3.Semester Projekt

14 12 2014

David Wroblewski, Daniel Jensen

Casper Hansen, Henrik Ørvald

Contents

1 Arkitektur Model

,

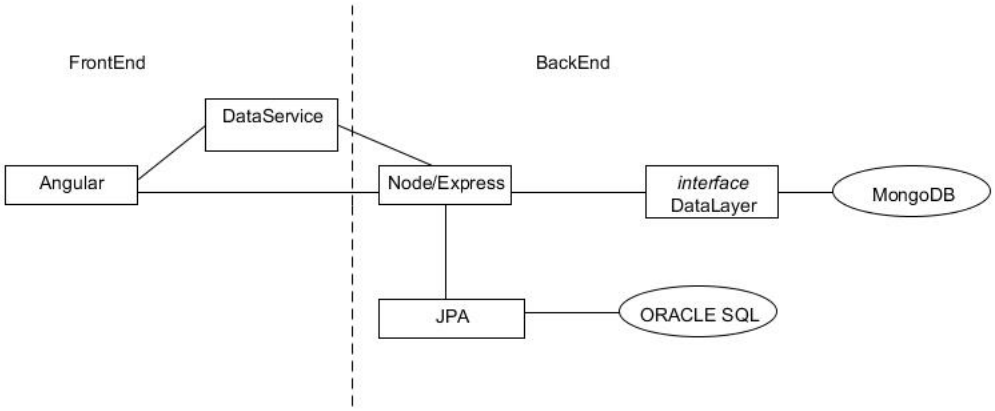

Figure 1: Arkitektur Model

Projektet er bygget op omkring konceptet MEAN STACK. Det vil sige at projektet benytter sig af MONGO DB, en non sql server, Express/Node som lader os tilgå http protokollen i node og bruge og Angular.js som vi bruger til at lave SPA og databinding med. Egentlig er der et lag mere med Bootstrap der lader os lave et lækkert layout, hurtigt og simpelt, så vores bruger får noget pænt og præsentabelt at se på. Angular og vores DataService tager sig af Frontend delen. Angular kan få data som skal vises på siden, enten direkte fra Backenden via skift i url'en, eller via DataService som kan hente information. JPA'en indeholder metoder til at hente password, brugernavn og role. Herefter kan Node sammenligne det hashede/saltede password og give besked om brugeren kan logge ind, samt med hvilken rolle. Mongo DB indeholder generelle informationer om lærer og studerende. Så egentlig er mongo hovedlager for information og JPA/SQL indeholder information om login.

2 Sidediagram for Frontend sider

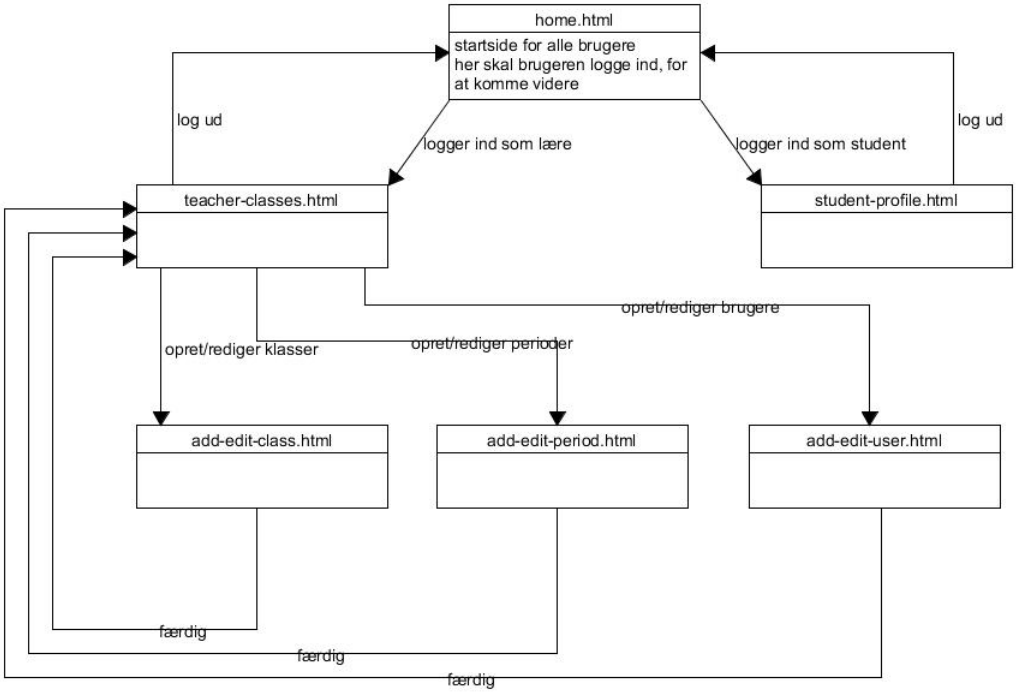

Figure 2: Side Diagram

Når vi kigger på diagramet er det selvfølgelig ment sådan at brugeren kun har index.html filen hos sig, og alt det vi ser ovenover er views der bliver skiftet imellem. Modellen viser måder hvorpå det er muligt at komme rundt mellem siderne og hvilke valg man kan tage på de forskellige sider.

