Christoffer Kalmar, Jon Andersen, Erik Dam, Nickolaj Sørensen
Semester 2, projekt2. Webudvikling 2021

Javascript & Sikkerhed - Data Integration
2. Semester
Andet Projekt

Indledning 2

Problemformulering 3

Metodeovervejelser 3

Research 3

Analyse/Konstruktion 4

Dependencies 4

Server.js 5

Routes 5

Views 5

Models 6

Config 6

Evaluering af proces 7

Afgrænsninger 8

Konklusion 9

Bilag 9

Referencer 15

Indledning

I vores sædvanlige gruppe, Nickolaj, Erik, Christoffer og Jon gik vi igang med planlægning over hvad der skulle skabes for at fuldføre projektet, og vi blev derfor enige om at vi måtte dele opgaven op i to.
1 - der skulle skabes en login mulighed.
2 - der skulle skabes et boglån system.
Det vigtigste for os i denne situation, var at skabe en app der var så virkelighedsnær som overhovedet muligt med vores evner. Det var derfor vigtigt for os at sitet skulle kunne ses online, og at funktionerne skulle fungere korrekt via en etableret online database.
Ud fra disse kriterier satte vi så vores problemformulering.

Problemformulering

Hvordan skaber vi et online site der kan fungere som bog database samt oversigt over bøger tilgængelig til lån i form af et virtuelt bibliotek?

- *Hvorvidt kan vi skabe en login funktion der kan gemmes på en server, der kan kryptere password.*
- *Hvorvidt kan vi skabe et bog-låne system, der kan lukkes når en bruger ikke afleverer bog, og som giver bøde til brugeren ved ikke indleveret dato.*

Metodeovervejelser

Den metode vi valgte at løse opgaven på var gennem brugen af det vi havde til rådighed fra Dkexit og youtube. Da vi havde fundet en step by step guide på youtube tænkte vi det ville være nemmest at få alt forklaret på samme tid som der bliver kodet, var der noget som manglede så gik vi som regel altid ind på nielses side Dkexit og fandt det som var behov for. Et eksempel kunne være kapitel 46. Hvor vi skulle bruge hjælp til sikkerheden, og det var primært for at kunne authenticate brugere.
Vi brugte dkexits kodestykker til en del små ting som der var brug for noget specifikt.

Research

Meget af den research som vi har taget i brug i forbindelse med sikkerheden, samt node er undervisningsmateriale som vi har haft tilgængeligt fra Nielses undervisning via dkexit. Samt har vi haft kontakt til Niels angående møder hvor vi får den hjælp vi har brug for. Derudover har vi blandt andet taget en youtuber i brug ved navn “Web Dev Simplified”[1] der kører online gratis lektioner i bl.a node.js hvor han har lavet en serie hvor han går igennem de forskellige metoder hvor man kan opsætte hjemmesider gennem Heroku og hvordan vi bruger mongodb-Atlas til at gemme vores oplysninger, og lidt andet server baserede ting. Ydermere har vi brugt google og stackowerflow.com til at finde kodeeksempler som minder om det vi skulle bruge til projektet.

Analyse/Konstruktion

Dependencies

- **nodemon**

kører serveren per automatik når man ændrer i dokumentet og gemmer

- **express**

[1] https://www.youtube.com/channel/UCFbNIlppjAuEX4znoulh0Cw

Bruges til strukturen af applikationen. Vi bruger express da vi bl.a gør brug af layouts.ejs modulet til vores html design.

- **express-ejs-layouts**

layouts.ejs bruges i samarbejde med hvad man kalder “partials”. på den måde opbygger vi altså vores applikationsdesign igennem layout, og tilføjer så de ekstra elementer (partials) vi ønsker at inkludere på de individuelle sites i applikationen. dette kombineres alt sammen i en views folder, som så givet brugeren det endelige “look” som vi ønsker at vise brugeren.

- **mongoose**

bruges til håndtering og validering af data. I samarbejde med online mongodb skaber vi en online database hvor applikationen ligger og har sin data.

- **body-parser**

til at tilgå det data vi submitter via forms -> putter det ind i req.body

- **connect-flash**

til at vise primært error messages -> fx ved register, hvis man har et for kort password etc.

