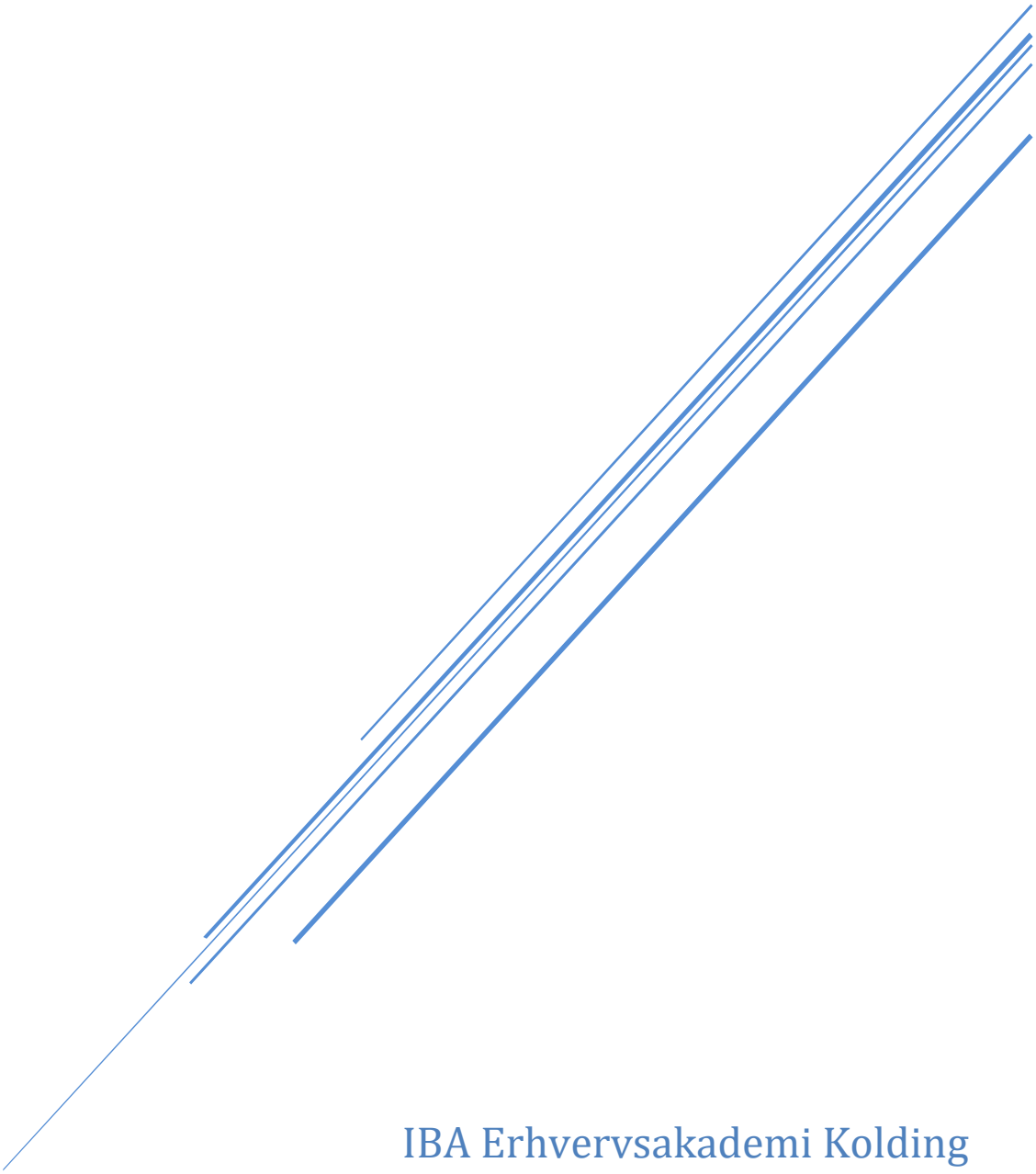
PROJECT 1

To Do List



IBA Erhvervsakademi Kolding
Webudvikling
Dennis, Freja og René

Indholdsfortegnelse

Indledning

Dette projekt tager udgangspunkt i at lave en "TO DO" liste. Det handler om at sikre at en bruger har mulighed for både at registrere sig samt at logge ind. Derudover skal brugeren have mulighed for at oprette en "TO DO" opgave og derved sikre overblikket over, hvilke opgaver der skal løses. Dette indebærer blandt andet at kunne oprette og se "TO DO" opgaver med datoer.
Derudover findes en administrator, som er med til at give brugere adgang til systemet. Administratoren skal aktivt ind og aktivere en bruger, før han/hun kan logge ind og bruge systemet.

Problemformulering

Hvordan kan vi skabe en applikation med Node.js, Express og MongoDB, hvor man som bruger kan logge ind og se sine To Do's, samtidig med at brugeren selv kan oprette sine egne To Do's. For at brugere har adgang til systemet, skal der tilknyttes en administrator til dette.

Metodeovervejelser

Tidsplan

Vi har valgt at lave en tidsplan, for at planlægge, hvornår vi skal have lavet de forskellige elementer. På den måde sikrer vi os, at vi har et overblik og kan nå at lave det vi ønsker.

Nodejs/Express

Vi bruger Express med pug-engine, til at skabe skelettet til vores projekt. Dertil bruger vi følgende opgave som skabelon til opbygningen af vores projekt. (Larsen 2021)

MongoDB/Mongoose

Vi bruger MongoDB til vores database. Hertil bruger vi npm modulet "mongoose" i stedet for "mongodb", da mongoose kan nogle flere ting end mongodb.
Vi bruger blandt andet mongoose til at oprette forbindelse til mongodb og dermed vores database. Dertil bruger vi den til at oprette vores collections og til at poste data fra databasen.

Git/Github

Vi anvender Git og Github til at dele filer med hinanden til projektet.

bcrypt

