Rapport Eksamen PG6300

Da vi fikk utlevert denne oppgaven skjønte jeg med en gang at den var svært omfattende. Oppgaven har vært utfordrende og lærerik, men også noe større enn forventet. Løsningen mangler noen av de spesifiserte kravene men fungerer ellers som tiltenkt.

Løsningen bruker de fleste av komponentene oppgitt som for eksempel MongoDB med Mongoose, React Appen laget ved hjelp av create-react-app, Express, JWT, Bcrypt, react router med flere.

Den største utfordringen har vært å få alt til å snakke sammen og fungere. Jeg har helt klart lagt mest tid og arbeid i backenden og API'et føler jeg er fornuftig satt opp. JWT-tokens fungerer også som tiltenkt og har vært veldig kult å jobbe med.

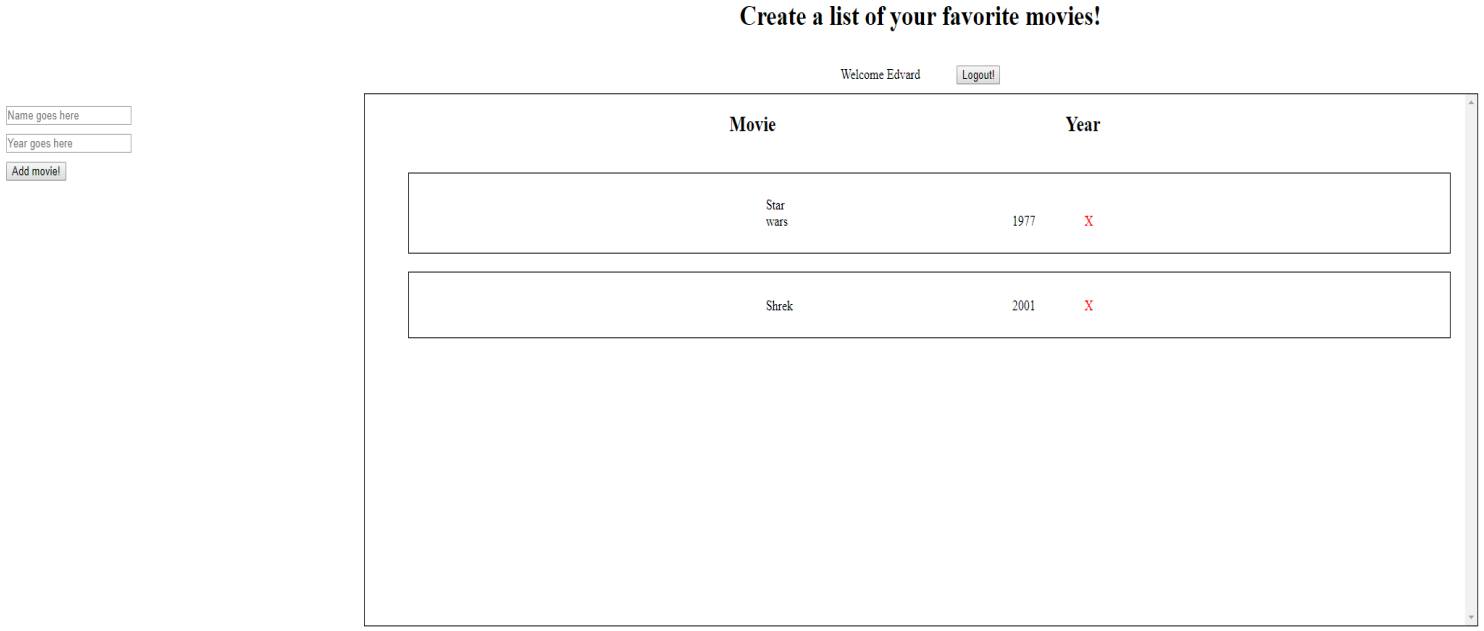
Alle funksjoner utenom inspirasjonssiden fungerer på web. Den mest utfordrende funksjonen var å sortere og hente ut listen basert på den innloggede brukerens ID. Dette tok ekstremt mye tid, men fungerer nå på en fornuftig måte uten for mye koderepetisjon.

For å starte denne applikasjonen:

- Start mongoDB (I mitt tilfelle med kommandoen:
  "C:\Program Files\MongoDB\Server\3.4\bin\mongod.exe")

- Åpne kommandolinjen og naviger til mappen «list-example»
- Kjør kommandoen nodemon/node index
- Åpne et nytt kommndovindu og naviger til /list-example/client/src og kjør kommandoen npm start
- Du vil nå ha en kjørende versjon av prosjektet på localhost:3000
- Gå til localhost:3000/login for å komme til innloggingssiden.

Innloggingssiden



Siden som viser brukerens liste.