- **express-session**

til at oprette en session til hver bruger, at give dem en unik session, hvor indholdet tilpasset derefter. Vi fik det ikke til at blive brugt som vi ville, da vi ikke nåede at implementere lån/aflever.

- **passport**

til at authenticate password -> er passworded rigtigt osv.

- **dotenv**

tillader os at tage variablerne fra vores local environment og smide det op på mongodb atlas + heroku

- **bcrypt**

til at kryptere password.

Server.js

Server.js er selvfølgelig primært der, hvor vi definerer vores routes og requirer vores dependencies, så de kan benyttes i applikationen.

Routes

Routes er selvfølgelig til at definere, hvordan applikationen er opbygget. Ser vi i koden (bilag 3) Har vi et simpelt eksempel på, hvordan vores routes fungerer. I tilfældet er det vores forside og register page. Her fortæller vi applikationen, at den skal tage fat i stien, hhv. "/" og "/register", og på baggrund af det skal der ske noget. I tilfældet er den response der skal ske, at den render en konkret .ejs page, altså "welcome.ejs" og "register.ejs".

Derudover kan vi i routes også definere de funktioner el. lign., der skal sendes med, når vi render eller poster til siden. Et eksempel kunne være vores login router (bilag 5). I eksemplet har vi både en router.get og router.post. router.get sørger som tidligere nævnt for, at render siden. router.post gør i dette tilfælde, at når man ønsker at logge ind, vil den authenticate password'et og emailen, hvis korrekt redirecter den en til siden med alle bøgerne, hvis ikke redirecter den til forsiden og displayer vores flash-meddelse.

Views

definerer vores views, altså vores sider der skal renderes og vises på skærmen. Vi har valgt at benytte os af EJS, nok primært da det var den view-engine vi skulle sætte os ind i, i undervisningen, men også fordi at den for sin vis minder meget om normale html-sider, hvilket gør det lidt mere lige til at arbejde i for vores vedkommende.

Models

Models bruges til at opsætte datamodeller for, i vores tilfælde, author, books og user. Author til når vi skal oprette en author, books til når vi skal oprette en bog og user når brugeren skal registrere sig. Hvis vi kigger på vores user model (bilag 6), sker der nogle forskellige ting. Først og fremmest require den selvfølgelig mongoose, og sætter derefter et nyt schema, som gøres ved at definere end const og bruge mongoose. Altså const X = new mongoose.Schema. Const x bliver derfor til et object, som indeholder key : value pairs. Her definerer man selvfølgelig, hvad skemaet skal indeholde (name, email mv), og hvilke egenskaber de objekter har. f.eks. ved name definerer vi, at det er en string, og at der skal smides et name med,

før at den godtager skemaets data. efter skemaet sætter vi en const, som indeholder skemaet og som vi derfor kan eksportere og bruge til funktioner i routes.

Config

her gemmer vi vores funktioner der skal bruges til at authenticate email og password, samt serialize og deserialize password, så det kan authenticates, at 123456 = hashed og omvendt. Kigger vi på vores passport.js (Bilag 7) sker der flere ting. Først og fremmest requires der de moduler vi skal bruge - User models, passport-local og bcrypt. Dernæst har vi en funktion som tjekker for, om password og email er korrekte.

```
new LocalStrategy({usernameField: 'email'},(email,password,done)=>{
    //match user
    User.findOne({email:email})
```

Her siger den new LocalStrategy (Passport-local) og sætter usernameField til email, med parametrene email, password og done. Dernæst tjekker den via. User.findOne først om e-mailen matcher, som ses nedenfor.

```
.then((user)=>{
    if(!user){
        return done(null,false,{message:'email not registered'});
    }
```

Her siger den, at hvis det ikke er user, returner den false og sender beskeden ‘email not registered’.

```
bcrypt.compare(password,user.password,(err,isMatch)=>{
    if(err) throw err;
    if(isMatch){
        return done(null,user);
    } else{
        return done(null,false,{message: 'password incorrect'});
    }
})
```

hernæst matcher vi passwordet vha. bcrypt.compare med parametrene password, user.password, error og isMatch. Den siger, at hvis der er en error (if (err)), thrower den error. hvis der er match, returner den user, ellers returner den en false med beskeden ‘password incorrect’.

