Dokumentasjon Verksted Oppgave

06.09.2023

Gøran Knutsen

# Om dokumentasjon

Dette er et dokumentasjonshefte som skal forklare arbeidet og koden til prosjektet. Jeg skal også ha noen skisser og tidsplan i dokumentet for å vise hva som tok tid og hvordan programmet er satt opp. Så skal jeg også drøfte hva som ble gjort hver dag av jeg jobbet på kodingen.

# Moduler

## Next-auth

Next auth er en modul som er laget for håndtering av alt med bruker informasjon å gjøre. Man kan velge mellom mange «providers» sånn som google, github, twitter, steam osv. Jeg valgte da å bruke google i min. Next auth er også en ganske så simpel løsning hvis du skal gjøre basic user håndtering som gjorde det til et bra valg for akkurat dette prosjektet.

## Axios

Jeg valgte å bruke axios for mine api calls for dette prosjektet. Dette er fordi det er en veldig utbredt modul som blir brukt av mange, så mann finner all informasjon man trenger om det hvis mann vil. Jeg valgte Axios også fordi det er tryggere en flere av de andre konkurrentene, sånn som fetch. Axios har også automatisk håndtering av JSON innebygd som gjør arbeidet lettere.

## Et bilde som inneholder tekst, skjermbilde, programvare, Multimedieprogramvare Automatisk generert beskrivelseRouting

For routing så valgte jeg å bruke den innebygde app routingen til Nextjs. Denne routingen kom ut ikke alt for lenge før jeg begynte på dette prosjektet. Siden den er innebygd så slepper mann å skaffe en annen modul til å gjøre jobben. App router er også ganske oversiktlig på hvordan filene er satt opp med koden i. Som du ser på siden, så blir det satt opp i en src/app folder, her lager mann da mapper for hver side mann skall ha i stedet for en fil. For eksempel så ser du jeg har en folder som heter login og en fil som heter page.tsx under. Dette er annerledes en normal som er at filen bare heter login.tsx. Dette gjør det også da mye lettere å ha kontroll på filer, siden mann kan lage type og schema filer i samme mappe som den som har page.

## Zod

Zod er en modul som er lagd for schemavalidering i typescript. Denne er for eksempel brukt i min kode på alle forms for å sjekke om at dataen folk sender inn er rett og ellers sender en error til brukeren og forteller hva som skal gjøres annerledes.

## TailwindCSS

Jeg har valgt å bruke et rammeverk for min css fordi det hjelper gjøre css både lettere å jobbe med, og kjappere å jobbe med. Tailwindcss er normalt med i nextJS, så når man starter et nextjs prosjekt så blir mann spurt om mann vil ha det, og den står automatisk på ja i starten.

## Prisma

Så har jeg valg å bruke prisma for å sette opp databasen min med såkalte «models» som man lager. Her definerer mann hva som skal være i databasen og vilken type verdi det er. Man kan også sette at de default blir en value og at noen felt er unique, noen er id og mann kan sette opp foreignkeys. Prisma gjør det også lett å sende/hente ting fra databasen, så det er også det jeg bruker for dette.

Et bilde som inneholder tekst, skjermbilde, programvare, Font

Automatisk generert beskrivelse

## React query

Jeg valgte også å bruke en modul som heter react query. Denne modulen brukte jeg for å håndtere fetching av data fra databasen fra client siden. Det denne gjør er at den henter data fra databasen imens den gir ut variabler som, isFetching, error, data og status. Disse kan du bruke for å implementere mere funksjoner i koden din lettere. De har også muligheten for å sette opp infinitequery, denne gjør at du har nesten innebygd pagination i modulen hvis du setter den opp på rett måte. For dette så har jeg en modul som heter react-intersection-observer. Denne modulen finner ut om jeg har scrollet til bunn av mine posts, så kjører den react query som henter nye posts.

Alle ting som blir hentet med react query kan bli gitt ren query key. Med denne keyen så kan mann gjøre mye med dataen fra andre deler av prosjektet også. Dette har jeg brukt for å hente ny data osv når jeg lager nye posts/sletter posts/ redigerer posts. Når dette blir gjort invalider jeg query key til dataen, og ny data blir hentet.

Under så ser du koden min jeg brukte for å hente verktøyene fra databasen. Den gråe teksten er kort forklaring av det som er i koden.

