Copenhagen Business school

https://github.com/TaoHaIt/ProgEksamen

Copenhagen Business school

Programmering og udvikling af små databaser

Eksamensrapport - reeksamen februar 2022

Vejledere: Nicolai Frost Jacobsen og Regitze Sdun

Indholdsfortegnelse

[Indledning: 2](#_Toc95412220)

[Mappestruktur: 2](#_Toc95412221)

[Kravspecifikation og arkitektur: 3](#_Toc95412222)

[Data oversigt: 7](#_Toc95412223)

[Løsningsovervejelser: 8](#_Toc95412224)

[Procesevaluering: 9](#_Toc95412225)

[Konklusion: 10](#_Toc95412226)

[Litteratur- og kildeliste: 10](#_Toc95412227)

# Indledning:

I denne rapport vil der fremlægges hvordan eksamensprojektet er blevet udarbejdet og hvilke metoder der er blevet gjort brugt af, samt hvordan de relaterer til undervisningen. Opgavens formål var at udarbejde en webapplikation i stil med Den Blå Avis.

Rapporten vil komme ind på projektets struktur, gennemgang af diverse kravspecifikationer samt dets arkitektur. Unit-testing og data oversigt vil også blive gennemgået, og til sidst vil der inkluderes løsningsovervejelser samt en procesevaluering.

# Mappestruktur:

Projektet er bygget op omkring en MVC softwarearkitektur. MVC (Model-view-Controller) er en klassisk model der bruges i udviklingen af webapplikationer, og er en oplagt model at bruge i de mere simple projekter.

Med denne mappestruktur bliver koden opdelt i tre dele, hver bygget til at håndtere et bestemt aspekt og med hvert deres formål. Dette gør det nemt at fokusere på én specifik del af projektet ad gangen. I views-mappen bliver den del af koden som skal kunne ses og interageres med placeret, altså HTML-siderne og de javascript-filer som giver dem liv. I controller bliver mine endpoints placeret, som kommunikerer direkte med mine views. I helpers placerer jeg en database, hvori det gøres muligt at gemme og slette filer.

I databasen er det vigtigt ar require(”fs”), så der kan arbejdes lokalt med filer. Filerne bliver herefter opbevaret lokalt i en datamappe, hvori der placeres JSON-filer.

Models-mappen er hvor bruger og produkter bliver defineret med constructorer, der definerer hvilke elementer produkter + brugere skal indeholde. Disse filer eksporteres også til controllers, så de kan bruges direkte når der laves endpoints.

**HTTP-request:**

Hyper-Text-Transfer-Protocol gøres brug af for at serveren og klienten skal kunne tale sammen. De kommunikerer ved at udveksle forespørgsler og svar protokoller. De http-request som gøres brug af i projektet, er GET, POST, PUT, DELETE.

GET bruges til at læse og hente oplysninger fra en kilde. Post bruges til at skabe nye elementer (som f.eks. oprette en ny bruger). PUT bruges til at opdatere data. Delete bruges til at slette data.

# Kravspecifikation og arkitektur:

|  |  |
| --- | --- |
| **Krav** | **Opfyldt/ikke opfyldt:** |
| 1. App’en skal tillade en bruger at oprette en profil | Opfyldt |
| 2. App’en skal tillade en bruger at slette sin egen profil | Opfyldt |
| 3. App’en skal tillade en bruger at opdatere sin egen profil | Ikke opfyldt |
| 4. App’en skal tillade brugeren at logge ind | Opfyldt |
| 5. App’en skal tillade at hvis en bruger er logget ind kan de forblive logget ind | Opfyldt |
| 6. App’en skal gøre det muligt at en bruger kan oprette en vare med en varekategori, billede og pris. | (Opfyldt) \*Mangler billede |
| 7. App’en skal gøre det muligt at en bruger kan slette en vare du har oprettet | Ikke opfyldt |
| 8. App’en skal gøre det muligt at en bruger kan opdatere en vare du har oprettet. | Ikke opfyldt |
| 9. App’en skal gøre det muligt for en bruger at logge ud | Opfyldt |
| 10. App’en skal kunne vise en bruger en tabel over de varer, som de har til salg | Delvis opfyldt |
| 11. App’en skal have en kategori menu hvor man kan klikke og få præsenteret alle varer til salg inden for en given kategori. | Ikke opfyldt |
| 12. Lave udførlige unit tests til 1 krav (vælg mellem krav 1-9). Dette betyder at alt funktionalitet for kravet skal være unit testet. | Opfyldt |