Evaluering af proces

Efter at vi havde fået vores briefing så var det research tid, vi satte os derfor ind i vores discord kanal, hvor vi fik snakket om hvordan vi ville angribe den her opgave. Og vi blev hurtigt enige om at vi ikke ville tage trello i brug, og vi i stedet valgte at bruge noget Scrum. Og den måde vi gjorde det på var at vi gav Jon titlen som Scrum master, og så dernæst udleverede vi lektier til hinanden via en video som vi skulle have set til på mandag. Når vi så mødtes op om mandagen så fik vi snakket og vist hvad vi hver især havde lavet og hvordan vi kunne bygge videre på det. Og i stedet for live-share via visual studio valgte vi så bare at uploade projektet via github hvor vi hver især kunne hente og arbejde videre på. Vi valgte at tage scrum i brug hvor vi aftalte at afholde møder hver dag 1-2 gange dagligt et møde om morgenen og et om aftenen for at se hvis nogen havde fået løst noget, eller om vi havde fundet noget via internettet vi muligvis kunne få brug for. Ellers så var vi fuldt i gang med at arbejde med at løse vores opgaver. Vi oprettede derfor hver især en localhost server som vi kunne bruge til når vi hentede githubben ned, så vi allesammen fik lavet noget slags kodning så vi ikke kun sidder 1-2 mand om kodning og de andre ser på. Det hjalp langt bedre med at få en forståelse for koden vi lavede. Og hvis nu vi gik i stå med noget, så havde vi 3 ekstra som havde en nogenlunde forståelse for det som var på skærmen og ville nemmere kunne hjælpe da vi allesammen teknisk set havde sat det op på samme måde. Og det gjorde at vi havde et par back-up filer liggende

Ud over det så var det en blanding mellem research og møder. Processen i sig selv var lagt mere overskuelig uden brugen af trello da vi følte os lidt mere frie i stedet for at skulle følge et bestemt skema da man aldrig rigtig ved om man kommer bagud eller kommer foran i tidsplanen. Derfor at kunne holde de 2 møder dagligt lå vi på et bedre niveau arbejdsvis.

Afgrænsninger

Mongo db og Heroku

Vi havde utallige mål med opgaven vi ikke nåede grundet utallige problematikker undervejs.
Til dels var der problematik i at få overført fra Eriks computer til heroku og mongodb korrekt via github.
Problematikken lå i at databaseforbindelsen slog fejl lokalt, selvom den egentlig bør gemme det sessions bestemt og køre på localhost:3000 da det er den bestemte rute. Det var som om at når fx. Jon havde arbejdet i koden, og den var uploaded til Heroku, så ville online mongodb ændre nogle af de interne node_modules connection filer, hvilket vi ikke kunne forstå at den gjorde.
(Se bilag 4).
Vej Nickolaj og Jon virkede kode problemløst og blev vist lokalt uden problemer.
Når Christoffer downloadede koden og satte den op, thrower den samme error som som den gør ved Erik.

For yderligere at undersøge dette, klonede vi Jon's Repo over til Erik's repo, hvor Erik så problemløst gennem repoet kunne displays koden på sin Heroku, hvilket vi havde forventet uden problemer. - Men såsnart han pullede koden til sin pc og pushede med minimale ændringer, begyndte fejlmeddelelserne. Fejlen *starter* når man udfører "registrer" ved at udfylde formen efterfulgt af at submitte ved at trykke på "registrer" knappen på register.js siden.
Det vi så formoder her, er at dataene fra input felterne ikke længere bliver korrekt gemt/uploaded til databasen, hvilket gør at undersiden /books ikke længere fungerer.
Vi valgte efter flere timers søgen at opgive fejlsøgningen.
Konklusionen på dette må i virkeligheden have været fejl i egen kode som clashede med det eksisterende kode og derfor ikke kunne læses af databasen. derfor forkastede vi idéen og gik videre så vi kunne afslutte projektet.