Et bilde som inneholder tekst, skjermbilde, programvare, Multimedieprogramvare

Automatisk generert beskrivelse

# Arbeidsprosess

## Tidsplan

# Tidsplan

Hvite teksten er den plan som ble lagd før fagprøve, den røde teksten er oppdatert fra når jeg jobbet med fagprøven. Den røde teksten sier da faktisk tiden som er brukt på hver del og når det ble gjort.

|  |  |
| --- | --- |
| Dag | Forklaring |
| Mandag 04.09.2023 | 0800-1300: 0800-0934  planlegging    1300-1500: 0934-1012  Så setter jeg opp oppgaven på github, setter opp databasen og gjør klar VSC  1500-1630: 1012-1031  Sette opp next auth sin backend |
| tirsdag 05. 09.2023 | Tirsdag 0800-1000: Mandag 1031-1055  Sette opp innloggings systemet med session  Tirsdag 1000-1330: Mandag 1055-1135  Sette opp hjemmeside og innloggings side  Tirsdag 1330-1530: Mandag 1135-1410  Starter med å lage siden der innleggene vises og lag der man lager nye innlegg. |
| onsdag 06. 09.2023 | Onsdag 0800 – 1000: Mandag 1410-1530  Gjør ferdig innlegg visning og side for oppretting av innlegg  Onsdag 1000-15:30: Tirsdag 0850-1023  Sette opp systemet for kommentarer på post og visning til disse |
| torsdag 07. 09.2023 | Torsdag 0800-14.00: Tirsdag 1055-1352  Sette opp administrasjons siden  Torsdag 1400-1600: tirsdag 1352-1800 og onsdag 0900-1100  Så se over alt jeg har gjort, forbedre der jeg kan og gjør koden oversiktlig |
| fredag 08. 09.2023 | Jobbing på kode litt her og der  Jobbing med dokumentasjon |
| lørdag 09. 09.2023 | Jobbing med dokumentasjon og kvalitetssikring |
| søndag 10. 09.2023 | Jobbing med dokumentasjon og kvalitetssikring |
| Mandag 11. 09.2023 | Se over alt som er gjort |

## HMS under arbeidet

HMS har vært ganske ok under arbeidet av denne oppgaven. Jeg har en ergonomisk kontor stol som jeg har stilt inn til å passe meg bra, men utover dagen etter jeg jobbet i en stund så lå jeg nesten i stolen å programmerte som er dårlig for ryggen. Ellers så har det vært fin temperatur og luftkvalitet på kontoret. Jeg tok å jobbet litt for hardt den første dagen og tok nesten ingen pauser i det hele tatt så jeg var utslitt når jeg kom hjem. Så burde tatt mer pauser og bevegd meg litt, ble bedre på dette dagene etter.

## Daglig arbeid

### Mandag 04.09

Mandagen så startet jeg med å få oppgaven, så skrev jeg planleggingsdelen og leverte det inn. Denne planen skrev jeg om hvilken ting jeg skulle bruke for prosjektet og hjelpemidlene jeg har tengt til å bruke.

Jeg gikk for dette alternativet:

* Framework: NextJS (TS)
* Server: postgres
* Utstyr: Postgres server
* Program: Visual studio code, docker
* Dokumentasjon og lagringsplass: Github
* Autentisering: nextauth

Denne planen ble godkjent og jeg begynte med arbeid på prosjektet.

Planen for dagen var å jobbe på å sette opp prosjektet og next auth, men det gikk mye kjappere enn planlagt så fikk gjort vekk en del ting fra de neste dagene og.

Jeg startet ned å sette opp hele prosjektet og databasen. Databasen satt jeg opp med user tabell, posts tabell, comment tabell og image tabell.

Så satt jeg opp next auth filen med google provider og credentials privider. Dette var for at admin skulle bruke credentials og brukere skulle bruke google. Så wrappet jeg sessionprovider til nextauth rundt koden minn i layout dokumentet, jeg tok også å satt opp query client provider samtidig som dette fordi jeg viste jeg kom til å trenge det seinere. Så tok jeg også å lagde navbaren, dette var fordi det var her jeg skulle ha login knappen osv. Dette gikk veldig bra og fikk ingen problemer, og det funket bra. Så lagde jeg en custom login page med credentials og google.

Så begynte jeg på hjemmesiden som skulle vise alle verktøy som hadde blitt laget. Jeg begynte med å sette opp backend fordi det ikke var vits å designe hvordan data framvisningen var uten dataen der. Så jeg lagde post og get request for verktøy (forklart lengre ned i dokumentasjon), og jeg satt opp schema som brukes for validering av forms. Jeg satt også opp image og upload api’et som også er nevnt senere i dokumentasjon.

Så satt jeg opp en modal i em component som heter <NewPostModal/> her har jeg en modal som bare har formen for laging av verktoy. Denne blir åpnet av å trykke på en lag post knapp på siden. Her satt jeg opp hele formen med bruk av Useform. Useform brukte jeg fordi den er veldig bra for håndtering av forms. Jeg satt {…register(‘navn av felt’) på hver input, dette navnet i register måtte passe til det navnet i schemaen jeg hadde lagd tidligere. Useform tar på hver submit og sjekker om dataen som har blitt sendt unn passer til kravene jeg har satt i schemaen og hvis det ikke gjør det så kjøres ikke at en post blir lagd og error blir sendt til html koden som vises til brukeren. Dette forteller hva brukeren må putte inn i input felt for at de skal kunde sende inn.

