. Det virker mest overskueligt, da man hurtigt kan se, hvilke opgaver der er i hvilket stadie. Se billede herunder.



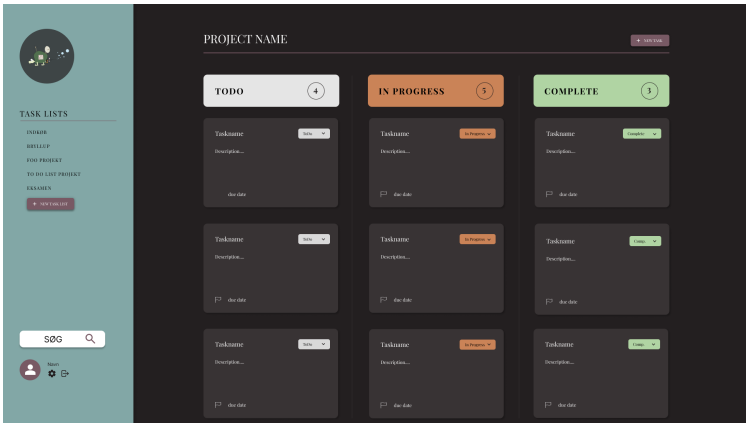
Ved begge sider fungerer det i øvrigt godt, at man kan klikke på en opgave, hvor der åbnes en modal med en længere beskrivelse af den enkelte opgave. Ved ClickUp kan der yderligere vælges om opgaven er en prioritet og ad hvilken grad.

QUICK DESIGN

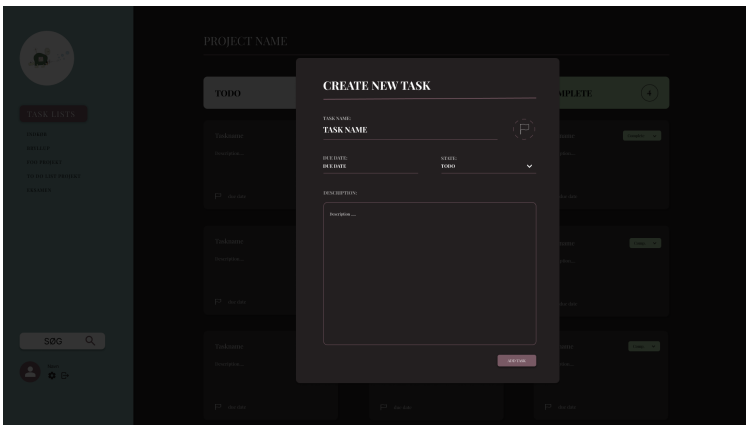
For at have en guide, at kode ud fra, har vi udviklet en hurtig prototype. Denne skal hjælpe med, at sikre et ensartet design af siden, når den udvikles. Designet holdes enkelt, så der er fokus på login, todo-lister og de enkelte todo's.
Der er lavet en login side udelukkende med mulighed for at logge ind eller at oprette en bruger. Denne ses herunder.



Derudover laves en side med overblik over opgaver i en todo-liste og hvilket stadie de er i, eller tilføje nye opgaver. I højre side ses en liste over alle ens todo-lister, som man kan navigere imellem eller oprette nye. Dette ses herunder.



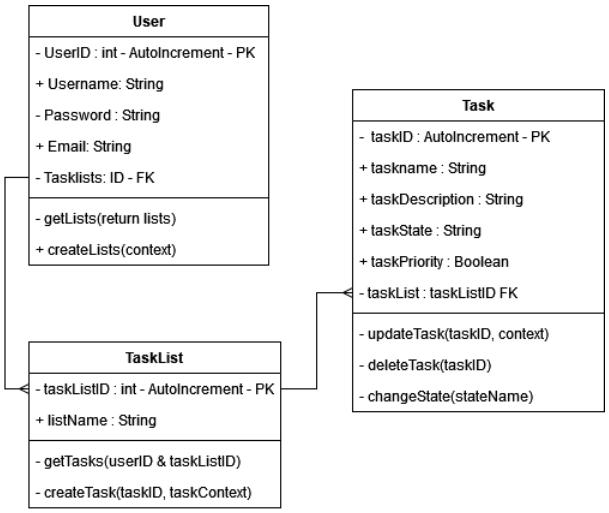
Der er også lavet en mulighed for at tilføje nye opgaver på ens todo-liste. Her tilføjes et navn, prioritet, deadline, state og en beskrivelse. Denne laves i en modal, som ses herunder.