Vi bruger bcrypt til at kryptere brugernes password, når de opretter en bruger, samt når vi logger ind.

Cookies

Vi bruger Express egen cookie modul, som hedder cookie-parser.

Research

Det første vi gjorde til vores research, var at gennemlæse projektbeskrivelsen, for at forstå, hvilke ting vi skal have lavet. Til denne research fandt vi frem til, at vi skal lave følgende elementer:

Elementer til projektet:

- To do liste
- Database
- Sign up
- Login
- Administrator
- Admin, som skal acceptere brugerne
- To do til JSON

Elementer til To Do liste:

- Titel
- Tekstindhold
- Startdato
- Deadline
- Prioritet
- Fuldført

Til projektet skulle vi vælge imellem at løse opgaven i PHP eller med Node.js/Express. Vi valgte at løse opgave i Node.js og Express med mongoose, og derfor har vi lavet lidt research på disse værktøjer.

Node.js og Express

Da vi skal arbejde med en database, skal vi have et dynamisk projekt i gang. Hertil skal man vælge hvilken engine man vil arbejde med, når man arbejder i Express. Efter at have udforsket lidt i de forskellige engines, besluttede vi os for at arbejde med pug.

Express-engine: pug

Pug har en noget atypisk skrivemåde, når man skal skrive html. Der bruges fx ikke almindelig open-tags og der bruges slet ikke slut-tags.
Noget af det vigtigste ved at bruge pug er, at holde styr på sin placering af kode. Pug kan ikke så godt finde ud af, hvis et element ikke står korrekt, og kan enten give en fejl eller et andet resultat end det man ville have.

Mongoose

Som en del til vores løsning skulle vi bruge mongoose, derfor har vi lavet lidt research på, hvad mongoose er. Mongoose er et npm modul, som bruges til at håndtere MongoDB. Mongoose bruges i stedet for modulet mongodb, idet det kan det samme og mere til. Mongoose er Schema-baseret løsninger, som bruges til at danne modeller, som kan bruges til ens webapplikation. (Larsen n.d.)