Jeg brukte en component i formen som er upload button. Dette er noe jeg har lagd for en stund siden men som kom til å passe perfekt i koden minn. Den var for håndteringen av bilde opplasting. Alt denne gjorde var å ha en fin input for bilder, og når de ble opplastet så satt den filen til en variabel for å bli brukt senere. Denne verdien ble også brukt for å vise preview av bildet til brukeren sånn at de kan se om de har opplastet rett bilde.

Min onsubmit for å lage en ny post er delt i to, dette er fordi jeg slet litt med det å opplaste bilde og sette imageId, før jeg da sender imageId med all den andre dataen til api’et. Så først så lagrer den bildet og setter ImageId, så kjører den onsubmit2 som da sjekker dataen mot schema før den kjører, hvis alt er rett så kjøres createpost(data) som er et api call til post i /api/verktoy. Når denne var kjørt så satt jeg at query for verktøy ble invalidated så listen ble oppdatert.

Et bilde som inneholder tekst, skjermbilde, programvare

Automatisk generert beskrivelseJeg hadde ikke satt opp noe håndtering for om folk hadde rettigheter til å se create post knappen enda så alle kunne trykke på den men ikke lage posts. Etter dette så funket det å lage posts men hadde ingen form for visning enda.

Bildet du ser er koden jeg lagde for å hente posts fra databasen. Det er en infinitequery call som kjører når fetchnextpage kjøres. Når den kjøres så setter den at alt dataen tilhører keyen «verktoy» som kan calles når som helst for å validere, oppdatere og hente denne dataen. Så kjører den funkjson getPosts som er lagt i service folderen. Dette er bare en simpel axios request, men den er puttet en annen plass for å holde koden mer oversiktlig.

Så har jeg en array.map lengre ned i koden som ikke kjøres før dataen fra infinitequery er satt. Denne mappen er en map av en component jeg lagde som var <Toolcard /> her sender vi bare dataen for det verktøyet som blir mapped inn, så har den all html for carden. Dette er for å minske koden i hoved siden og holde den mer oversiktlig. Så nå funket visningen for posts og laging for posts helt fint. Så lagde jeg edit og slette knappene og puttet inn i cardet også.

Et bilde som inneholder tekst, hammer, verktøy

Automatisk generert beskrivelse

Hjelpemidler brukt:

Litt hjelp av daniel for å fikse regex for datepurchased i verktøy schema

Tidligere egen prosjekt: <https://github.com/gorknu19/formula1>

Resultat av dagen

* Planleggings del ferdig
* Databse, github og vsc satt opp
* Onnlogings system og next auth satt opp
* Hjemmeside og innloginsside satt opp
* Posts visning og lagin satt opp

Mye mer gjort ferdig en planlagt som gjør at jeg har mer tid til å fikse ting og forbedre kode. Pluss mer tid for dokumentasjon.

### Tirsdag 05.09

Et bilde som inneholder tekst, skjermbilde, Font, programvare

Automatisk generert beskrivelseEt bilde som inneholder tekst, skjermbilde, verktøy, design

Automatisk generert beskrivelsepå tirsdagen begynte jeg å jobbe på kommentarer og håndteringer av kommentarer, mye av koden her er ganske lik som verktøy laging/henting, så gjenbrukte og bare endra koden på noen ting for kommentarer. Startet med å lage post og get api for comments som blir forklart lengere ned i dokumentasjon. Så måtte jeg lage felt for å lage comments og for å vise comments. Dette valgte jeg å putte i en <comments/> component, men fant ut at det ble mye rot hvis det står under hver verktøy som vises. Så etter å ha lagd filene for comments så tok jeg å lagde en modal component som heter verktoy.modal. Dette var bare en modal som viste all informasjon om verktøyet så den har nesten bare html kode for visning av verktøy data. Etter denne var laget kunne jeg jobbe på <comments/> componenten igjen. Her satt jeg opp en simpel useform som bare hadde input av tekst, og den hentet inn postId fra verktøyet sinn data. Dette ble gjort med defaultvalues som satt at postId skulle settes til verktøyet sinn id. Input feltet til comments satt jeg opp så det kommer rett under informasjon til verktøyet. Når denne ble sendt inn så ble createComment kjørt som bare er et post api call til /api/verktøy/comment. Etter dette satt jeg opp en use query som hentet alle kommentarer på nesten lik måte som verktøy bare uten pagination, så alle komentarer blir hentet på en gang. Dette ble da mappet under inputen til kommentarene så de ble vist. Etter dette var alt med laging og visning av posts ok kommentarer ferdig og jeg kunne begynne på admin tilganger.

Det første jeg gjorde for å begynne på admin tilganger var at jeg lagde delete api’et til comments. puttet inn en slett kommentar knapp på hver kommentar, denne kunne bare bli vist av folk som har whitelsited true på brukeren sinn i databasen. Denne var veldig lett å lage, alt den gjorde var at når mann trykker slett så kjøres deletecomment(comment.id) som er et delete api call til /api/verktoy/comments som bruker commentId til å finne og slette kommentaren, så imvalideres queryen til comments og listen blir oppdatert.

