Mustafa Mehdi - Notes

**Databas struktur**

Jag började med att tänka ut vilka tables applikationen ska ha för att fungera enligt instruktionerna. Från början var det inte jätte lätt då det blev snabbt komplicerat i huvudet, så jag såg till att hitta en bra design-verktyg som har support för att sätta upp visuella mock-ups till relationella databas, vilket blev https://dbdiagram.io/ som var ganska enkel att förstå och komma in i.

I huvudet tänkte jag kanske ha 3 tables där jag skulle ha ett värde för hard-delete i ena basen och bibehålla soft-deletes i huvud-dokument table. Men det insåg jag var lite onödigt, samt fått det bekräftat efter att det nämndes i en lektion där vi sa att det kanske inte är lurt att skapa en logg-table. Då blev det en två-table lösning där vi har användare och dokument, varav användare har UUID som primary key och går in som foreign key i dokument-table.

* users
  + userId (Primary-key)
  + userName
  + userEmail
  + password
  + createdAt
* documents
  + documentId (Primary-key)
  + userId (foreign-key)
  + title
  + documentBody
  + createdAt
  + lastUpdated
  + isDeleted

**Backend-routes process**

Efter att ha bestämt för hur databas skulle se ut på ett ungefär, hade jag påbörjat sätta upp mina routes och försökt tänka till och testat så att de olika metoderna skulle försöka hålla repetition av kod minimalt.

Users routen var ganska enkel och staright-forward då vi hade använt likadan struktur i tidigare projekt, dock med NoSQL, men logiken var densamma och därmed enkel att applicera på en MySQL upplägg. Dokument routen däremot var lite mer komplex, just för att det krävs visualisering kring de olika funktionerna jag hade planerat till. Efter att ha byggt upp routsen, började jag testa med en .rest fil där jag fick tweaka till några endpoints och lägga in rätt metod som till exempel PUT och UPDATE, där det inte var så enkelt att se var de skulle in och till vilket ändemål, men sen läste jag på lite och kom fram till att UPDATE används mer till enstaka uppdateringar där man kanske skulle ändrat namn på användaren till exempel, men PUT däremot är större uppdateringar som en hel dokument, där jag la in den i uppdatering av doc routen.

När jag såg att routsen fungerade hyfsat, var jag osäker om jag hade logiken till hundra procent korrekt, men jag bestämde då för att påbörja uppbyggnaden av client. Den första funktionen jag ville få till var login och sign-up av user, för att sedan kunna spara in userId då det är en viktig parameter för alla andra funktioner. Efter det kom jag till dokument skapandet, det tog lite tid att sätta sig in i tinyMce och det var lite chockande att det är så enkelt att kunna nyttja en extern tjänst, samtidigt var det hyfsat komplicerad att tänka på hur värden ska nyttjas av mina end-points, efter att ha jobbat på tinyMce ett litet tag så fick jag ganska många buggar när det skulle initaliseras, eftersom jag har en knapp som ska initalisera editorn, så blev det att editorn försvann vid ett andra klick, jag förstod då att det hade mer med init logiken att göra, men för att vinna så mycket tid som möjligt fick jag leta efter en editor som passade mig bättre. Editorn jag bestämde mig för att använda var Froala, vilket var ganska enkelt att förstå logiken bakom, samt att det var väldigt bra dokumenterat.

Efter att ha fått editorn att komma upp korrekt, så anpassade jag funktionaliteten i toolbar för att det ska passa till innehållet, jag såg att det fanns även val för att ladda upp bilder vilket jag ville försöka mig på, medveten om att det kommer krävas lite tweak i antingen databasen eller måttan att spara innehållet på. Efter att ha satt upp basic funktionalitet i dokument routsen för att kunna spara, sedan hämta dokumentet som hör till inloggade användaren, kom jag på att det blev nämnt i en tidigare lektion att vi skulle kunna använda oss av "blob" filformat för att kunna spara bilder och andra typer av filer som binär data. Jag gav det ett försök och lyckades spara innehållet som blob, men problemet kom när jag skulle försöka hämta ut blob datan och översätta det till innehåll som webbläsaren kan tolka. Efter att ha försökt med det ett tag och även en snabb försök i handledning i hopp om att det var en enkel fix, så gick det inte utan att fastna.