Foreign keys i MongoDB

Da vi skal have to collections i vores database, skal vi have en form for forbindelse mellem de to, så brugerne får den rigtige to do liste vist frem, når de logger ind. Dette svarer til en foreign key fra relationsdatabaser.

Analyse

Databasen

Da vi på forhånd vidste, at vi skulle arbejde med en database, gik vi ind og analyserede, hvilke data vi skulle have med, og hvilken datatype disse skulle være. Vi fandt frem til følgende:

Database For TO DO			
Database	**Collection**	**Fields**	**Value**
tododatabase	user	firstname	string
tododatabase	user	lastname	string
tododatabase	user	username	string
tododatabase	user	password	string
tododatabase	user	e-mail	string
tododatabase	user	status	string
tododatabase	user	role	string
tododatabase	todo	title	string
tododatabase	todo	text	string
tododatabase	todo	startdate	Date
tododatabase	todo	deadline	Date
tododatabase	todo	priority	int
tododatabase	todo	is_complete	Date
tododatabase	todo	done	Boolean

Vi fandt frem til, at vi skulle have en database med to collections. En collection til brugeren og en til to dos. Til hver af disse collections har vi angivet navnene til de fields, som en colleciton skal indeholde, hvor vi også fandt frem til, hvilken datatype hver field skal have. Disse informationer skal vi nemlig bruge, når vi skal arbejde med mongoose-modulet.

Views

Vi har fundet frem til, hvilke sider vi skal have med. Først og fremmest skal vi have et login/signup side, hvor brugerne har mulighed for at logge sig ind. Hertil skal der være en side til dem hver, så en bruger både kan gå ind og signup eller logge sig ind.
Når en bruger har logget sig ind, skal vi have en forside, hvor brugerens to dos skal vises.
Hertil skal vi have en underside, hvor en bruger har mulighed for at tilføje to dos.
Til sidst skal vi have en admin-side, hvor en administrator kan gå ind og acceptere en bruger, når han/hun har signet sig up.

Signup

Vi skal lave en signup, hvor en bruger har mulighed for at oprette sig. Disse oplysninger af brugeren skal ind i en user collection i en mongo database.
Hertil skal vi have et system, hvor en admin skal godkende en bruger, før han/hun kan logge ind.
Fordi vi har med et kodeord at gøre, skal vi sørge at det bliver hashet.

Login

Ved vores login skal vi sørge for at tjekke, om det angivne brugernavn og password matcher - ellers kan brugeren ikke komme ind.
Til login'et skal vi også have en cookie, som registrerer en bruger, når han/hun logger ind.

Admin

Til admin-siden skal det kun være brugere, som er registreret som admin, som har adgang til denne side. Til admin-siden skal der være et overblik over alle registrerede brugere samt en status over, om de er blevet aktiveret eller ej.

Konstruktion

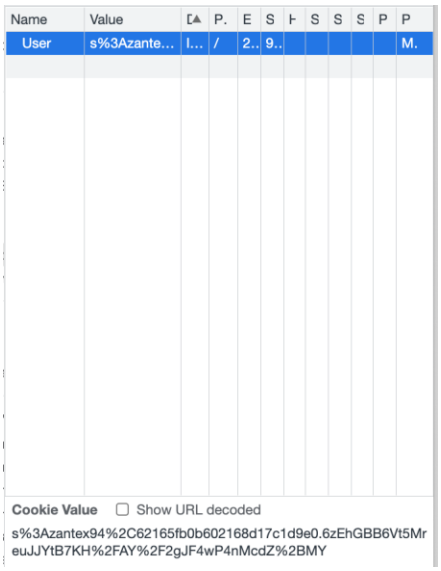
MVC

Selve vores konstruktion tager udgangspunkt i MVC(model, view & controller). Da vi har en brugerdel og en to do del, har vi lave en MVC til hver af disse. Det gør det mere overskueligt at finde rundt i.

