Webapplikasjoner

ITPE3200-1

Oppgave 1 - Aksjehandel

Mohammad Abo Khalifa	S354364
Sajaval Ahmad	s354368
Hassan Mehmod Hussain	s354545
Naveen Vijayasanker	s354362

1 av 14

INNHOLDSFORTEGNELSE

2 av 14

3 av 14

Hvordan vi kom frem til oppgaven om aksjehandel

Vi fikk i oppgave å implementere en applikasjon for en av tre mulige oppgaver:

- Aksjehandel side
- Offentlig UFO observasjons database
- Symptom til diagnose kalkulator

Vi gikk sammen en gruppe på fire studenter og etter en rekke mengde diskusjoner kom vi frem til en enighet om å velge første oppgaven, aksjehandel side. Vi valgte aksje på grunn av at vi har litt kjennskap til aksje sider som vi tidligere har sett og brukt og da ble det ganske lett og komme frem til en enighet at dette er det de fleste av oss har best kjennskap med ut ifra oppgavene vi fikk fordelt. Videre fordelte vi arbeidsoppgaver om hvem som skulle ta seg av brukergrensesnitt, programmering og modellering av nettsiden.

Planlegging og diagrammer

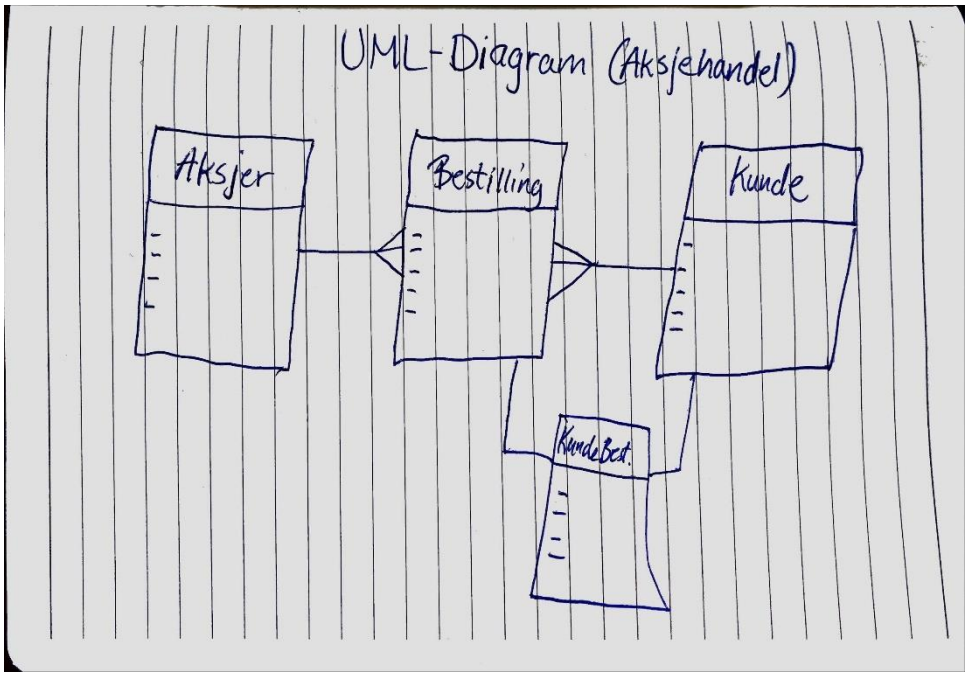
Modellering av UML-diagram skjedde over ulike faser av arbeidet vårt. Vi skisserte de ulike modellen og eliminerte delene som vi følte ble litt unødvendige. Startsfasen besto av flere gruppemøter hvor vi gikk over hva vi burde inkludere i databasen vår, og hva vi kunne vente med eller fjerne. Skisseringen startet av på papir, og ble skapt av ulike ideer over hvordan vi tenkte for oss sluttresultatet kunne se ut.

Etter at vi hadde valgt aksjehandelsoppgaven, møtte gruppen opp og diskuterte om vi skulle modellere diagrammene før eller eventuelt etter programmeringsbiten. Vi kunne enten velge å ta i bruk “Forward Design” eller Backwards Design”. Gruppen følte det ville vært mer nyttig å bruke “Forward Design”, ettersom det resulterer i økt forståelse av hva man ønsker, og hvordan sluttproduktet kan se ut. Vi mente også at det ville hjelpe med å skjønne hvordan koblingene skulle se ut på nettsiden, og hvordan klassene fra UML-diagrammet blir implementert. Use-Case diagrammet var noe vi følte for å ha med slik at vi forstår hva som forventes av oss, for eksempel at kunden skal kunne kjøpe og selge aksjer (Bluescape.com : 2019).