KONSTRUKTION

Database

Vi har i denne opgave valgt at bruge en MAMP/WAMP database med mySQL. Databasen er sat op med et table til henholdsvis user, task og task list. User er sat op med userID, som primary key. Informationen på brugeren er name, password, email og en foreign key til brugerens task lists, der gemmes i task list table. Brugerens id, password og tasklist, skal være privat, så andre ikke kan få adgang til det.
Den table der skal indeholde task lists, der er tilknyttet brugeren, er sat op med et id og et navn. I Task table skal vil have de forskellige tasks, som indeholder id, der er privat, et navn, beskrivelse, state, prioritet og en foreign key, for at referere til den tasklist, den tilhører. Priority typen er som en af de eneste ikke en string men en boolean. Se entity relation diagram herunder.



Relationen mellem de forskellige tables er sådan, at en bruger kan have mange lister, men en liste kan kun have en bruger, relationen er altså en til mange. Det samme gør sig gældende mellem listen og task, hvor en liste kan have mange tasks, men en task kan kun have en liste. Altså er relationen her også en til mange.

Under hver enkelt table findes også funktioner, der skal være tilknyttet. Disse ligger ikke i databasen, men selve funktionaliteten tilgår databasen. Under user er disse funktioner en getList, som skal hente alle de lister, som brugeren har, og en createList, som skal oprette en ny liste med navn valgt af brugeren. Under task list skal det være muligt at hente de tasks, som ligger under denne liste eller oprette en ny task tilknyttet denne liste. Til sidst ved den enkelte task, skal det være muligt, at opdatere den, slette den eller ændre dens state.

Opsætning og struktur

Projektet udvikles med express.js og Node.js. Via npx generate opretter vi en projektbase med express og pug til templates, som automatisk opretter mappestrukturen for os, som skal danne den base, som vi kan arbejde videre på. Dette indebærer en public mappe med images, scripts og stylesheets. Under stylesheets laves to css filer, da vi har to typer af sider, login og selve todo siden. Da de to typer af sider er ret forskellige vælger vi at splitte det op. Under mappen med images tilføjer vi vores logo og endnu en mappe, som skal holde alle vores ikoner.

Derudover findes der en views mappe, som indeholder vores pug filer, der skal generere det der kan ses på skærmen. Her laves filer til de typer af views, der skal være på siden. Altså, login, opret bruger, index og en error side. Derudover ses også en layout fil, som skal være base for index og error-siden, og for at undgå at skulle gentage kode. Denne indeholder opsætningen af siden med titel, link til stylesheet og navigation. Derudover sættes et block content tag ind, som fungerer som placeholder for vores views og siger, at det vi skriver i vores views i block content, det skal placeres der på siden. I vores index og error kan vi altså extend layout filen og blot skrive det ind, vi gerne vil have vist på siden.

Der findes yderligere en routes mappe, som bruges til at håndtere de forskellige requests fra brugeren (Aayush 2021). Her kan vi håndtere fx. get requests. Eller hvis brugeres skal oprette en ny opgave under sin todo-liste, så vil det blive håndteret der med en post request. Det er her vores fra vores funktioner set i ER diagrammen skal køre. Et eksempel på dette er vores createTask funktion, som skal køres, når der skal laves en ny task i en liste. I vores

controller er selve funktionen skrevet, som ses herunder. Her bliver vores data hentet ind fra bodyen, altså navn, beskrivelse og deadline. Derefter bliver der connected til databasen. Herefter ser vi vores query, hvor vi vil sætte en ny række ind i databasen "tasks", hvor der skal sættes values ind i "TaskName", "TaskDescription", "TaskState" og "TaskDueDate". Derefter bliver vores values defineret, og dette er altså de values vi hentede ind fra vores body - de ting brugeren har skrevet ind. Nu bliver denne query sendt, og hvis der er en fejl, så vil den give fejlen, men ellers bliver den blot sendt.

```
// Handle task create on POST.
exports.task_create_post = (req, res) => {
    console.log(req.body)
    //Variables to hold body properties
    const newTask_TaskName = req.body.name;
    const newTask_TaskDescription = req.body.description;
    const newTask_dueDate = req.body.dueDate;

    //Make connection to Database
    connection.connect();

    //Query to insert all user data into Database
    let taskCreateQuery = `INSERT INTO tasks (TaskName, TaskDescription, TaskState, TaskDueDate)
    VALUES ('${newTask_TaskName}', '${newTask_TaskDescription}','ToDo', '${newTask_dueDate}')`;

    //Insert Data into Database
    connection.query(taskCreateQuery, (err, result) => {

        if (err) throw(err);


    })
    connection.end();
};
```

Denne funktion i controlleren vil blive kaldt i vores router. Vi har en action sat på vores form med en path, og når denne form bliver submitted, så vil den blive kørt i vores router, som ses herunder. Her har vi en post request, som bliver kørt ved denne path, altså når der bliver submitted, og den kalder vores funktion i controlleren, som sørger for det bliver sat ind i databasen.

```
// POST request for creating task.
router.post( path: "/task/create", task_controller.task_create_post);
```

Alle disse requests vil blive loadet ind i app.js, som bruges til at lave vores endpoints. Disse endpoints lytter efter et match på det enkelte endpoint, som så kører det specifikke request (Expressjs n.d.).