Brugerdel

I projektet har vi valgt at dele alt med brugere op i en del, samt to do i en anden del. Herunder har brugere deres egen router users.js samt deres egen controller userController. Ved at dele logikken i denne MVC-struktur op er det nemmere at holde styr på de enkelte arbejdsopgaver som systemet tager sig af.

For at kunne bruge systemet var det nødvendigt at have et login system. Systemet kan kun bruges, når en bruger er logget ind. Dertil hører, at man som bruger har sine egne personlige to dos og det har vi gjort ved hjælp af cookies, hvor browseren gemmer en key samt value på hver eneste bruger i browseren.



Browserens key: User samt value : username + id

Derudover var opgaven at sikre at en bruger kan oprette sig i systemet. Dog skal en bruger først aktiveres af administratoren for at kunne gøre brug af systemet.

Cookie-parser module

For at sikre at browseren gemmer på brugerens data efter login var det muligt at gøre brug af et modul fra express. Dette modul hedder cookie-parser og skulle installeres ved hjælp af npm install cookie-parser. (*Cookie* 2022)
Derefter var det muligt at gøre brug af modulet ved blot at tilføje dette inde i app.js

```
// app.js
var cookieParser = require("cookie-parser");
```

```
// app.js
app.use(cookieParser('my cookie'));
```

Derefter var det muligt at gøre brug af cookies i routeren hvor vi sætter en cookie og derefter giver en response til browseren om at der eksisterer en cookie.

```
res.cookie('User', users[0].username + ',' + users[0]._id , { signed : true, maxAge: 1000*60*60*24*7 });
```

Cookien består af en key: User hvor værdien er username samt id. Derudover bestemmer vi hvor lang tid cookien skal eksistere i maxAge.

Hvis vi ønsker at få en cookie i routeren kan vi bruge req.signedCookies.name(i vores tilfælde User) for se om der eksisterer en cookie. Hvis ikke der er sat en cookie i systemet bliver brugeren automatisk sendt retur til login, da systemet kræver at der mindst er en cookie med de givne værdier som foroven.

Når en bruger vælger at logge ud bliver cookien sat til nul og brugeren henvises derefter tilbage til login siden.

Signup

Systemet gør brug af en signup side, hvor brugeren har mulighed for at registrere sig. Efter registrering er det op til administratoren at aktivere brugeren. Systemet opretter nemlig en bruger med status inactive, så brugeren ikke kan bruge systemet med det samme.

```
postNewUser: async function (req) {
    const db = await mongoConnect.mongoConnect();
    bcrypt.hash(req.body.password, 10, function(err, hash)
        let user = new User({
            firstname: req.body.firstname,
            lastname: req.body.lastname,
            username: req.body.username,
            password: hash,
            email: req.body.email,
            role: "user",
            status: "inactive"
        });
        User.create(user, function(error) {
            if (error){
                console.log(error);
                db.close();
                return true;
            }else{
                db.close();
                return true;
            }
        });
    });
},
```

Signup består af en formular med action post, som ved aktion går ind i routeren hvor vi sender request med som parameter. Req parameteren indeholder de værdier fra formularen, som vi bruger, når vi laver vores schema user som vist foroven. Derefter bliver en bruger oprettet ved hjælp af vores schema.

Administrator

Systemet har en administrator som er med til at bestemme, hvilke brugere der får adgang. Denne administrator er oprettet med en bestemt username: zantex94, og kun denne har mulighed for at tilgå admin-siden som vist nedenunder.

Activate/Deactivate users

Firstname	Lastname	E-mail	Username	Status	
Dennis	Nissen	dennis@iba.dk	den	active	deactivate
Freja	Hyldgaard	freja@iba.dk	freja	inactive	activate

Admin for alle brugere.