**Krav 1:**

Når en bruger går ind på forsiden, vil de, hvis de ikke allerede er oprettet, blive sendt til /register. Her indtastes så bruger informationer der bliver sendt til lokal lagring når det bliver submitted. Når brugeren er oprettet, kan de så starte en session på siden, og tilgå alle sidens funktioner.

Måden det foregår på rent teknisk er med en POST-request. De felter som brugeren skal udfylde for at oprette sig, er linket sammen med den constructor som blev lavet til en bruger (email, password, userID). Filerne bliver behandlet med FS i min /helpers/database. Med JSON.stringify bliver brugeren sendt til data/userdata.json, og lagret lokalt.

**Krav 2:**

Når brugeren er logget ind, vil de komme ind på en ”startside”, hvor sidens muligheder befinder sig. Dette inkluderer blandt andet en ”Delete your userID”. Hvis brugeren trykker på knappen, vil der blive sendt en DELETE-request der leder efter brugeren ud fra de nøgleord som er knyttet til dem (email). Mere specifikt, vil Javascript søge efter et array der matcher. Hvis Javascript finder et match, vil det blive slettet fra den lokale database med LocalStorage.removeItem.

**Krav 3:**

Dette krav blev ikke opfyldt.

**Løsningsovervejelse:**

For at få kravet opfyldt skulle der gøres brug af en PUT-request, som skulle overskride den gemte data med ny data, som skulle hentes via et user-input.

Ved at bruge JSON.parse skulle programmet lede efter den eksisterende brugerdata, og så overskride den med de nye værdier. Værdierne som skulle overskrides ville være password og/eller userID.

**Krav 4:**

Når en bruger har oprettet en profil, vil de blive sendt direkte videre til login.html, hvor de skal indtaste deres brugerinformationer. Det som de indtaster bliver så transformeret til et JSON objekt, og med JSON.stringify skal objektet så ændres til et string. Derefter kan serveren læse stringet, og søge efter værdierne i /data/userdata.json. Hvis værdierne findes logges brugeren ind, hvis de ikke findes kommer fejlkoden ”User data was not found”.

Dette krav bliver opfyldt ved brug af et POST-request. Værdierne som søges efter er email og password.

Herudover gøres der brug af if/else statements. If user is found …. Returneres en http 200 statuskode. Else, hvis email og password ikke passer sammen, returneres et http 401 statuskode. Hvis email og password ikke findes i databasen, returneres et http 404 statuskode.

**Krav 5:**

Når en bruger er logget ind, vil de blive gemt i Local Storage, hvilket er weblagring. For at opfylde kravet skal der tilføjes lidt til login.js

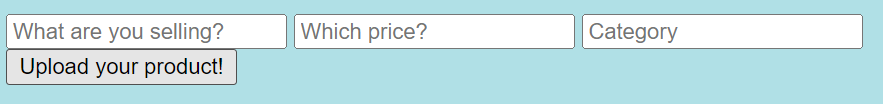
Et billede, der indeholder tekst

Automatisk genereret beskrivelse

Hvis ”user” (email, userID) findes i localstorage, så bliver man automatisk logget ind. Hvis man ikke findes, vil man automatisk blive sendt til login.html.

**Krav 6:**

Når brugeren er logget ind, vil de have mulighed for at uploade deres egne produkter. Her vil de skulle indtaste tre værdier, product, price, productID(kategori), som er dem som matcher product constructor værdierne.



Produktet uploades med et POST-request. Der gøres brug af /helpers/database, så produktet kan gemmes i /data/productdata.json.