Etter dette begynte jeg på admin rettigheter til verktøy. Første jeg gjorde var å gjøre at bare de som er whitelsited kan se create post knappen, edit og delete. Etter dette var gjort satt jeg opp backend for sletting av posts som funket på samme måte som comments sinn bare at den kjørte deletePost(verktoy.id) i stedet.

Et bilde som inneholder skjermbilde, hammer, tekst, verktøy

Automatisk generert beskrivelseSå satt jeg opp en modal som heter editToolModal. Denne satt jeg opp på nesten akkurat samme måte som create post modal. Forskjellen er at jeg tar inn all verktøy data fra det verktøyet som skal redigeres, også setter jeg at defaultValues til alle inputs er den dataen, dette gjør at alle input felt starter med dataen som var til det verktøyet. Jeg tok også med inn postId som bare var verktøyet sinn id. Dette er fordi det trenges for å redigere den posten i backend. Med dette så fikk jeg et lite problem med visning av bilde i formmen, siden bildet er på databasen, og det ikke er sikkert de skal legge inn nytt bilde. Så jeg gikk inn i UploadButton componenten og puttet inn at hvis imageId er satt allerede og ingen ny bilde er opplastet, så viser den bilde som kommer igjennom imageId hvis ingen ny bilde er opplastet. Dette funker bra siden når folk lager posts, så finnes ikke Image id, og når folk redigerer og opplaster nytt bilde, så endres det til at bare den nye som er opplastet vises i stedet.

Etter dette var frontend for post editing ferdig, så lagde jeg backend for editing, som er nevnt seinere i dokumentasjon. Og editing av posts var ferdig. Når editing av posts var ferdig så hadde jeg satt opp hele admin delen av siden og rettighetsstyring.

Når jeg var ferdig med administrasjon siden, så fant jeg ut at jeg hadde et lite problem med koden. Det ble mye styr med visning av kommentarer og det å lage kommentarer for brukere som ikke var admin. Jeg hadde programert mens jeg var i admin brukeren heletiden, og når jeg byttet til google konto, så kunne jeg ikke se eller lage noen comments, side userId ikke fantes for de og det var det jeg brukte for å sjekke inloggingen til folk. På grunn av dette så måtte jeg bygge om autentiserings systemet mitt, så jeg fjernet credentials provider og heller satt opp at alle logger igjennom google. Problemet nå var at jeg har ingen måte å sjekke om det bare er en bruker eller admin bruker. På grunn av dette så tok jeg å satt opp en prisma adapter, denne prisma adapteren tar å lagrer konto data fra de som logger inn med google på min database, med dette kunne jeg da lage et eget felt som heter whitelisted for admin håndtering, men for å få dette til å funk måtte jeg bygge om databasen til å lagre dataen som prisma adapter jobbet med. Heldigvis så var det dokumentasjon på dette på deres side, og alt jeg trengte å gjøre var å kopiere over deres user tabeller fra siden deres. Jeg gjorde dette og puttet inn min whitelisted variabel i user tabellen. Etter dette var gjort så funket koden som den skulle, og de som var admin så det admin skulle, brukere så det de skulle, og ikke logget inn folk kunne bare se verktøyene.

Hjelpemidler:

* For autentiserings fiks: <https://next-auth.js.org/v3/adapters/prisma>
* Tidligere egen prosjekt: <https://github.com/gorknu19/formula1>

Resultat av dagen:

* Kommentar systemet satt opp
* Hele administrasjons og rettighetsystem satt opp
* Fikset problemer med administrasjon håndtering og måtte bygg og autorisjasjon system som var styr

### Onsdag 06.09

Startet dagen med å gå igjennom hele koden minn og kommentere alt, dette er for at da er koden godt dokumentert, fordi de beste programmene trenger mann ikke ekstra dokumentasjon for å kunne jobb med. Så jobbet jeg med dokumentasjon.

Resultat av dagen:

* Jobbet med dokumentasjon
* Jobbet med å kommentere koden minn
* Brukt eraser.io til å lage skisse av databasen

### Torsdag 07.09

Jobbet med dokumentasjon

Resultat av dagen

* Mye dokumentasjon unngjort
* Nesten bare egen vurdering igjen og små jobb på dokumentasjon

### Fredag 08.09

Startet dagen med å se over koden om det var noen små ting som trengtes fiksing. Fant ut jeg hadde en del engelsk blanda med norsk i HTML koden som vistes, så dette fikset jeg sånn at alt var norsk. Så viste jeg prosjektet til en kollega som da nevnte sjekk for bilde opplasting for om den er for stor eller rett type. Så jeg fikset håndtering for sjekk om at det er en image som er lastet opp og at filen ikke er for stor. Dette gjorde jeg ved å først hente typen av filen som er opplastet, så sjekke om den starter med image, gjør den ikke det så blir det error. Så brukte jeg bytes variabelen som var der for å sjekke om det var mer enn 10mb i bytes, og om det er det så sender den error.