Forsøger andre brugere i systemet at tilgå admin-siden bliver disse henvist til en anden side. Når admin vælger at ændre status på en bruger trykker denne på et link, som derefter opdaterer databasen.

```
    updateUser: async function (req, x, y, status, user) {
        const db = await mongoConnect.mongoConnect();                    // connect
            User.findOneAndUpdate({username: user}, {status: status}, function(error) {
                        if (error){
                            console.log(error);
                        }
                        db.close();
                    });
    }
};
```

Udsnit af userController.js som tager sig af at opdatere en bruger. Parameteren status består enten af active eller inactive. user holder på brugerens navn så den ved hvilken bruger der bør opdateres.

To do

To do liste

For at kunne markere en todo som færdig, bruger vi en button der er tilknyttet en boolean fra databasen med en default værdi som false. Hvis man så klikker på knappen, vil den ændres til true og på den måde skal den respektive todo, som repræsenterer et unikt dokument fra databasen, flyttes til en sektion som indeholder alle de færdige todos på siden.

```
each todo in todos
    - if (!todo.expired) {
    form(method='POST' action=`/todolist/${todo._id}/completed`)
            tr.aligncenter
                td #{todo.title}
                td #{todo.description}
                td #{todo.startdate.toLocaleDateString()}
                td #{todo.deadline.toLocaleDateString()}
                td #{todo.priority}
                td
                    button(type='submit' class="btn w-xs green") Done
                tr
                - }
```

På billedet ses den her button, som er inde i en form, hvis action indeholder et dokuments unikke _id, som bliver indsat i URL parameteren.

```
router.post("/todolist/:id/completed", function (req, res, next) {
    if(typeof(req.signedCookies.User) === "undefined"){
        res.render('login', {
                title: TITLE,
                subtitle: 'Login'
          });
        }else{
  controller.checkBtn(req, res, next);
  res.redirect("/todolist");
```

Herefter kommer det ind i routeren hvor "/:id" bliver erstattet med det respektive dokuments id, og så bliver checkBtn called.

```
checkBtn: async function (req, res) {
  await mongoConnect.mongoConnect();
  let todoId = req.params.id;
  await Todoschema.findById(todoId, function (err, docs) {
    docs.done = !docs.done;
    if (err) {
      console.log(err);
    } else {
      console.log("Result : ", docs);
    }
    return docs.save();
  });
```

checkBtn tager så Id'et fra URL-parameteren og putter det ind i variablen todoId. Derefter bruger vi findById til at identificere dokumentet, som så bliver refereret i parameteren "docs". Herefter kan vi så tage fat i "done" med docs.done, som er den boolean tidligere nævnt. docs.done bliver så sat til det modsatte ved !docs.done, hvilket gør at hver klik på knappen nu vil ændre værdien fra false til true og omvendt.

```
controller.getTodo({userId: userarr[1]}, { sort: { priority: 1 } }).then(function (results) {
    let todos = results.filter(function (todo) {
    return !todo.done;
    });
    let doneTodos = results.filter(function (todo) {
      return todo.done;
    });
    let expiredTodos = results.filter(function (todo) {
      if (new Date() > todo.deadline) {
        todo.expired = true;
      }
      return todo.expired;
    });
    res.render("todolist", {
      todos: todos,
      doneTodos: doneTodos,
      expiredTodos: expiredTodos,
      todolist,
      user: req.signedCookies.User,
```

Nu hvor button virker, mangler vi at specificere de forskellige sektioner på hjemmesiden hvor en todo skal være i forhold til dens status. Det gøres ved at deklarere lokale variabler, i tilfældet her bliver der lavet tre. Todos som skal indeholde igangværende todos, doneTodos som indeholder todos markeret som færdige og sidst expiredTodos, der skal have todos som har overskredet en deadline som en bruger har sat, ved oprettelsen af sin todo.
For at finde ud af om deadline er overskredet, bruger vi et simpelt if-statement til at afgøre det. Det kigger på nuværende dato i millisekunder siden 1970 og sammenligner det med den opgivne deadline. Hvis det er større den nuværende dato er større, så er deadline overskredet og "expired" som er en boolean i databasen som default er false, bliver sat til true.

```
each todo in expiredTodos
                - if (todo.expired) {
                        tr.linethrough.aligncenter
                            td #{todo.title}
                            td #{todo.description}
                            td #{todo.startdate.toLocaleDateString()}
                            td #{todo.deadline.toLocaleDateString()}
                            td #{todo.priority}
                - }
```