Konklusion

På baggrund af projektbeskrivelsens opgaveformulering, har vi så løst opgaven? Vi mangler selvfølgelig at kunne låne og reservere bøger. Havde vi haft en dag eller to mere, havde vi formentlig kunnet implementere det. Vi fik dog løst funktionerne med at oprette og søge i bøger, samt oprette og søge i authors, og lavet en login/register funktion, som hasher passwordet, og som begrænser adgangen til siden, medmindre man er logget ind. Samt gemmer informationer i collection på databasen så disse kunne bruges til de funktioner vi manglede at lave.

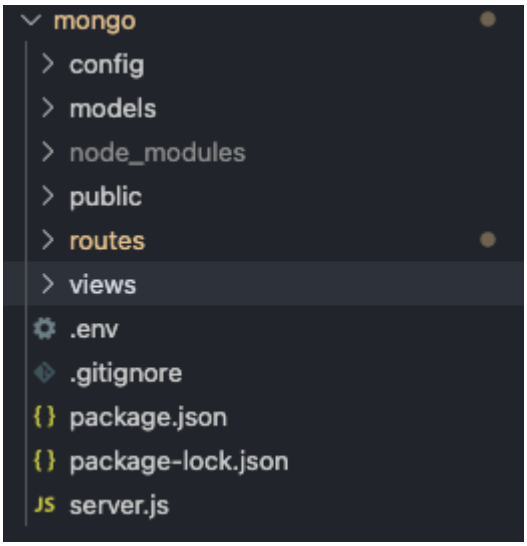
Bilag

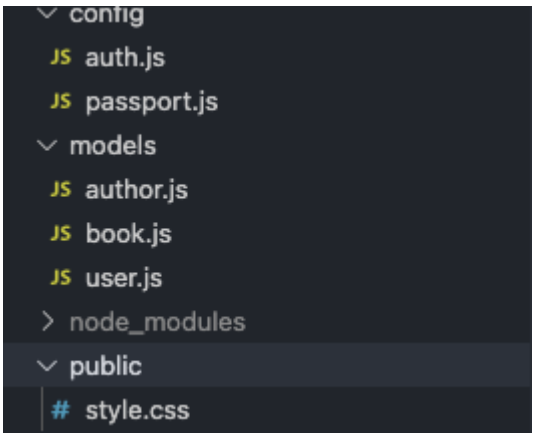
bilag 1

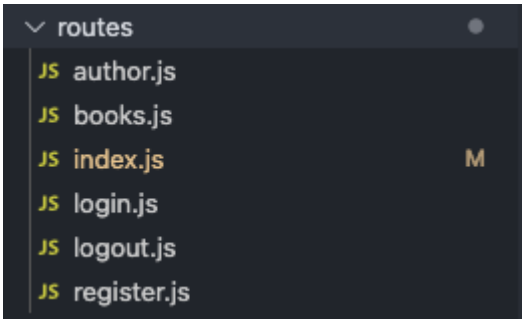
```
if (process.env.NODE_ENV !== 'production') {
    require('dotenv').config()
}

const express = require('express');
const mongoose = require('mongoose')
const router = express.Router();
const app = express();
const expressEjsLayouts = require('express-ejs-layouts');
const bodyParser = require('body-parser')
const flash = require('connect-flash') //
const session = require('express-session') //
const passport = require('passport') //

const indexRouter = require('./routes/index')
const authorRouter = require('./routes/author')
const bookRouter = require('./routes/books')
const registerRouter = require('./routes/register')
const loginRouter = require('./routes/login')
const logoutRouter = require('./routes/logout')

app.set('view engine', 'ejs');
app.set('views', __dirname + '/views')
app.set('layout', 'layouts/layout')

app.use('/public', express.static('public'));
app.use(expressEjsLayouts);
app.use(express.static('public'));
app.use(bodyParser.urlencoded({ limit: '10mb', extended: false}))
app.use(express.urlencoded({extended : false}))
app.use(session({
    secret: 'secret',
    resave: true,
    saveUninitialized: true
}))
```