Et bilde som inneholder tekst, skjermbilde, programvare, Multimedieprogramvare

Automatisk generert beskrivelse

Så tok jeg også å jobbet litt med zod, dette er fordi brukeren kunne sende inn så lange strings de ville i mange felt, så jeg puttet inn en maks grense for hver tekstfelt sånn at mann ikke kunne sende inn så mye.

Så jobbet jeg med ferdig stilling av dokumentasjon også ferdigstilling egenvurderingen.

### Lørdag 09.09

Delvis syk så jobbet ikke med det

### Søndag 10.09

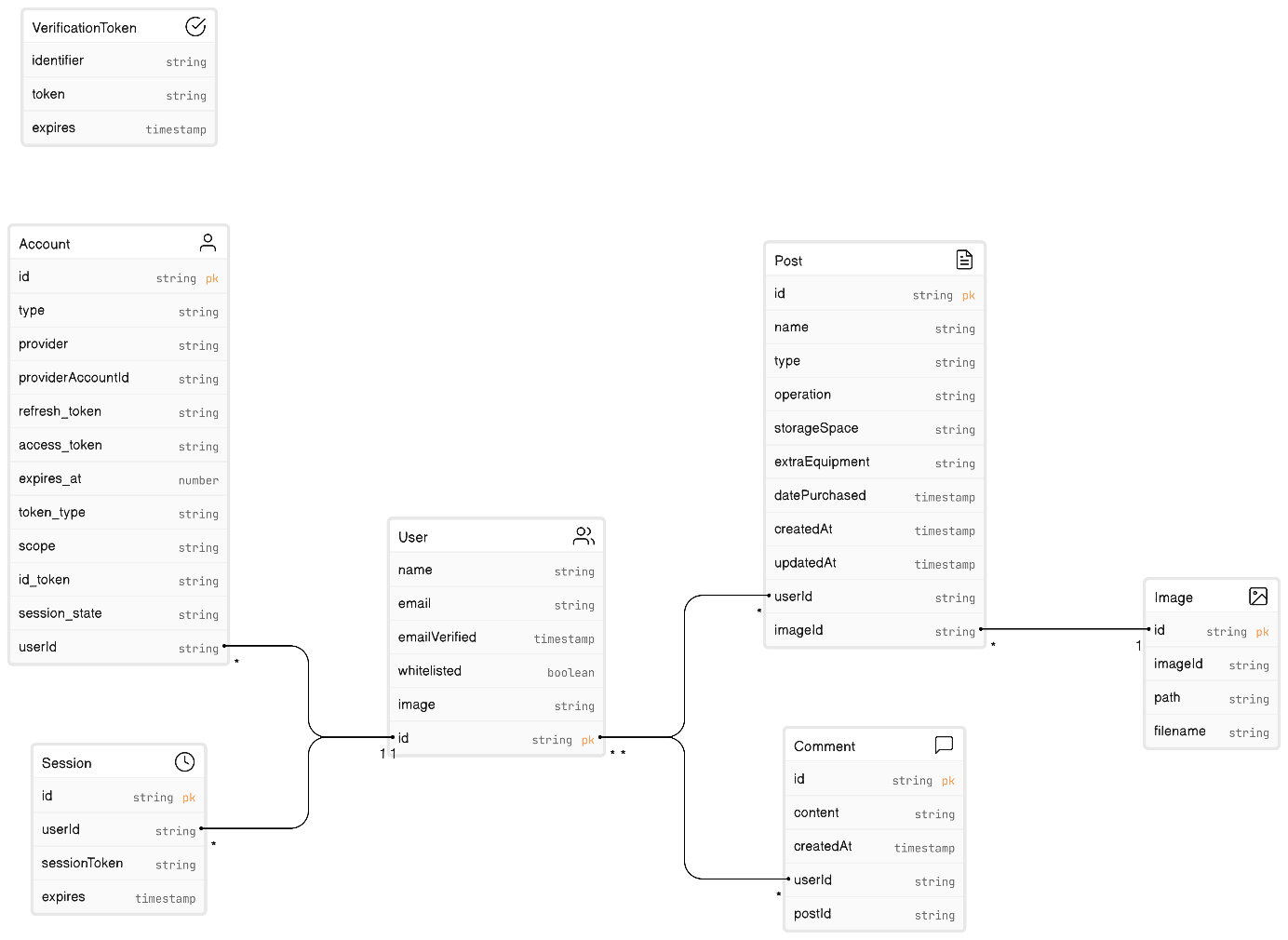
Fortsatt litt syk så jobbet ikke med det

### Mandag 11.09

Så over dokumentasjon og kode.