3 Database diagrammer

3.1 SQL

SQL		
NAME	DATATYPE	NULLABLE
USERNAME	VARCHAR	NO
PASSWORD	VARCHAR	NO
ROLE	VARCHAR	NO

Figure 3: SQL Diagram

Tabellen er ret simpel. Vi har valgt at holde det i et table da dette virkede nemmest og mest overskueligt. Der kunne komme udvidelser til programmet hvor det ville være mere brugervenligt for programmører at sende role over i et andet table, da en user så kunne indeholde flere roles. Hverken username, password eller role er nullable da det ikke skal være muligt at oprette sig uden at have alle disse ting.

3.2 Mongo DB

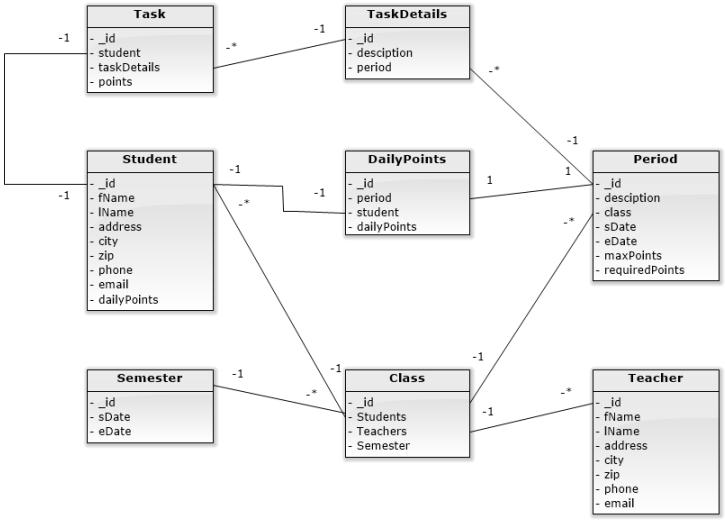

Figure 4: Mongo DB Diagram

Modellen er fra vores Mongo DB og indeholder de nødvendige relationer og informationer vi har brug for til programmet. For at simplificere brugen af Mongo DB gør vi brug af mongoose. Mongoose er at Node.js bibliotek som giver os mulighed for at lave object mapping ligesom ORM. ODM(object data mapping) mongoose laver dataen fra databasen om til javascript objekter.

4 REST interfaces

4.1 Bruger/Server

For brugeren i programmet er det essentielt at kunne finde oplysninger vedrørende brugeren selv, hvor det der imod, for en lærer/admin, vil være vigtigt at kunne finde informationer om klasser/perioder/semestre og brugerne i disse, og kunne editere i dataene. Vi har derfor brug for vidt forskellige funktionalitet ud ad til, men for at kunne finde information om en given bruger, vil der under overfladen være noget overlap i metoderne til at hente disse data. REST API'erne gør brug af interfaces som enten vil tilhøre en forbindelse til vores JPA med tilgang til Oracle SQL databasen, eller en direkte forbindelse, via forespørgelser, til vores MONGO database.

4.1.1 REST API for brugere

Dette API kan tilgås efter at en bruger er logget ind, og først efter dette er sket korrekt.

router.get('/getMyProfile/:userid'

Returnerer en brugers profil ud fra param.userid.

router.get('/getMyClass/:userid'

Returnerer den klasse som bruger med param.userid tilhører.

router.get('/getMyPeriods/:classid'

Returnerer de perioder som tilhører klassen med id params.classid.

router.get('/getMyTasks/:studentid'

Returnerer de tasks som tilhører params.studentid.