Meningen med UML diagram er for å visualisere hvordan vi tenker ryggraden til prosjektet vårt skal se ut, så vi kom tegnet ned de verdiene vi tenkte var de mest essensielle, og i hvilken eventuell klasse de skulle holdes til. Under vil man kunne fått sett på den første “low-fidelity”

4 av 14

tegningen vi lagde for å få litt sikt på hva vi trengte.

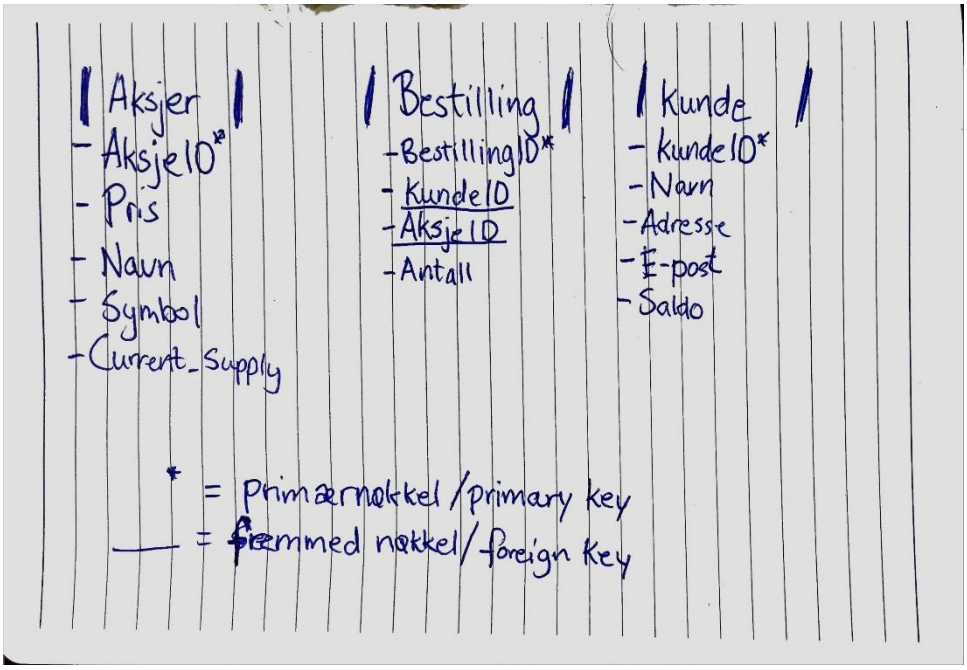


Som man ser over, hadde vi en skisse av penn på papir som viser de ulike tabellene vi tenkte kunne være en start på modelleringsfasen vår. Vi likte de tre øverste tabellene, og siden det ikke er en mange til mange kobling mellom noen av dem, følte vi at "kundebestilling" tabellen kunne bli fjernet, ettersom det ikke var nyttig å ha det.

Verdiene ventet vi med, ettersom vi ventet på å samle oss enda en gang for å gå gjennom hva vi kunne ha brukt som verdiene, og eventuelle primær og sekundær-nøkler. Etter god gjennomgang av de ulike ideene og å ha gransket ned til de mest viktige verdiene, kom vi frem til et resultat som kan sees på litt lenger nede i dokumentet.

Her er den første skisseringen av tabellen, som da ble produsert under gruppemøtene:

5 av 14



I skissen over hadde vi skrevet ned noen av verdiene som vi kom på etter en runde med "brainstorming", hvor alle kom med innspill over hva som kunne blitt med. Etter at den første verdi-skissen ble skapt (som kan ses over) valgte vi å legge til noen ekstra verdier og endre navnene til noen, slik at det vil være lettere å huske verdinavnene når vi koder dem.