Måten databasen er satt opp er at når en bruker legger til en film vil filmen lagres i «movies»-collection. ID'en til denne filmen vil også lagres i collectionen «userlists» sammen med ID'en til brukeren som la til filmen. Hvis to brukere legger til samme film vil ikke filmen bli lagt til to ganger, men brukeren som la til samme film vil heller vi knyttet til ID'en til denne filmen gjennom «userlists». Dette hindrer at collectionen «movies» blir fylt opp av identiske objekter.

Å legge til en film er naturligvis en POST-request og det å hente ut listen etterpå er en GET-request. Igjen så er API'et det jeg er mest fornøyd med og har brukt mest tid på, og jeg føler selv at alle http-verbene er fornuftige og hensiktsmessige.

Fordeler ved å bruke en stack kun med JavaScript:

Den første fordelen jeg umiddelbart tenker på er selvfølgelig at man bruker samme språk i hele applikasjonen. Dette gjør at man som utviklker kan bygge en hel stack, fra database til frontend, uten å lære seg et nytt språk. Det er bedre å være god i ett språk, enn å være ok i to forskjellige. Dette gjør at man er mer effektiv når man koder. Selv om JavaScript kanskje er et unikt språk er det helt klart det mest brukte på web og ved å kun bruke JavaScript blir løsningen portabel og fleksibel.

Hva er et API?

Et API eller Application Programming Interface er sett med protokoller og ruter som gjør at man som utvikler kan opprette kommunikasjon mellom de forskjellige komponentene i løsningen. '

Man bør lage et API når man ønsker en form for abstrahert og enkel kommunikasjon med koden i programmet/applikasjonen. Et API gjør det ikke bare lettere for utvikleren å kommunisere på tvers av komponenter, men lar også tredjeparter benytte seg av koden på en trygg og adskilt måte.

Det man må passe på når man lager et API er at datastrukturen i programmet/databasen din kan bli avslørt og utnyttet. Det er derfor viktig som utvikler å «skjule» dette når man skriver et API.

JSON webtoken vs cookie

HTTP protokollen er som kjent stateless og trenger sikring og noe som holder på en såkalt «state».

Både cookies og JWT gjør dette, men på forskjellige måter.

En fordel med tokens er at det fungerer bra på tvers av domener. Mange websider gjør kall til flere forskjellige domener og tokens fungerer veldig bra til dette. En ulempe er at tokens krever noe mer programmering implementasjonsmessig, men er til gjengjeld mer fleksibelt.

REST

REST er en slags «oppskrift» webtjenester bruker for å snakke sammen på internett.

REST er et lettvektig og lite ressursintensiv måte å sende HTTP-forespørsler.

REST er derfor svært populært og brukes mer og mer. Løsninger som bruker dette kalles ofte «RESTful Web services»

Noen fordeler ved å bruke nivå 2 er at i stedet for å kun bruke POST-requests deler man det opp i flere typer requests med forskjellige funksjoner, for eksempel GET, POST, DELETE og PUT. Dette gjør det er lettere å tildele riktige http responskoder, noe som gjør at det er lettere å se hva som har gått galt og deretter fikset dette.

Automatisert testing

Det å teste koden for eksempel ved hjelp av JEST gjør at man kan utvikle sikrere løsninger, raskere. Automatisert testing gjør hele utviklingsprosessen lettere og mer sømløs da man med en gang vet hva som har gått galt.

Websockets

Websockets er måte å oppnå «real time» kommunikasjon mellom klient og server. Websockets brukes ofte i applikasjoner hvor raskere kommunikasjon er nødvendig. For eksempel spill og livechat.

Det bør derimot ikke brukes til noe annet enn realtime kommunikasjon da http har svært mye feilhåndtering og er mye mer utbredt og støttet. En parallell man kan dra er til UDP/TCP protokollene hvor UDP har liten grad av feilhåndtering og respons og TCP har høy grad av feilhåndtering og gir beskjed om dataen ikke ble levert riktig. Websockets minner da om UDP mens HTTP minner om TCP

Avslutning

Alt i alt har dette vært en ekstremt krevende oppgave og har tatt mye tid. Jeg har i gjengjeld lært mye om et område jeg var helt blank på. Jeg skulle gjerne ha sett at inspirasjonssiden ble implementert og skrevet automatiserte tester. Jeg skulle også ha implementert feilhåndtering på backend i større grad og jeg håper jeg har dekket de groveste. Mobilapplikasjonen med react-native ble heller ikke gjort, dessverre. De andre oppgavene viste seg å være en så stor utfordring at tiden ikke strakk til. Igjen, så beklager jeg dette. Selv om løsningen har noen mangler mener jeg at de komponentene som fungerer, fungerer godt og er fornuftig satt opp.

Takk for et lærerikt og utfordrende kurs.