# Systemskisser, database og API

## Database

Bildet under er en skisse av hvordan min database er satt opp igjennom prisma. Her kan du se hver tabell og hvilken verdier som er i hver tabell, og hvilken type verdi hver av de er. Du kan også se hvilken felt av hver tabell er linket til felt av andre tabeller. Et eksempel på dette er på User tabellen på id så ser mann at den er linked med foreignkey til session sinn userId, Account sinn userID, Post sinn userId og comment sinn user ID. Dette er gjort sånn at for eksempel hvis en bruker lager en post, så blir den brukeren sinn id lagret i posten sånn at når mann tar ut posten, så kan mann ta med bruker informasjon til brukeren som laget den.

Verification, Account og session tabellene er laget av og for prisma adapteren til nextauth. Denne adapteren håndterer alt med brukere på serveren på databasen for deg. Den har også laget deler av User tabellen, men jeg har lagt in min egen variabel som heter whitelisted, denne bestemmer om folk er admin eller ikke. Whitelisted kan bare endres på databasen og ingen andre plasserm så er der mann setter om folk er admin. Alle disse tabellene blir automatisk fylt av Prisma adapter og nextauth, så er ikke noe spesielt jeg gjør med de. Når folk logger inn for første gang med en bruker, så tar nextauth å sender all dataen fra brukeren til account og user tabellene. Verification og session tabellene er ting som blir brukt når folk logger seg inn og bruker nettsiden.

Post og Image tabellen er de som holder informasjon om alle verktøyene som er lagt inn i systemet. Her har du all nødvendig informasjon om produktet, og i post tabellen har du en ImageId variabel som er linket til image tabellen sinn id, dette gjør det mulig å ta ut bilde informasjon samtidig som mann tar ut en post fra databasen. Post tabellen har også userId lagret på seg som er linket til user tabellen sitt id felt. Dette er gjort for at mann kan ta med all bruker informasjon samtidig som mann tar ut post informasjon. Det at mann kan ta ut så mye informasjon lett igjennom prisma også med en simpel include i callet, gjør at man kan ta ut mye informasjon på en gang som sparer tid, og det hjelper holde koden oversiktlig. Det er brukt i min kode for å inkludere mer data i mine api calls.

Comment tabellen er den som holder all informasjon om kommentarer til verktøy. Denne kommentar delen av tabellen er satt opp med en foreignkey til user siden userId blir lagret i tabellen. PostId blir også lagret. Har gjort at man ikke kan include comments i api call til posts , dette er fordi for å gjøre koden mer effektiv, så blir ikke kommentarer hentet før du åpner en av verktøyene. Dette er gjort sånn at den ikke henter kommentarer til alle verktøy med en gang sammen med verktøyene, noe som gjør dataen mer oversiktlig også. Når mann åpner et av verktøyene blir da et api call gjort med postId til comment databasen som henter ut alle comments som matcher postId’en som er sendt inn. Også displayer det under verktøyet.

Et bilde som inneholder tekst, skjermbilde, programvare, Font

Automatisk generert beskrivelse

## Api

Api siden består av fire forskjellige mapper. Den første er /auth/[…netxauth], denne er for autentiseringssystemet mitt. Så har jeg /image/[id], det er her henting av bilder blir gjort. Så har jeg /upload som er håndteringen av bilder som blir opplastet. Den siste er /verktoy, denne er for all henting/posting/sletting/patching av dataen til verktøyene.

### /Auth/[…nextauth]

I nextauth delen av api’et mitt så har jeg 2 forsjkellige filer, den ene er auth-options og den andre er route. Auth-options filen er der all autentiserings kode er lagret. Her har mann alle callbacks som blir brukt for å hente session data, adapteren som er satt opp med bruk av innebygde prisma adapteren til nextauth, og providers. I dette prosjektet er det bare satt opp 1 provider, og dette er google sinn. For å få google provider til å funke så trenger mann 2 variabler, disse variablene er Client\_id og client\_secret. Disse kommer fra google sinn developer side: [**https://console.developers.google.com/apis/credentials**](https://console.developers.google.com/apis/credentials). Her setter mann opp prosjektet og får ut client\_id og client\_secret for å legge inn i koden.

Et bilde som inneholder tekst, skjermbilde, programvare, Font

Automatisk generert beskrivelseDet er også gjort endringer i callbacks på api siden for å sleppe igjennom to felt med mer data. Når et call for å hente session data blir gjort, så henter den ut mer bruker data fra serveren. Denne dataen blir da lagret i userData og lagt inn i updatedsession objektet. Denne blir da returnert til brukeren. Hver gang session data blir spurt for nå, så blir den nye dataen sendt ut i stedet for den gamle som var uten id og whitelisted variablene.