* = Primærnøkkel

__ = Fremmednøkkel

AKSJER	BESTILLINGER	KUNDER
- AksjerId* - Symbol - Aksjenavn - Exchange - Pris - Image	- bId* - antall - AksjerId - KunderkId	- KunderkId* - kNavn - tlfNummer - rolle - balance

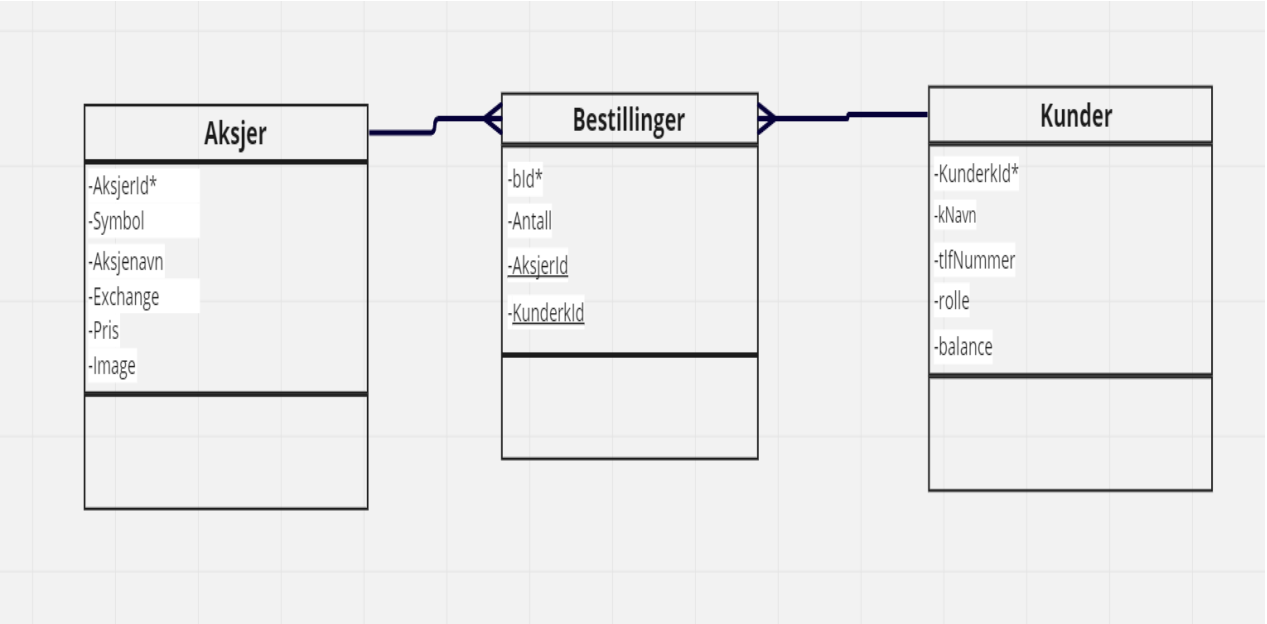
Radene over viser de verdiene vi har brukt i den offisielle databasen vår, dette er en forandring fra den første skissen, ettersom vi følte det ville vært mer kompakt og nøyaktig å legge til en verdi ved navn "Exchange" ettersom det ville forklare mer om aksjene til kundene hos oss. Adresse og E-post fra kunde-klassen har blitt fjernet, og erstattet med "tlfNummer" for å holde diagrammet kompakt. Dette er noe vi følte var essensielt, ettersom det ville da bli enklere å programmere senere.

6 av 14

Klassene aksjer og kunde har et en-til-mange forhold til Bestilling. Dette er noe vi kom frem til ved å diskutere over hva som ga mest mening for løsningen vi tenkte å lage, og det var disse argumentene som fikk oss bestemt på forholds-beslutningen:

- En kunde kan ha flere bestillinger/ kjøp, men en bestilling/kjøp kan ikke ha flere kunder.
- En aksje kan bli kjøpt i flere bestillinger, men en bestilling kan ikke ha flere aksjer.