Resultatet kan så anvendes i vores pug template i en ny if-statement. I tilfældet her kigger det efter om expired er true, og i så fald vil todo'en blive flyttet her til.

Add to do

For at lave siden, hvor en bruger kan tilføje en to do, har vi først lavet et mongoose Schema, som mapper ud, hvordan vores collection i vores database skal se ud. Her angiver vi, hvilke data der skal være i vores database, og hvilken type data disse skal have. Vores Schema ser således ud:

```
const mongoose = require("mongoose");
var Schema = mongoose.Schema;

const todoSchema = mongoose.Schema({
  title: {
    type: String,
    required: true,
  },
  description: {
    type: String,
    required: true,
  },
  startdate: {
    type: Date,
    required: true,
  },
  deadline: {
    type: Date,
    required: true,
  },
  priority: {
    type: Number,
    required: true,
  },
  userId: {
    type: Schema.Types.ObjectId,
    ref: 'User',
    required: true,
  },
  done: {
    type: Boolean,
    default: false,
  },
});

module.exports = mongoose.model("Todo", todoSchema);
```

I bunden bruges mongoose.model(), som er en funktion som har ansvaret at oprette og læse "documents" fra den tilhørende database. Det første parameter svarer til navnet på den collection der vil blive oprettet i databasen. Mongoose laver automatisk navnet om til lowercase og i flertal, hvilket vil sige, at vi vil få en collection med navnet "todos". Man kan også angive et tredje parameter, hvor man kan angive det præcise navn til, hvad en collection skal hedde. (*Mongoose v6.2.3: Models* n.d.)

Det oprettede Schema bruger vi inde i vores controller, hvor vi laver en funktion, som opretter et nyt objekt ud fra det Schema vi har lavet. I funktionen angiver vi, at til hver af de data der skal i databasen skal have de værdier, som bliver indtastet i vores formular. Dette angives med "reg.body", som netop indeholder de værdier, som kommer fra en formular der er blevet postet.

```
postTodo: async function (req, res, next, user) {
  const db = await mongoConnect.mongoConnect(); // connect to the database
  // new object from the Schema, which says that the values are the values from reg.body which are the datas from the form
  let todolist = new Todoschema({
    title: req.body.title,
    description: req.body.description,
    startdate: req.body.startdate,
    deadline: req.body.deadline,
    priority: req.body.priority,
    userId: user,
  });
  // function that creates documents into the database
  Todoschema.create(todolist, function (error, savedDocument) {
    if (error){
      console.log(error);
    }
    db.close();
  });
},
```

Til siden med "add to do" har vi angivet to routere. Den første er en get router, hvor vi angiver stien og det dokument der skal sendes tilbage, når man requester stien.
Derudover har vi lavet en post router, som bruger funktionen fra controlleren (postTodo) til at poste dataene ind i databasen.
Hertil er det tilføjet en funktion, som tjekker om der er en cookie. For hvis typen af denne cookie er lig med 'undefined' bliver brugeren sendt tilbage til loginsiden, idet de ikke har adgang til systemet.

```
// GET page with "Add to do"
router.get('/addtodo', function(req, res, next) {
    if(typeof(req.signedCookies.User) === "undefined"){
        res.render('login', {
              title: TITLE,
              subtitle: 'Login'
          });
        }else{
        res.render('addtodo', {
            title: 'Add To Do',
            user: req.signedCookies.User,
        });
}
});
```

```
/* POST function that uses a function from controller to post data into the database */
router.post("/addtodo", async function (req, res, next) {
  const user = req.signedCookies.User;
  const userarr = user.split(",");
    if(typeof(req.signedCookies.User) === "undefined"){
        res.render('login', {
              title: TITLE,
              subtitle: 'Login'
          });
    }else{
        controller.postTodo(req, res, next, userarr[1]); // write to dos into db
        res.redirect("/todolist");
        }
});
```