Når produktet er uploadet modtager brugeren en http 200 statuskode.

**Første begrænsning**:

Når produktet er uploadet, er det ikke knyttet til den bruger som har uploadet det. Produkterne vil derfor kunne opdateres eller slettes af andre brugere, samt man ville i teorien kunne købe og betale for sig eget produkt.

Dette kunne løses ved at give produkterne et unikt ID som kunne lagres på LocalStorage sammen med brugerens userID. For at kunne opdatere eller slette produktet, skulle dette produkt ID så indtastes.

**Anden begrænsning:**

Det er ikke muligt at uploade et billede sammen med produktet.

Dette skulle være blevet oprettet med express.static, og billederne skulle gemmes i en mappe ekskluderet fra resten af produkt data’en.

**Krav 7:**

Dette krav blev ikke opfyldt.

For at få kravet opfyldt, skulle der gøres brug af en DELETE-request. Kravet kunne være blevet opfyldt med en metode som ville minde særdeles om hvordan en bruger uploader et produkt. Produktet skulle så slettes ud fra dets produktID.

**Krav 8:**

Dette krav blev ikke opfyldt.

Til dette krav kunne der være blevet gjort brug af et PUT-request, som skulle opdatere produktets data. Fremgangsmetoden i krav 4 og 8 havde været den samme.

**Krav 9:**

Når en bruger er logget ind og gemt inde på LocalStorage, skal de selvfølgelig også have muligheden for at logge ud igen.

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

Der tilføjes en ”Sign out” knap, som når trykkes fjerner brugeren fra LocalStorage. Dermed sluttes deres session, og brugeren sendes til login.html.

**Krav 10:**

Her henter jeg produktdata i min JSON-fil, og sender dem til frontend ved at bruge fetch. Herefter designes tabellen, og der laves et forEach loop, som returnerer productName, price, og kategori. Et billede, der indeholder tekst

Automatisk genereret beskrivelse

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

**Krav 11:**

Dette krav blev ikke opfyldt.

For at opfylde kravet, skulle der ligesom i krav 10 gøres brug af fetch. Her skulle brugeren så have mulighed for at søge efter en kategori og fremkalde en tabel, som matcher det produkID(kategori) som der søgtes efter.

**Krav 12:**

Der testes om siden tillader en bruger at oprette en profil. Først installeres mocha, chai, og chai-http. Herefter skal



Tilføjes til package.json.

Mocha er et Javascript test framework, som kan anvendes sammen med Chai.

Der laves en ny mappe ”tests”, hvori der tilføjes en unit.js fil. Herefter importeres server.js.

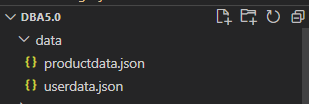
Testen skal i bund og grund kunne oprette en ny user, så der laves en POST-request og herefter hentes /users/create fra serveren.

Når testen lykkes, sendes en http 200 statuskode, og en ny bruger bliver tilføjet i JSON-databasen.

# Data oversigt:

Alt data gemmes lokalt på serveren i flade filer. De bliver gemt i henholdsvis to JSON-filer, en til brugere og en til produkter. Disse filer kan blive brugt med ”fs”. Med JSON kan dataene hurtigt blive eksekveret.

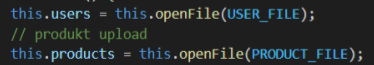
Derudover gøres der brug af Local Storage for at brugeren kan logge ind og ud.



For at sidens client kan sende data-input til serveren og gemme den, placeres de to JSON-filer i denne data-mappe. Der oprettes en /helpers/database, hvor der hentes ”fs” så der kan arbejdes lokalt med filer på computeren. Der skal altså kunne åbnes og gemmes filer. Der laves en stig i database.js.

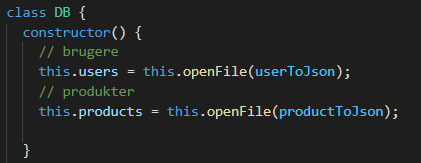


Herefter defineres en product og users fil, så der kan arbejdes med data der relaterer til varer og brugere.