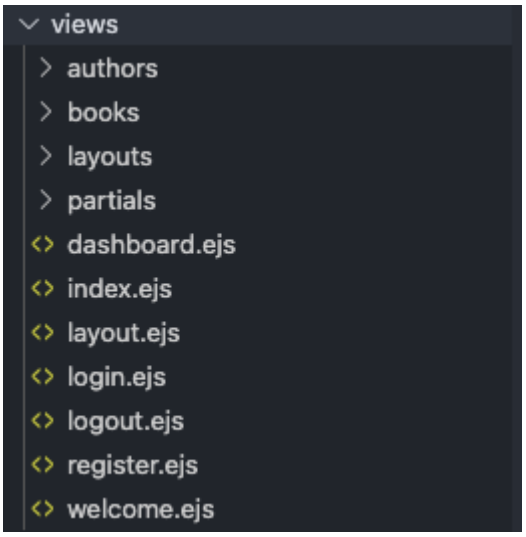
```
app.use(passport.initialize());
app.use(passport.session());
app.use(flash());
app.use((req,res,next)=> {
    res.locals.success_msg = req.flash('success_msg');
    res.locals.error_msg = req.flash('error_msg');
    res.locals.error  = req.flash('error');
    next();
    })
app.use('/', indexRouter)
app.use('/author', authorRouter)
app.use('/books', bookRouter)
app.use('/register', registerRouter)
app.use('/login', loginRouter)
app.use('/logout', logoutRouter)

require('./config/passport')(passport)

mongoose.connect(process.env.DATABASE_URL, {
    useNewUrlParser: true,
    useUnifiedTopology: true })
const db = mongoose.connection
db.on('error', error => console.error(error))
db.once('open', () => console.log('Connected to mongoose'))
app.listen(process.env.PORT || 3000)
```

bilag 2




config
auth.js
passport.js
models
author.js
book.js
user.js
node_modules
public
style.css


routes
author.js
books.js
index.js
M
login.js
logout.js
register.js


views
authors
books
layouts
partials
dashboard.ejs
index.ejs
layout.ejs
login.ejs
logout.ejs
register.ejs
welcome.ejs

Bilag 3

```
const express = require('express');
const router  = express.Router();
const {ensureAuthenticated} = require('../config/auth')
//login page
router.get('/', (req,res)=>{
    res.render('welcome');
})
//register page
router.get('/register', (req,res)=>{
    res.render('register');
})


module.exports = router;
```