Etter gode diskusjoner og møter, kom vi frem til en konklusjon over hvordan UML-diagrammet skulle se ut, og hvilke opplysninger det skulle gi. Gruppen ble også bestemt på å bruke verktøyet "Miro" ettersom det var lett å bruke, og ga oss gratis tilgang til verktøyene vi trengte. Dette er sluttresultatet på vårt UML-diagram:



Figur 1 – Use-Case diagram

7 av 14

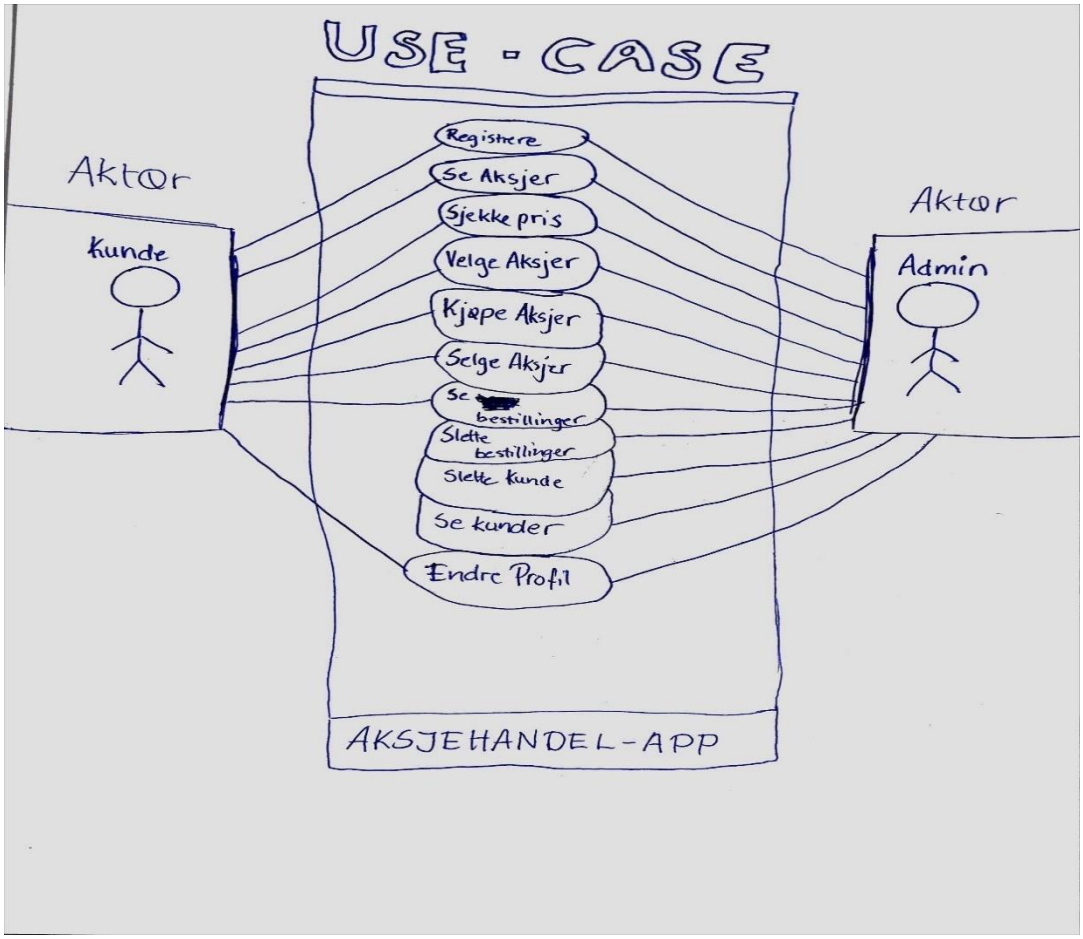
Use-Case diagram

Ettersom modelleringen av UML-diagram var noe vi følte ga oppgaven vår struktur, tenkte vi også å produsere et Use-Case Diagram som viser oss de ulike handlingene som vil bli gjort på applikasjonen. På denne måten, vil vi da kunne få et helhetlig bilde av hvordan nettsiden skal være, og hvilke funksjoner de ulike aktørene kan ta i bruk. Vi bestemte oss for to aktører, nemlig Kunde og Admin. De ulike handlingene som disse aktørene vil kunne ta i bruk er følgende:

Kunde	**Admin**
Registrere	Registrere
Se Aksjer	Se Aksjer
Sjekke Pris	Sjekke Pris
Velge Aksjer	Velge Aksjer
Kjøpe Aksjer	Kjøpe Aksjer
Selge Aksjer	Selge Aksjer
Se Kjøp/bestillinger	Se Kjøp/bestillinger
Endre profil	Slette Bestillinger
Se Kunder	Se Kunder
Endre profil	Endre profil
	Slette Kunde
	Slette aksjer for alle

Som det kan ses over, er det elleve handlinger valgt for Admin, mens Kunde forholder seg til 10 av de samme, men får ikke tilgang til ekstreme handlinger som sletting av bestillinger og kunder. Dette ble valgt utover prosessen av oppgaven, og er noe som hele gruppen syntes var aktuelt. Under, vil man kunne se et "low-fidelity" Use-case diagram som viser litt mer av det samme, bare med en bedre struktur.