Så jag bestämde för att köra in texten som vanlig sträng in i databasen och då lyckades jag spara in det som HTML sträng och skriva ut texten som InnerHTML. När det fungerade för mig att spara dokument och testa med flera användare och det var ganska så straight forward med tanke på att datan hör ihop med parametrar.

När det var dags för att bygga upp ändring av dokument, stötte jag på lite problem eftersom det var lite trix med att kunna skicka in data till samma dokument, i ju med att editorn har ett eget fält som datan hämtas ifrån, tog det lite tid att få till de korrekta elementen som ska läsas in. Det trodde jag var den svåra delen tills jag kom till att radera dokument, där bestämde jag för att lägga in en soft-delete för att testa, och via endpointen i backenden funkade väldigt bra, men sen så hade jag byggt routen som en one-way funktionalitet, där den bara raderar dokument och inte vice-verca.

Anledningen till att jag ville göra endpointen generisk var helt enkelt för att kunna lägga till en liten funktionalitet som tillåter användaren att ångra efter att ha raderat. Till slut fick jag det att fungera och lyckades skicka rätt värden till de olika funktionerna. Dock stötte jag på lite buggar, där ibland dokumentsidan renderas för snabbt, innan användaren hinner bekräfta radering av dokument, eller när användaren ångrar radering, då måste man ladda om sidan för att det ska synas.

För övrigt så blev det så mycket styling som jag orkade med, känner mig all in all ganska så nöjd med uppgiften jämfört med förra, då märkte jag hur mycket kunskap som jag fick med mig därifrån, och att kunna göra rätt från början. För en sak jag märkte där var att när jag var vid slutet, så var det lite försent att rätta till viss logik och funktionalitet utan att behöva bygga om hela sidan.

**Sammanfattning av varje end-point:**

**USERS-Route:**

* **GET /users**
* Hämtar alla användare från databasen.
* Returnerar en JSON-array med användar-ID:n.
* **GET /users/profile/:userId**
* Hämtar användarinformation (användarnamn och e-post) baserat på användar-ID.
* Returnerar användarinformation som JSON-objekt.
* **POST /users/add**
* Lägger till en ny användare i databasen.
* Genererar ett UUID för användar-ID och krypterar lösenordet.
* Kontrollerar om e-postadressen redan finns i databasen.
* Om inte, lägger till användaren och returnerar användarnamnet.
* **POST /users/login**
* Kontrollerar användarens inloggning genom att jämföra med det krypterade lösenordet.
* Returnerar användarinformation om inloggningen var en success.
* **PATCH /users/updateprofile/:userId**
* Uppdaterar användarens användarnamn baserat på användar-ID.
* Returnerar en framgångsmeddelande om uppdateringen var en success.
* **PATCH /users/editpass/:userId**
* Kontrollerar det existerande lösenordet innan användaren uppdaterar det till det nya lösenordet.
* Returnerar ett success message om rätt lösenord var skrivet.

**DOCUMENTS-Route:**

* **GET /documents/all/:userId**
* Hämtar alla dokument för en specifik användare baserat på användar-ID.
* Endast o-raderade dokument hämtas (isDeleted = 0).
* Returnerar en JSON-array med dokumentdata.
* **POST /documents/add**
* Lägger till ett nytt dokument i databasen.
* Använder userId, title och documentBody från requesten.
* **POST /documents/getdoc/:documentId**
* Hämtar ett specifikt dokument baserat på dokument-ID och användar-ID.
* Returnerar dokumentdata som JSON-objekt.
* **PUT /documents/update/:documentId**
* Uppdaterar ett dokument baserat på dokument-ID och användar-ID.
* Använder titel, dokumentkropp och användar-ID från förfrågan.
* Uppdaterar tidsstämpel för senaste uppdatering.
* Returnerar ett framgångsmeddelande.
* **DELETE /documents/delete/:documentId**
* Soft-delete eller återställer ett dokument beroende på om isDeleted är 0 eller 1.
* Returnerar dokumentdata efter att ändringen skett.

**Övrigt:**

Felhantering implementerad för de olika end-pointsen.

Använder bcrypt för lösenordskryptering.

Använder UUID för att generera unika användar-ID.

Använder isDeleted-flaggan för soft-delete av dokument.

Använder CURRENT\_TIMESTAMP för att uppdatera tid/datum vid ändring/uppdatering av dokument.