Bilag 4

```
npm
 Name 1 email :1@1 pass:111111
MongooseServerSelectionError: connect ECONNREFUSED 127.0.0.1:27017
    at NativeConnection.Connection.openUri (C:\Users\erikh\Desktop\Library\node_modules\mongoose\lib\connection.js:839:32)
    at C:\Users\erikh\Desktop\Library\node_modules\mongoose\lib\index.js:348:10
    at C:\Users\erikh\Desktop\Library\node_modules\mongoose\lib\helpers\promiseOrCallback.js:31:5
    at new Promise (<anonymous>)
    at promiseOrCallback (C:\Users\erikh\Desktop\Library\node_modules\mongoose\lib\helpers\promiseOrCallback.js:30:10)
    at Mongoose._promiseOrCallback (C:\Users\erikh\Desktop\Library\node_modules\mongoose\lib\index.js:1140:10)
    at Mongoose.connect (C:\Users\erikh\Desktop\Library\node_modules\mongoose\lib\index.js:347:20)
    at Object.<anonymous> (C:\Users\erikh\Desktop\Library\server.js:55:10)
    at Module._compile (internal/modules/cjs/loader.js:1063:30)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:1092:10)
    at Module.load (internal/modules/cjs/loader.js:928:32)
    at Function.Module._load (internal/modules/cjs/loader.js:769:14)
    at Function.executeUserEntryPoint [as runMain] (internal/modules/run_main.js:72:12)
    at internal/main/run_main_module.js:17:47 {
  reason: TopologyDescription {
    type: 'Single',
    setName: null,
    maxSetVersion: null,
    maxElectionId: null,
    servers: Map(1) { 'localhost:27017' => [ServerDescription] },
    stale: false,
    compatible: true,
    compatibilityError: null,
    logicalSessionTimeoutMinutes: null,
    heartbeatFrequencyMS: 10000,
    localThresholdMS: 15,
    commonWireVersion: null
  }
}
(node:15944) UnhandledPromiseRejectionWarning: MongooseServerSelectionError: connect ECONNREFUSED 127.0.0.1:27017
    at NativeConnection.Connection.openUri (C:\Users\erikh\Desktop\Library\node_modules\mongoose\lib\connection.js:839:32)
    at C:\Users\erikh\Desktop\Library\node_modules\mongoose\lib\index.js:348:10
    at C:\Users\erikh\Desktop\Library\node_modules\mongoose\lib\helpers\promiseOrCallback.js:31:5
    at new Promise (<anonymous>)
    at promiseOrCallback (C:\Users\erikh\Desktop\Library\node_modules\mongoose\lib\helpers\promiseOrCallback.js:30:10)
    at Mongoose._promiseOrCallback (C:\Users\erikh\Desktop\Library\node_modules\mongoose\lib\index.js:1140:10)
    at Mongoose.connect (C:\Users\erikh\Desktop\Library\node_modules\mongoose\lib\index.js:347:20)
    at Object.<anonymous> (C:\Users\erikh\Desktop\Library\server.js:55:10)
    at Module._compile (internal/modules/cjs/loader.js:1063:30)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:1092:10)
    at Module.load (internal/modules/cjs/loader.js:928:32)
    at Function.Module._load (internal/modules/cjs/loader.js:769:14)
    at Function.executeUserEntryPoint [as runMain] (internal/modules/run_main.js:72:12)
    at internal/main/run_main_module.js:17:47
(node:15944) UnhandledPromiseRejectionWarning: Unhandled promise rejection. This error originated either by throwing inside of an async f
unction without a catch block, or by rejecting a promise which was not handled with .catch(). To terminate the node process on unhandled
promise rejection, use the CLI flag `--unhandled-rejections=strict` (see https://nodejs.org/api/cli.html#cli_unhandled_rejections_mode).
(rejection id: 1)
(node:15944) [DEP0018] DeprecationWarning: Unhandled promise rejections are deprecated. In the future, promise rejections that are not ha
ndled will terminate the Node.js process with a non-zero exit code.
undefined
MongooseError: Operation `users.insertOne()` buffering timed out after 10000ms
    at Timeout.<anonymous> (C:\Users\erikh\Desktop\Library\node_modules\mongoose\lib\drivers\node-mongodb-native\collection.js:185:20)
    at listOnTimeout (internal/timers.js:554:17)
    at processTimers (internal/timers.js:497:7)
```

bilag 5

```
router.get('/', (req,res)=>{
    res.render('login');
})

router.post('/', (req,res,next)=>{
    passport.authenticate('local',{
        successRedirect : '/books',
        failureRedirect: '/',
        failureFlash : true
    })(req,res,next)
    })
```

bilag 6

```
const mongoose = require('mongoose');
const UserSchema  = new mongoose.Schema({
  name :{
      type  : String,
      required : true
  } ,
  email :{
    type  : String,
    required : true
} ,
password :{
    type  : String,
    required : true
} ,
date :{
    type : Date,
    default : Date.now
}
});
const User= mongoose.model('User',UserSchema);

module.exports = User;
```

Bilag 7

```
const User = require('../models/user');
const LocalStrategy = require('passport-local').Strategy;
const bcrypt = require('bcrypt');

module.exports = function(passport){
    passport.use(
        new LocalStrategy({usernameField: 'email'},(email,password,done)=>{
            //match user
            User.findOne({email:email})
            .then((user)=>{
                if(!user){
                    return done(null,false,{message:'email not registered'});
                }
                //math passwords
                bcrypt.compare(password,user.password,(err,isMatch)=>{
                    if(err) throw err;
                    if(isMatch){
                        return done(null,user);
                    } else{
                        return done(null,false,{message: 'password incorrect'});
                    }
                })
            })
            .catch((err)=>{console.log(err)})
        })
    )
```

Referencer

https://www.youtube.com/watch?v=XlvsJLer_No&list=PLZlA0Gpn_vH8jbFkBjOuFjhxANC63OmXM&t=0s

https://www.youtube.com/watch?v=-RCnNyD0L-s&t=750s

http://dkexit.eu/webdev/site/ch46.html

http://dkexit.eu/webdev/site/ch22s03.html