8 av 14


USE - CASE
Aktør
Kunde
Aktør
Admin
Registrere
Se Aksjer
Sjekke pris
Velge Aksjer
Kjøpe Aksjer
Selge Aksjer
Se bestillinger
Slette bestillinger
Slette kunde
Se kunder
Endre Profil
AKSJEHANDEL-APP

9 av 14

Fremgangsmåten fra koding til produkt

Fra diagram til kode

Gjennom skoleårene har vi lært ulike måte å planlegge hvordan vi ønsker at en applikasjon skal fungere, og hvordan vi ønsker at brukere skal kommunisere med serveren. Use-Case diagram er en veldig god måte hvor vi kan definere flere aktører og hvordan vi ønsker at disse skal kommunisere med applikasjonen. For nettsiden vår, ønsket vi to aktører, en admin, og en kunde. Selv om det ikke er nødvendig å implementere login-system til oppgaven, var det viktig for oss å simulere dette så best som mulig, uten å trenge å logge inn, dette er også grunnen til at vi har basert vårt use-case på 2 aktører. Dette vil også hjelpe oss i neste oppgave, som gir oss en ide over hvordan siden skal se ut og fungere. Ut ifra use-case diagrammet var vi klare til å lage UML-diagrammet, som hele databasen vår har blitt basert på. Det som var viktigst var å vite hvilke data vi ønsket skulle bli satt inn i databasen, noe UML-diagrammet hjalp utrolig mye med, og det neste vi gjorde var å finne en API som kunne gi oss disse dataene.

Valg av API

API'en vi bruker heter TwelveData fra rapidapi API'en vi bruker gir oss all mulig informasjon om flere hundre aksjer. Alt fra informasjon om bedriften, til informasjon om aksjer som daterer tilbake et helt år. TwelveData tilbyr også på historiske og økonomiske data som er til stor hjelp når det kommer til å lage grafer, som er et verktøy brukere kan bruke til å studere markedet og se ulike trender. En slik graf kan vi se på siden hentEn, hvor en bestilling blir hentet ut. Disse dataene kan representeres i form av tekst på nettsiden, men dette er mindre beskrivende enn selve grafene og diagrammene vi har på nettsiden, men som likevel gir brukeren mye informasjon om hvordan aksjen gjør det på markedet. Dette var grunnen til at vi valgte denne API'en. Den tilbyr på et stort utvalg av informasjon, og økonomisk data som også passet med UML-diagrammet vårt og hvilken vei vi ønsket å ta denne oppgaven i. Oppretting av databasen ble også basert rundt denne API'en og informasjonen den tilbyr. Det er viktig å notere seg at koden for denne grafen som ligger i metoden createChart(chart) har blitt hentet fra nettsiden til apexchart.com. Dette er en side som tilbyr en chart bibliotek i javascript, det vi gjorde var å plotte inn dataene inn i koden, og grafen lages automatisk.

10 av 14

Første stegene mot en Database

Første steget vi tok, før vi begynte å lage databasen var å lage et layout til nettsiden, hvis vi hadde dette på plass, vil det bli enklere å forestille seg hvordan programmet skal fungere, og hvordan ulike data skal fordeles ut på nettsiden. Ut ifra dette, kunne vi også gjøre endringer på databasen etter behov, da slipper vi å lage databasen, også måtte gjøre endringer på den etter hvert. Dette var heller ikke et veldig stort problem, fordi oppgaven sier spesifikt hva brukeren kan gjøre på nettsiden, og use-case diagrammet ble basert på kravene. Selv om dette ikke var et krav, ønsket vi å gjøre applikasjonen mer brukervennlig for all type displays. Dette gjør nettsiden mer brukervennlig, slik at alle, uansett plattform kan bruke den, selv om dette var viktig for oss, var det likevel det siste vi gjorde, slik at vi kunne gjøre endringer basert på hvordan dataene ville se ut på siden, noe som er mye enklere å gjøre når det er ordentlig data på nettsiden.