Stierne indeholder parametre som request objektet kan tilgå. Vi kalder metoder fra mongoInterface med parametrene. MongoInterfacet bruger herefter en model fra model.js, og bruger en find() funktion på modellen med parameteret, som laver en forespørgelse til MONGO databasen og returnere et JSON objekt svarende til den predefinerede model. Informationen føres herefter hele vejen op til view'et og kan her efter bruges på siden.

router.post('/changePassword'

Fordi dette er et post call er der ingen parametre med i url'en. Dem må vi finde i request body. Efter de er hentet ind bliver et JSON objekt lavet med brugerens nuværende password og brugernavn, og systemet forsøger via JPA'en at logge ind. Herefter opretter vi så et ny JSON, men denne gang med det nye password som er salted og hashet, samt brugernavn og brugerens nuværende password. Dette bliver så igen sendt af sted til JPA'en som udskifter det gamle password med det nye på Oracle SQL serveren.

4.1.2 REST API for lærer/admin

Dette API kan kun tilgås efter en lærer/admin er logget ind, og dette er sket korrekt.

router.get('/getMyClasses/:teatcherid'

Returnere de klasser som en lære er tilknyttet via params.teatcherid

router.get('/getPeriodsByClassId/:classid'

Returnere de perioder der tilhører en klasse med id params.classid

router.get('/getDayliPointsByPeriod/:periodid'

Returnere dayli points for en periode som har periode id params.periodid

router.get('/removeStudentFromClass/:studentid'

Returnere en klasse efter at have fjernet bruger med bruger ID params.studenid

router.get('/getSemesters/'

Returnere alle semestre på mongo DB

Router.get('/getClasses/'

Returnere alle klasser på mongo DB

Router.get('/getPeriods/'

Returnere alle perioder på mongo DB

Router.get('/getStudents/'

Returnere alle studerende på mongo DB

Router.get('/getTeachers/'

Returnere alle lærerne på mongo DB

Router.post('/udpateStudentDayliPoints/'

Opdatere en studerendes dayli points for en periode

Router.post('/addClass/'

Tilføjer en klasse på mongo DB

Router.post('/updateClass/'

Opdatere en klasse på mongo DB

Router.post('/addPeriode/'

Tilføjer en ny periode på mongo DB

Router.post('/addDayliPoints/'

Tilføjer dayli points til mongo DB

Router.post('/updatePeriod/'

Opdaterer en periode på mongo DB

Router.post('/addTaskDetails/'

Tilføjer en beskrivelse af en task/opgave på mongo DB

Router.post('/addTask/'

Tilføjer en ny task/opgave på mongo DB

Router.post('/addUser/'

Tilføjer en ny bruger til mongo DB og opretter også en bruger på Oracle SQL med brugernavn og password

Da der heller ikke her er parametre med i url'en på post kald, bliver vi nød til at finde de ting i body på requestet. Udover at kunne finde information om lærer/admin selv, er det muligt at finde og ændre data for elever, og oprette ny data i form af semestre, perioder,tasks og studerende. Vi vælger stadig at referere en lærer som værende admin/lærer da vi mener at der ligger lidt for mange rettighedder hos en lærer. Iifølge os, ikke bør være muligt for en lærer at oprette nye semestre eller nye elever. Disse funktionaliteter bør ligge på et højere brugerniveau, fx hos en administrator.

4.2 Server/Server

Dette afsnit beskriver JPA'ens måde at håndtere kald fra Node

'/login' , new LoginHandler

POST

Parameteret username findes i body på requestet da det er et post request. Herefter kontrolleres det om der findes en bruger med det username i SQL databasen, og hvis det er tilfældet returnere vi brugeren.

'/user', new UserHandler

POST

Parametrene username, password og role findes i body på requestet da det er et post request. Herefter kontrollers det om der findes en bruger med username, hvis dette er tilfældet returneres en fejl. Ellers opretter SQL en ny bruger.

PUT

Parametrene username, currentPassword og newPassword findes i body på request da dette er et put request. Derefter kontrollers om newPassword og currentPassword er ens, og hvis dette er tilfældet returneres en fejl. Ellers opdateres og erstattes SQL currentPassword med newPassword.

DELETE

Parameteret username findes i body på requestet da dette er et delete request. Derefter kontrolleres det om databasen indeholder en bruger med dette username. Hvis dette er tilfældet slettes brugeren fra SQL, ellers returneres en fejl.

5 Fejlhåndtering

Vi har to steder hvor vi forsøger at håndtere brugers input og kontrollere dem for fejl. Node kan kontrollere om brugerens hashede/saltede password stemmer overens med det på SQL databasen og ellers returnere en fejl til brugeren ('Invalid password or username'). Det andet sted vi håndtere bruger input og kontrollere for mulige fejl er på JPA'en. JPA'en indeholder en hel metoder hvor der sammenlignes data, og hvis disse data ikke stemmer overens så returneres en fejl hele vejen op til bruger. Ellers har vi prøvet at reducere brugeren muligheder for at indtaste data der kan skabe en fejl. Dette er gjort bl.a. ved hjælp af dropdown menuer hvor brugeren kun kan vælge mellem tilgængelige muligheder, eller via radio buttons. Her er det muligt for brugeren at lave fejl ved at indtaste en forkert dag i forhold til hvad brugeren mente, men det er et problem der ikke er til at håndtere for programmet, men ligger hos brugeren selv.