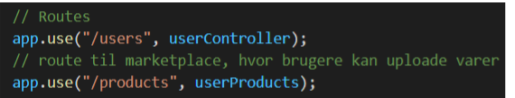
Der laves en saveFile funtkion med to input, fileName og contentString, hvor der bruges fs.writeFileSync().

Når data bliver sendt fra websiten bliver det sendt afsted som en string. Derfor bruges der JSON.parse(), så stringen kan omdannes til Javascript objekt.



openFile og saveFile kan der nu gemmes og åbnes filer, og der er dermed funktionalitet for databasen. Det skal nu bruges til at kunne slette og oprette produkter samt brugere.

Dette gøres ved at bruge en push funktion til at tilføje et produkt eller en bruger til et array.



Derefter er det at oprette router til server.js, for at få hele MVC mappen til at snakke sammen.

# Løsningsovervejelser:

I dette afsnit vil der diskuteres løsningsovervejelser, begrænsninger, samt begrundelse for valgte fremgangsmetode.

Den mest iøjnefaldende begrænsning i dette projekt, er at skulle gemme alt data lokalt i JSON-filer. Da projektet kun er ment til at være et udkast til en mere omfattende hjemmeside er det fint at gemme data lokalt, men i længden ville det være uholdbart med sådan en begrænsning på hvor meget data der kan gemmes. I udviklingsfasen, hvor der kun er minimal trafik på siden, er det dog en fin fremgangsmetode, da det gør det muligt at teste de forskellige elementer og funktionaliteten på siden, samt hurtig og sikkert at kunne gemme lidt data. Samtidigt gør det også, at udvikleren har fuldt kontrol over tilgængeligheden og ikke ville skulle bekymre sig synderligt om data-leaks.

Lokal lagring kontra cloud lagring er derfor også et væsentligt læringspunkt at kunne tage med fra opgaven.

I Javascript projekter skal man være opmærksom på at undgå et ”Callback hell”, hvor ens kode er så proppet med if/else statemens, at man kan miste overblikket over sine loops.

Ved opstarten af projekter som dette, skal der selvfølgelig også tages stilling til hvilke mappestruktur man vil benytte sig af. I dette projekt var det oplagt at benytte sig af Models-views-Controllers strukturen, da den er begyndervenlig og relativ simpel. Uden den rigtige struktur kan projektet hurtigt blive uoverskueligt, og man kan komme til at mixe koden til frontend og backend sammen.

At have en klar vision for hvordan ens struktur skal se ud er derfor også et læringspunkt at tage med fra projektet.

# Procesevaluering:

Dette afsnit vil fokusere på de udfordringer som projektet stødte på, hvordan de blev håndteret og hvordan de i fremtidige projekter kan undgås.

1. Opstarten

Processen for hvordan man effektivt starter et nyt projekt op, er noget som kræver erfaring og planlægning. Første gang jeg skulle lave projektet ”spildte” jeg en masse tid på at starte halve løsninger op, for efter at investeret tid og energi i det, fundet ud af at det ikke var/ville være brugbart i længden. Med bedre planlægning og research før projektstart, kan dette undgås. Anden gang jeg skulle starte projektet op (reeksamen), vidste jeg præcis hvordan jeg skulle håndgribe det for at nå lige så langt som sidst, og kunne derfor gøre det på langt kortere tid.

I fremtidige projekter vil jeg have fokus på at lave grundigt feltarbejde før projektstart, og være forsigtig med at inkludere elementer som i sidste ende er for svære at få implementeret.

1. Git og Github

I første forsøg af projektet, benyttede jeg mig overhovedet ikke af hverken Git eller Github. Jeg sprang det over da jeg havde for meget bøvl med at få det til at fungere, og så ikke vigtigheden i at inkludere det. Git og Github er fundamentalt for ethvert professionelt kodningsprojekt, og er derfor et fokuspunkt jeg har haft den her gang. Både så projektets proces er tydeligt for dem som skulle kigge nærmere på det, samt for at sikre mig at intet skulle gå tabt.