Oppretting av klasse Aksje var det første vi gjorde da vi var klare til å starte på programmeringen. Her har vi alle kolonnene vi kommer til å ha i tabellene på ett sted, med get og set metoder. Inne på AksjeContekst deler vi opp disse kolonnene i ulike tabeller. De tabellene disse kolonnene ble delt inn i er Kunder, Aksjer og Bestillinger, hvor bestillinger er den tabellene som kobler sammen. Hver gang en kunde gjør et kjøp, vil dette bli registrert inn i bestilling tabellen. Siden dette er tabellen som limer databasen sammen, må vi også legge to virtual metoder, en for kunder og en for aksjer. Virtual metodene vil hente primær nøkkelen i både kunder og aksjer, og bruke disse som fremmednøkler i Bestillinger tabellen. På den måten, hver gang vi kaller på bestillinger tabellen, vil vi også hente ut brukere og kunder. Disse to klassene satt grunnlaget for hvilke metoder vi skulle ha med i controlleren.

Det er Aksjecontroller filen som inneholder alle metodene som gjør at vi alle funksjonene vi vil skal skje, kan skje. Det ble først laget en metode som lagrer alle aksjene i databasen. Aksjene skal bare bli vist på nettsiden så derfor blir skrevet inn i databasen og hentet med metoden hentAlle. Når brukeren velger en aksje for å få mer informasjon, og for å kanskje kjøpe den så blir metoden hentAksje tilkalt ved bruk av aksjeid, og informasjon om den spesifike aksjen vil bli hentet ut fra APIen. Hvis kunden velger å kjøpe en aksje så vil det bli lagret en ny rad i databasen med verdiene som blir valgt. Videre når man trykker på “se kunder” i nettsiden vil alle kundene som ligger i databasen vises ved hjelp av hentKunder metoden. Hvis man skal se på sine egne bestillinger så viser den alle bestillingene for den kunden ved å gi metoden hentBestilling kundens id. Velger kunden å selge en av sine kjøpte aksjer, så vil slettBestilling metoden bli tatt i bruk, og denne metoden fjerner bestillingen fra

11 av 14

databasen. Eller hvis det bare er deler av aksjen som skal bli solgt så vil det endre på gitte rad i databasen. Helt nederst til venstre så har du mulighet til å registrere en kunde, som vil bli lagret og hentet til «se kunder» siden. Under valg av profil så kan man trykke på profil og da har man mulighet til å endre på brukeren som har blitt laget. Sletting av kunder kan bare gjøres av admin brukere, og her blir metoden slettKunde brukt.

Dersom vi ønsker å hente ut data fra databasen, bruker vi jQuery $. get kommandoen, og sender gjennom en url som tilkaller klassen Aksje, og selve metoden, dette vil se slik ut

```
$.get("Aksje/HentAlle", async function(data) { //Informasjon går inn her});
```

Hvor HentAlle er selve metoden vi kaller på, som returnerer enten et objekt, en liste av kunder eller bestillinger. For å sende verdier inn til databasen, bruker vi $.post, hvor vi sender gjennom en id, et objekt av klassen kunde, eller det vi prøver å kommunisere til serveren, dette ser slik ut:

```
$.post("Aksje/SlettBestilling", selg, function(verify) { //Informasjon går
inn her});
```

SlettBestilling er igjen en metode i AksjeController, som sletter bestillinger/kjøp basert på hvilken informasjon som blir sendt gjennom variablen selg. Selg er et objekt som blir sendt gjennom, som inneholder informasjonen metoden vil trenge for å kunne slette bestillingen/kjøpet. Samme prinsipp går for endring av kunder og sletting av kunder.