6 Sekvensdiagram

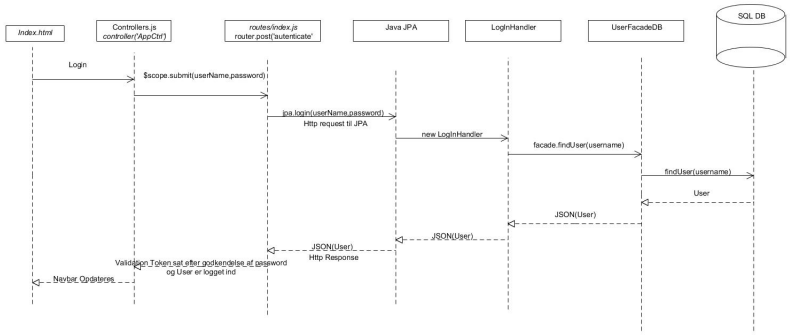

Figure 5: Sekvensdiagram

Fra index.html laver brugeren et login kald med et brugernavn og password. Dette bliver sat på scopet og controlleren kalder submit med begge parametre. Kaldet ryger derefter videre ned i routes, hvor vi kan beder router om at lave et kald til vores JPA. JPA'en modtager kaldet og via dens LoginHandler ber den UserFacade DB om at finde en User fra sql databasen med brugernavnet. Brugernavnet finder den i request body fordi vi har valgt at lave et post request. Det har vi gjort for ikke at skulle have for meget information stående i url'en. SQL databasen svare tilbage med en user, som bliver lavet om til et JSON objekt indeholdende username, et hashet og salted password, og en rolle. Dette føres så op tilbage til router, som, hvis password, fra responset fra JPA, passer med password fra login, giver en et Token alt efter hvilken rolle der hører til brugeren. Her efter opdateres navbaren på index.html og man får options for ens rolle til rådighed.

7 Wireshark

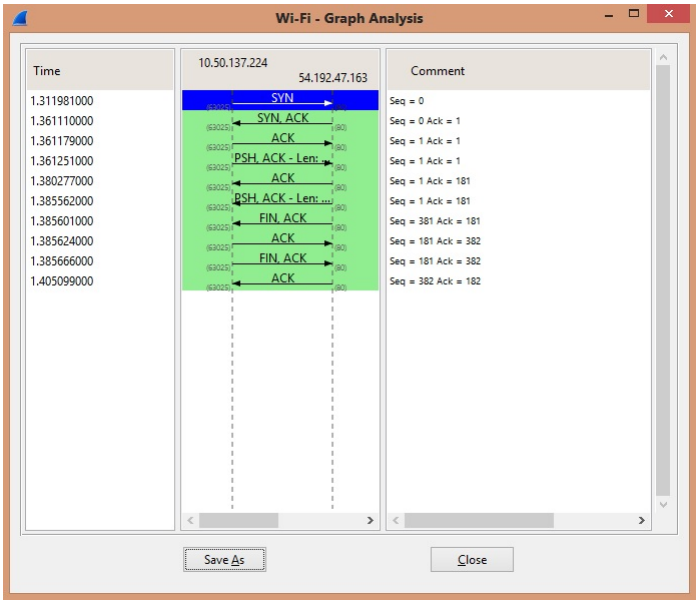

Figure 6: Wireshark Diagram

Vi ser "Three way Handshake" blive udført på de tre første linier. Client sender et SYN til server, og server svare med et SYN,ACK og bekræfter at den har fået et SYN. Client svare herefter med et ACK og bekræfter at client har forstået at server har forstået at der skal oprettes en connection.

8 Genbrug af system dele

JPA og SQL delen er muligt at genbruge i andre sammenhæng, da de kun indeholder login informationer. Disse informationer kunne bruges til at logge ind på et andet skolerelateret system. For at dette kan lade sig gøre skal det nye system selvfølgelig kunne sammenligne de hashede/saltede passwords. Role i sql tabellen kunne muligvis flyttes til et separat table så en person kunne have flere roller. Dele af mongo databasen kunne genbruges til et eventuelt statistik system over elevers fremmøde og karaktere, men dette kunne også være en tilføjelse til det eksisterende system.