Datasikkerhed

For at validere og sanitere input af email, brugernavn og password fra brugeren har vi brugt Express-validator, som er et Express bibliotek, der bruges til at validere data i applikationer.

```
const { body, validationResult } = require("express-validator");

router.post("/create",
 body("email").isEmail().blacklist("'").blacklist(" ").blacklist("-"),
 body("username").isLength({min: 6}).isAlphanumeric().blacklist("'").blacklist(" ").blacklist("-"),
 body("password").isLength({min: 6}).blacklist("'").blacklist(" ").blacklist("-"),
 (req, res) =>{

  const errors = validationResult(req);
    if (!errors.isEmpty()) {
      console.log(errors);
      return res.status(400).json({ errors: errors.array() });
    }
```

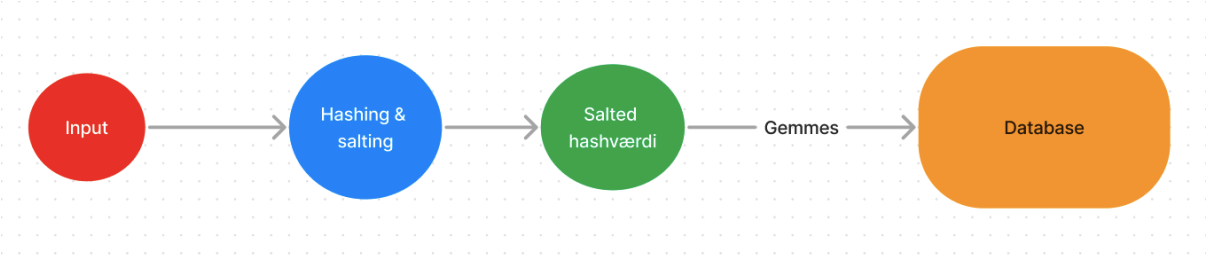
I dette eksempel fra routeren “createUser.js” valideres tre input, som er taget fra bodyen i pug-filen “createUser.pug”. Ved hjælp af Express-validators indbyggede funktioner tjekker den bl.a. om email inputtet er en email, giver en vis længde til password og brugernavn, og validerer om det er bogstaver eller tal. Udover det har vi bedt den om at fjerne bestemte specialtegn, hvilket sikrer imod SQL-injections.
Hvis kravene er godkendte går udførelsen igennem og bliver gemt i databasen. Hvis kravene ikke bliver godkendt, gemmes valideringsfejlene i et errors array og viser fejlene ud på siden i en JSON.

På ‘login’ og ‘opret bruger’ har vi benyttet hashing og salting til at gøre password sikkert. Til det bruger vi CryptoJS, som er en library der allerede har funktioner og metoder der hasher via SHA256, der er en af de mere sikre cryptographic hash-funktioner.
Ideen i vores sikkerhed er, at når en bruger opretter et login, vil hash-funktionen først executes lige inden passwordet bliver sendt til databasen. På den måde vil hash-værdien blive gemt og ikke det reelle password.

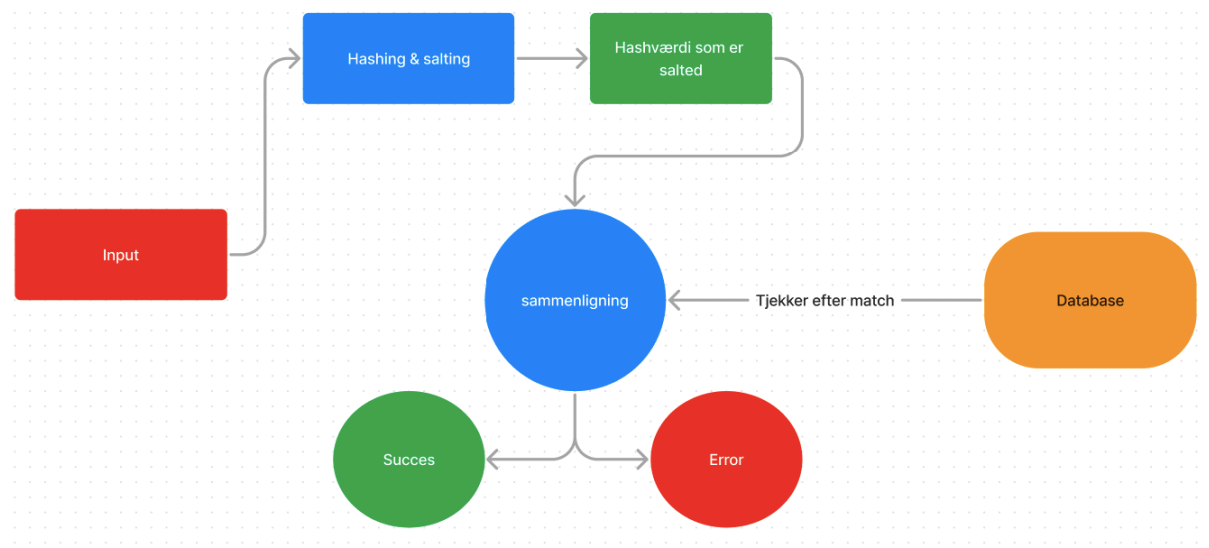
```
const newUser_username = req.body.username;
const newUser_password = req.body.password;
const newUser_email = req.body.email;

let salt = crypto.lib.WordArray.random(128 / 8).toString();

let saltUsername = (newUser_username + salt).toString();
let hashedPassword = crypto.SHA256(newUser_password + salt).toString();
```