Fra Javascript til Sluttprodukt

I denne oppgaven ønsket vi å utfordre oss selv, ved å klare å fremstille en aksjeside som faktisk oppdaterte informasjonen basert på brukeren sin input. Dette kan vi se gjennom hele nettsiden. Siden det ikke var nødvendig å lage login system, ville vi likevel prøve å fremstille dette ved at den som bruker nettsiden må lage en «profil» eller som vi kaller det, registrere en kunde. Ellers vil kunden ikke kunne kjøpe og selge aksjer, eller aksessere admin panel nettsiden, men likevel vil brukeren kunne navigere gjennom nettsiden, og se på de ulike aksjene. Etter at man har registrert en bruker, vil denne automatisk bli lagt til i en nedtrekks liste hvor man kan bla gjennom alle de registrerte kundene, og når man har valgt en bruker fra nedtrekks listen, vil kunden automatisk kjøpe og selge aksjer. Under registrering kan kunden også velge hvor mye saldo han skal starte med, og hvilken rolle man skal ha. Disse rollene er viktig når det kommer til å kunne aksessere admin panelet. Hvis rollen til kunden som er valgt i nedtrekks listen ikke er admin, vil denne navigeringsmuligheten ikke være i

12 av 14

menyen. Dersom man bytter til en bruker i nedtrekks listen som har admin rettigheter, vil denne bli synlig. Om man prøver å komme seg inn på admin siden ved å skrive den inn i url på nettleseren, vil det igjen bli sjekket om den brukeren som er valgt er admin eller ikke. Dette fikk vi til ved å bruke localStorage, som lagrer informasjon på en side uansett om man går ut og inn av siden. Denne måten å gjøre det på er ikke sikker i en ordentlig løsning, men blir brukt i dette tilfelle for å simulere en session, informasjonen som blir lagret i localStorage blir hentet rett ut fra databasen, og om brukeren ikke er admin i databasen, så vil man ikke ha tilgang til denne siden.

For å representere aksje handel, og for å utfordre oss selv enda mer, ønsket vi at ved kjøp og salg av aksjer, så ser man at pengene forandrer seg. Ved vil prisen bli sendt inn til databasen, og saldo (Balance i tabell Kunder) bli oppdatert automatisk basert på antallet, og dette vil bli vist på nettsiden. Ved salg av aksjer, vil man kunne velge hvor mye man ønsker å selge, og saldo i databasen vil bli oppdatert og vist på nettsiden. I admin panelet, kan man slette kunder, og selge aksjer for hvilken som helst bruker. Dette har du bare tilgang til om du har rolle Admin. Man kan ikke slette seg selv, og dersom man sletter en kunde som er tilknyttet til en aksje, vil den aksjen også bli slettet fra databasen. Om man selger en aksje, vil hele summen for antallet bli returnert til brukeren den aksjen er tilknyttet til.

Konklusjon

Oppgaven ønsket at vi skulle lage en aksje side, der man kan kjøpe og selge aksjer, og det skal støttes CRUD i oppgaven. Dette er noe vi har oppnådd. Likevel ønsket vi å utfordre oss selv enda mer med oppgaven, og forlenge oppgaven og på best mulig måte klare å visualisere en aksjehandel side uten å faktisk implementere et login system, og dette gjorde vi gjennom å bruke localStorage, hvor informasjon fra databasen ble lagret. Dette er ikke en løsning man ville ha brukt med tanke på sikkerhet, fordi localStorage kan bli endret fra konsollen i nettleseren. Oppgaven er også dynamisk, og å hente flere aksjer fra APIen og legge de inn i databasen kan gjøres ved å endre antall på en variabel, og hele nettsiden vil fortsatt være funksjonell. Det er ikke anbefalt å bytte på variabelen, for noen deler av APIen mangler logo, eller informasjon om aksjene, som vil føre til at informasjon ikke hentes ut ved å klikke på «View More». Vi kan derimot redusere antallet på variablene aksjeView, som er Default satt til 9 i index.js, og da vil vi se et antall aksjer basert på hva vi setter denne variabelen til.

13 av 14

Kilder:

- Bluescape.com, 2019 “UML Diagrams – Everything You Need to Know to Improve Team Collaboration”: https://www.bluescape.com/blog/uml-diagrams-everything-you-need-to-know-to-improve-team-collaboration
- *Twelve data API documentation (twelvedata)*. (n.d.). RapidAPI - The Next Generation API Platform. https://rapidapi.com/twelvedata/api/twelve-data1
- *ApexCharts.js*. (2020, March 7). ApexCharts.js. https://apexcharts.com/Her er lenken til
- UML-diagrammet: https://miro.com/app/board/uXjVPPxGHEs=/?share_link_id=768878939215

14 av 14