### /image/[id]

Et bilde som inneholder tekst, skjermbilde, programvare, Multimedieprogramvare

Automatisk generert beskrivelse/image/[id] er som nevnt tidligere api’et for å hente ut bilde informasjon og selveste bilde for å vise fram på siden. Dette er gjort på en måte der mann gjør et api GET call med iden til bilde direkte i lenken. På grunn av dette så kan jeg sette source for bilder i koden til bare en link som for eksempel «src=`/api/image/’id til bildet’`». Når callet blir gjort, så henter den ut iden fra urlen og lagerr det i en variabel, så henter den ut dataen om bildet, etter denne dataen er tatt ut så bruker api’et pathen til bildet som er lagret på databasen til å lese bildefilen som er lagret. Etter dette sender den tilbake bildet som blir lest til brukeren.

Dette api’et har også en sjekk som kjøres før den prøver å lese filen fra pathen. Denne sjekker bare om den klarte å få en path for bildet ut av databasen, så hvis den ikke gjør det så stoppes koden og en response blir sendt som sier at bildet ikke finnes. Dette stopper koden for å prøve å lese en fil som ikke eksisterer som kommer til å skape errors.

Et bilde som inneholder tekst, skjermbilde, programvare

Automatisk generert beskrivelse

### /Upload

Upload er apiet for håndtering av bilder som blir lastet opp til siden. Når et api POST call blir gjort til denne siden med en fil, så henter den først ut data fra filen. Så tar den å henter bytes ut av filen, og da lager en buffer ut av den. Så setter den path og id til bildet før den sender inn dataen om bildet til databasen. Etter det så blir da filen lagret der pathen peiker med bruk av bufferen som ble lagd tidligere. Etter bildet har blitt lagret så sender apiet tilbake informasjon som ble lagret på databasen til brukeren.

Dette apiet har også en sjekk i seg som sjekkes før all dataen blir lagret. Dette er en liten if statement som bare sjekker om det i det heletatt er opplastet en fil, og hvis det ikke er det så sender den tilbake en error til brukeren. Så henter den filtypen til filen som er lastet opp og sjekker om det er et bilde i det heletatt, er det ikke et bilde blir det returnert en error. Siste sjekken som blir brukt er at den henter ut hvor mange bytes det er i filen, og sjekker om det er mer enn 10mb, eller 10000000 bytes. Hvis det er over dette er filen for stor og det blir returnert en error.

### Verktøy

Verktøy APIet er delt opp i to forskjellige deler. Route og schema som er lagret direkte under verktøy folderen er de som er meint for alt med bare verktøyet å gjøre. Inni verktøy mappen så er det en annen mappe som heter comments som har route og schema i seg også. Comments mappen har alle api call som har med kommentarer til verktøy å gjøre.

### Verktøy API

Verktøy apiet har 4 forskjellige typer av api calls i seg, det er en get request, patch request, delete request og post request. Post requeste er såklart for å lage nye verktøy på databasen. Delete er for å fjerne verktøy. Patch er der for når mann redigerer et verktøy. Get er der for når mann skal hente ut alle verktøyene.

#### Et bilde som inneholder tekst, skjermbilde, programvare, Operativsystem Automatisk generert beskrivelsepost

Post requeste starter med å ta inn dataen som har blitt sendt inn og validere det mot schemaet som er laget ned zod for validering av dataen som blir sendt inn. Etter dette så tar den å henter ut session informasjon fra serveren. Så sender den bare all dataen inn til databasen og returnerer responsen som databasen gir.

Så klart så har post callet også noen sikkerhetshåndteringer. Den første er bare sjekken mot schemaet som er laget som sier hva dataen inni skal være og kravene til de. Etter dette så bruker den session dataen som er hentet for å sjekke om brukeren som lager posten har whitelisted rettigheter til å kunne lage posts. Så tar den å sjekker om valideringen igjennom schema var successful eller ikke. Etter dette så er det en sjekk for om userId kommer igjennom fra session, dette er fordi den trenges for å lage et verktøy på database, så vi må vite om den har kommet igjennom.

#### Et bilde som inneholder tekst, skjermbilde, programvare, Multimedieprogramvare Automatisk generert beskrivelseGet

Get requesten starter med å hente ut parameterene som blir sendt av react-query for pagination. Disse verdiene er pageSize som sier hvor mange som skal bli tatt ut for hver side, skip forteller hvor mange som skal skippes av de i databasen. Alle posts som ikke blir tatt med ut av skip er posts som allerede har blitt tatt ut og vist på siden. Etter denne dataen er blitt satt så henter apiet ut hvor mange posts så finnes, også henter den ut de postene som skal hentes ut på det kallet. Altså den bruker skip variabelen for å skipe de som er hentet ifølge react-query også tar den ut så mange som vi har sagt skal bli tatt ut hver gang. Etter Dette settes nextcursor som er verdien av skip og pagesize sammen, denne blir laget fordi den forteller da hvor mange posts vi har tatt ut og på hvilken post nye må bli tatt ut. Hvis alle posts har blitt tatt ut så blir nextCursor undefined, dette er fordi react-query da ikke prøver å hente ut mere posts.