I dette eksempel requester vi brugernavn og password fra pug filen og tilskriver dem som variabler. Vi laver variablen salt som genererer en saltværdi fra CryptoJS’ eget bibliotek.

Efterfølgende salter vi brugernavnet og hasher (SHA256) og salter passwordet med hjælp fra CryptoJS' bibliotek.



Når brugeren logger ind, har vi opsat et event, der starter processen med sammenligning af hash-værdi af brugerens input med det hashed password der er i databasen for den respektive bruger. Er der et match vil brugeren kunne logge ind og få adgang til applikationen. Hvis der er fejl, at man f.eks har tastet forkert eller at der ikke findes en bruger endnu, vil der opstå et fejl i stedet.



PROCESEVALUERING

I dette projekt har samarbejdet været godt. Der har ikke været nogle uoverensstemmelser i forhold til dette. Derimod har vores problemer været på selve projektet, da vi har været udfordrede på opgaven. Vi har haft udfordringer med at få MAMP/WAMP databasen til at fungere online, så vi alle kunne tilgå den samme, og derfor er der brugt lang tid på dette. Der har været andre ting vi har måtte droppe. Bl.a er det gået ud over funktionen til at logge ud. Derfor er der ikke implementeret en logud funktion, og det er derfor ikke muligt at logge ud, men dog er den tænkt ind i systemet. Havde der været mere tid, ville det også blive implementeret.

Derudover er det heller ikke muligt at vælge en state og sætte priority på. Der bliver blot sat en default state på, når der oprettes en ny task. Der er lavet en en dropdown til at kunne ændre state, men fordi den ikke er i en select, så kan værdierne ikke trækkes ud i routes. Disse er i stedet lavet som en section der bliver vist ved et klik. Dette opdagede vi for sent i processen, så derfor er det ikke muligt, at ændre state.

Når der bliver postet til databasen, når der laves bruger, task eller en list, er der lige nu ikke nogen respons til brugeren. Dette er noget der bør tilføjes, da det er vigtigt, at brugeren ved, at deres forespørgsel er gået igennem. Dvs. hvis de opretter en ny task, så skal de gerne have at vide, om den er blevet oprettet, altså en form for feedback.

KONKLUSION

Vi startede med at opsætte projektet med NodeJS og ExpressJS. Her har vi brugt npx generate, som sætter projektet op for os. Her har vi valgt at sætte det op med pug som templates. Via pug filerne har vi lavet siderne til login, create user og todo og derefter kan vi skabe forbindelse til databasen.
For at sikre vores login oplysninger, har vi salted og hashed brugerens passwords, når der oprettes en ny bruger. Når brugeren logger ind, bliver der matched hashed password fra brugeren input og den der ligger i databasen. Det skulle gerne give beskyttelse imod SQL injection.
For at gøre vores database public, har vi brugt WAMP til at gå fra require connection der ligger lokalt til ‘all granted’. Ved at gøre en database public kan man være flere om at tilgå databasen, og hvis vi har kunder kan de oprette sig og tilgå deres egen data.

REFERENCER

Aayush, Z. (2021) ‘What Is the Use of Router in Express.Js ?’ [2 June 2021] available from <https://www.geeksforgeeks.org/what-is-the-use-of-router-in-express-js/> [16 February 2023]

Expressjs (n.d.) *Express Routing* [online] available from <https://expressjs.com/en/guide/routing.html> [16 February 2023]

Lake, J. (2018) ‘Encryption, Hashing, Salting: What’s the Difference and How Do They Work’. [21 December 2018] available from <https://www.comparitech.com/blog/information-security/encryption-hashing-salting/> [16 February 2023]

Polak, P. (2020) ‘What Is Desk Research And How To Do It?’ [21 August 2020] available from <https://invotech.co/blog/what-is-desk-research-and-how-to-do-it/> [16 February 2023]