1. Debugging

Et stort problem jeg har mødt i dette projekt, har været og få debugget koden. Dette har resulteret i at der er blevet hoppet rundt imellem kravene og efterladt en masse løse ender, som i sidste ende gør fejlene mere uoverskuelige.

Ifht debugging er udviklerkonsolen et værktøj jeg vil bruge aktivt fra start af.

1. Forventningsoverensstemmelse

I opstarten af et projekt er det nemt at komme til at blive overambitiøs. Det er vigtigt at have en vision og mål for hvad man gerne vil konkludere, men mindst lige så vigtigt at være fleksibel og klar til at tilpasse projektet, hvis det bliver tydeligt at der er elementer som ikke kommer til at fungere.

# Konklusion:

Denne rapport har beskrevet hvordan reeksamensprojektet er blevet lavet. Mappestrukturen er blevet uddybet, med dets styrker og svagheder. Alle 12 krav er blevet gennemgået, i dem som er blevet løst er der beskrevet hvad der er blevet gjort, og i dem som ikke er blevet opfyldt er der beskrevet hvad der kunne være gjort.

Data bliver opbevaret på JSON-filer samt localStorage.

I løsningsovervejelserne er der blevet argumenteret for hvordan problematikker der opstod undervejs kunne være blevet løst, og i procesevaluering bliver der reflekteret over hvordan et tilsvarende projekt i fremtiden kan udarbejdes.

# Litteratur- og kildeliste:

*BINTO1064U.LA\_E21: Billeder i Express server og formularer. (2021, November 24). YouTube. Retrieved November 28, 2021, from* [*https://www.youtube.com/watch?v=7aTGmAo2EnM*](https://www.youtube.com/watch?v=7aTGmAo2EnM)

*BINTO1064U.LA\_E21: Godkendelsesopgave 3. (2021, November 6). YouTube. Retrieved November 6, 2021, from* [*https://www.youtube.com/watch?v=SoyJmZ7KexQ*](https://www.youtube.com/watch?v=SoyJmZ7KexQ)

*BINTO1064U.LA\_E21: Skrive/læse JSON og import af filer. (2021, November 24). YouTube. Retrieved November 29, 2021, from* [*https://www.youtube.com/watch?v=ltyrpWwHzSY*](https://www.youtube.com/watch?v=ltyrpWwHzSY)

*BINTO1064U.LA\_E21: Testing af Express server. (2021, November 24). YouTube. Retrieved December 9, 2021, from* [*https://www.youtube.com/watch?v=-bX0G1X\_Lyc*](https://www.youtube.com/watch?v=-bX0G1X_Lyc)

*BINTO1064U.LA\_E21: Upload af billeder med Fetch og hente ned i tabeller. (2021, November 30). YouTube. Retrieved December 5, 2021, from* [*https://www.youtube.com/watch?v=caHwtvUoJy0*](https://www.youtube.com/watch?v=caHwtvUoJy0)

*Dubray, J. J. D. (2016, February 3). Why I No Longer Use MVC Frameworks. InfoQ. Retrieved December 1, 2021, from* [*https://www.infoq.com/articles/no-more-mvc-frameworks/*](https://www.infoq.com/articles/no-more-mvc-frameworks/)

*Hansen, S. M. G. (medstuderende), Programmerings forelæsning - loginsystem, Retrieved December 4, 2021, from https://studentcbsmy.sharepoint.com/personal/ayka21ab\_student\_cbs\_dk/\_layouts/15/onedrive.aspx?id=%2Fpersonal%2Fa yka21ab%5Fstudent%5Fcbs%5Fdk%2FDocuments%2FRecordings*

*Haverbeke, M. (2019). Eloquent JavaScript [E-book]. Amsterdam University Press.*

*Using Fetch - Web APIs | MDN. (2021, October 13). MDN Web Docs. Retrieved December 2, 2021, from https://developer.mozilla.org/en-US/docs/Web/API/Fetch\_API/Using\_Fetch*