#### Delete

Et bilde som inneholder tekst, skjermbilde, programvare

Automatisk generert beskrivelseDelete apiet starter med å hente ut dataen som har blitt sendt til callet. På delete så er det bare en verdi som blir sendt og det er postId som er iden til posten som skal slettes. Så tar den å henter ut session dataen også.Etter dette så sletter den da alle kommentarer som har postId på seg som passer til den posten som skal slettes, Så tar den å sletter posten. Dette blir gjort på denne måten fordi hvis man sletter posten først, så får mann foreignkey error fordi alle de kommentarene ikke er «linked» til en post lengere.

Det er såklart litt sikkerhets sjekk bygd inn i delete apiet. Etter at session dataen er hentet fra serveren, så blir det kjørt en sjekk for om den brukeren som prøvde å slette har whitelist rettigheter til å kunne slette. Hvis brukeren ikke har det stoppes koden og det blir sendt en 401 error som sier at brukeren ikke har rettigheter til å gjøre det. Det blir også såklart gjort en sjekk for om postId som skal bli brukt for slettingen i det hele tatt er satt, ellers så stoppes koden og en error med kode 400 blir sendt så sier at ingen postid ble oppgitt.

#### Patch

Et bilde som inneholder tekst, skjermbilde, programvare, Operativsystem

Automatisk generert beskrivelseDet siste kallet for verktøy er patch, denne er bygd opp nesten helt lik som post callet. Den starter med å hente inn dataen som har blitt sendt, dette blir da sjekket mot en validator fra zod som sjekker om dataen passer til det den skal være. Eneste forskjellen her fra post requestet er at mann har postId med i dataen som kommer inn. Så tar den å sender inn dataen til databasen, her bruker den postId til å finne ut hvilken post som skal endres, så oppdaterer den dataen til den med den nye dataen som ble sendt inn. Etter det returnerer den responsen fra databasen.

Patch callet har også noen sikkerhetshåndteringer. Den første er bare sjekken mot schemaet som er laget som sier hva dataen inni skal være og kravene til de. Etter dette så bruker den session dataen som er hentet for å sjekke om brukeren som lager posten har whitelisted rettigheter til å kunne lage posts. Så tar den å sjekker om valideringen igjennom schema var successful eller ikke. Etter dette så er det en sjekk for om userId kommer igjennom fra session.

### Comments API

Comments apiet er for alle kommentarer som er laget for verktøy. Den består av 3 forskjellige type api calls i stedet for verktøy sine 4. Comments har Post for laging av nye kommentarer, get for henting av kommentarer, og delete for sletting av kommentarer.

#### Post

Et bilde som inneholder tekst, skjermbilde, programvare, Operativsystem

Automatisk generert beskrivelsePost requeste starter med å ta inn dataen som har blitt sendt inn og validere det mot schemaet som er laget ned zod for validering av dataen som blir sendt inn. Etter dette så tar den å henter ut session informasjon fra serveren. Så sender den bare all dataen inn til databasen og returnerer responsen som databasen gir.

Det er en sikkerhetssjekk for å sjekke om det er en bruker som lager en kommentar. Dette gjør den ved å bruke session dataen som har blitt hentet inn for å sjekke om user.id er satt. Dette er fordi hvis den ikke er satt så er brukeren som lager kommentaren ikke logget inn og da ikke har rettighetene til å gjøre dette.

Et bilde som inneholder tekst, skjermbilde, programvare, display

Automatisk generert beskrivelse

#### Get

Get requesten starter med å ta ut dataen som ble sendt til den. På get requeste så blir det sendt inn en postId verdi som seinere brukes ved henting av data fra databasen. Etter postId har blitt hentet ut av params og satt til en variabel så tar callet å henter ut alle kommentarer fra databasen som har samme postId lagret som den som ble sendt til apiet. Etter dette returnerer den kommentarene som er til den posten.

Det er bare en liten sjekk i dette api callet, og det er postId. PostId blir sjekket om den er satt, og hvis den ikke er satt så stoppes koden og får vi en response som sier at den ikke er satt.

#### Delete

Et bilde som inneholder tekst, skjermbilde, programvare, Operativsystem

Automatisk generert beskrivelseDelete requesten starter med å hente ut dataen sendt til callet, som i dette tilfellet er bare commentId. Etter denne har blitt satt så henter den ut session dataen, før den da sletter kommentaren som hadde samme ocmmentId som den som ble sendt inn. Så returnerer callet den sletta kommentaren.

Det er såklart litt sikkerhets sjekk bygd inn i delete apiet. Etter at session dataen er hentet fra serveren, så blir det kjørt en sjekk for om den brukeren som prøvde å slette har whitelist rettigheter til å kunne slette. Hvis brukeren ikke har det stoppes koden og det blir sendt en 401 error som sier at brukeren ikke har rettigheter til å gjøre det. Det blir også såklart gjort en sjekk for om commentId som skal bli brukt for slettingen i det hele tatt er satt, ellers så stoppes koden og en error med kode 400 blir sendt så sier at ingen commentId ble oppgitt.