Foreign key

Gennem projektet har det været nødvendigt at gøre brug af en fremmed nøgle. Dette bruges i forbindelse med at koble en to do på en bruger, så denne bliver personligt. Uden en fremmed nøgle ville enhver bruger i systemet se den samme to do, hvilket ikke er hensigten i vores projekt.

I vores tilfælde har vi sat en reference til vores User schema inde i todo schema.

```
userId: {
  type: Schema.Types.ObjectId,
  ref: 'User',
  required: true,
},
```

Udsnit af todoschema hvor fremmed nøgle bruges

Derefter er det muligt for os at indsætte vores to do med reference af vores id som peger på en user id.

```
postTodo: async function (req, res, next, user) {
  const db = await mongoConnect.mongoConnect(); // connect to the database
  // new object from the Schema, which says that the values are the values from reg.body which are the datas from the form
  let todolist = new Todoschema({
    title: req.body.title,
    description: req.body.description,
    startdate: req.body.startdate,
    deadline: req.body.deadline,
    priority: req.body.priority,
    userId: user,
  });
  // function that creates documents into the database
  Todoschema.create(todolist, function (error, savedDocument) {
    if (error){
      console.log(error);
    }
    db.close();
  });
},
```

PostTodo får en parameter med fra routeren i index.js, som holder på brugerens id. Fra routeren henter vi vores cookie som består af en username + id. Ved at sende id´en med ind som parameter ved systemet, ved create, hvilken reference denne står i forbindelse med.

Evaluering af proces

Vi synes det generelt har gået godt gennem projektet. Vi fik uddelt nogle opgaver, som vi hver især har arbejdet med.
Vi valgte at arbejde med Node.js, Express og mongoose, idet vi gerne vil arbejde mere med disse. Men især arbejdet med mongoose var ret nyt for os, som vi lige skulle forstå hvad var, idet vi kun lige havde fået en kort gennemgang af det i brug.
Projektet har haft en passende størrelse, når man sammenligner med den tid vi har haft til det.

Konklusion

Gennem projektet har vi fået opfyldt stort set alle krav der var. Vi har fået lavet et loginsystem, hvor en bruger kan signe up eller logge ind. Hertil har vi fået lavet en admin-side, hvor en admin skal acceptere en bruger, før han/hun kan logge ind.
I forhold til to do listen kan vi konkludere, at vi har fået lavet en side, hvor alle ens to do bliver vist. Hertil er det muligt for brugerne at klikke af, om han/har har udført dem. Hvis en to do er fuldført kan brugere trykke "done", hvorefter to do'en vil blive flyttet ned til de to dos, som er fuldførte. Samtidig vil en to do, som er gået over deadline, blive rykket ned under en anden liste med expired to dos.
Derudover kan vi konkludere at vi har gjort det muligt for en bruger at tilføje flere to dos.
Alt i alt synes vi at vi er kommet godt igennem projektet og dens opgaver.

Referencer

Cookie (2022) available from <https://www.npmjs.com/package/cookie> [24 February 2022]

Larsen, N.M. (2021) *Niels Müller Larsen / EmployeeProject* [online] available from <https://gitlab.com/arosano/employeeProject> [23 February 2022]

Larsen, N.M. (n.d.) *22.3. Architecture of an Express Application II- A Case With Database* [online] available from <http://dkexit.eu/webdev/site/ch22s03.html> [22 February 2022]

Mongoose v6.2.3: Models (n.d.) available from <https://mongoosejs.com/docs/models.html> [22 February 2022]