Oppgave 1
ITPE3200 Webapplikasjoner
Høsten 2022

Henrik Larsen - s341878
Adrian August Grøtter - s354378
Atif Aziz - s354542
Jørgen Berntsen - s354410

Innhold

Innledning

Verktøy brukt i prosessen

Visual Studio (utvikling backend)

Rider (utvikling backend)

WebStorm (utvikling frontend)

Figma (Prototype verktøy)

GitHub

Postman

Diagrams.net

Db browser (SQLite)

Rammeverk/biblioteker brukt

C# - asp.net core 6

React

Typescript

Polygon.io (Web API til aksjer)

Object-Relational Mapping (ORM)

Tailwind

Kravspesifikasjon

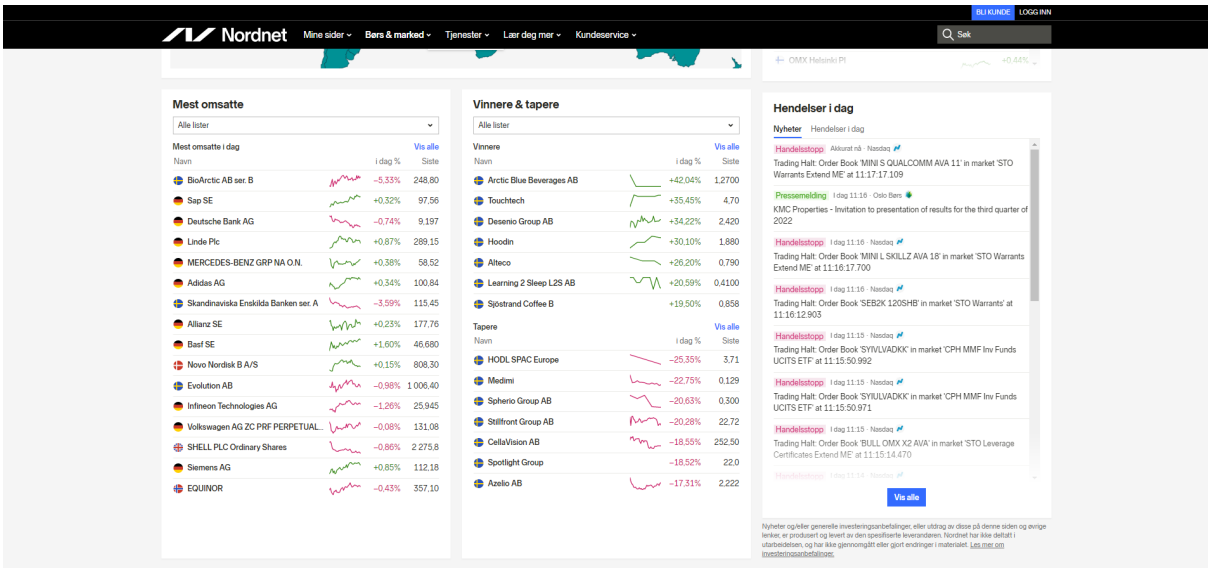
Utgangspunkt
Den overordnede kravspesifikasjonen har fra start vært å utvikle en MVP.
Kravspesifikasjonen for MVP'en ble definert i oppgaven vi fikk, også bygde vi på noen krav ved å idemyldre i gruppen.

- Kunden skal ha et bredt utvalg av aksjer å velge mellom
- Skal ha CRUD funksjonalitet på minst en entitet
- Kunne se graf historikk av en aksje.
- Kunne kjøpe og selge aksjer.
- Kunden skal kunne se, endre transaksjoner og slette transaksjoner dersom det er samme dag som den ble kjøpt.
- Se portfolioen av aksjer, samt totale verdien.
- Gi kunden en oversikt over hvilke aksjer som har gjort det bra og dårlig den siste tiden.
- Kunden skal kunne se direkte data, hentet fra et tredjeparts API

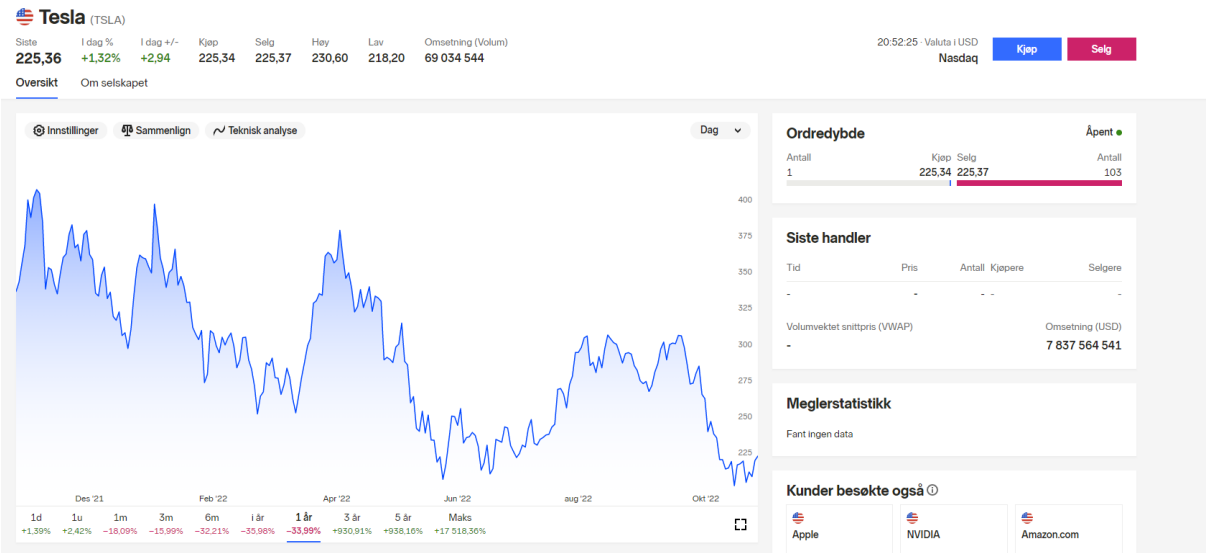
Utviklingsprosessen

Oppstartsfasen

Vi startet med å se litt på hvordan tilsvarende aksjehandel sider ser ut og hvilke funksjoner de har og hva som er viktig for å ha en fungerende aksjehandel side.

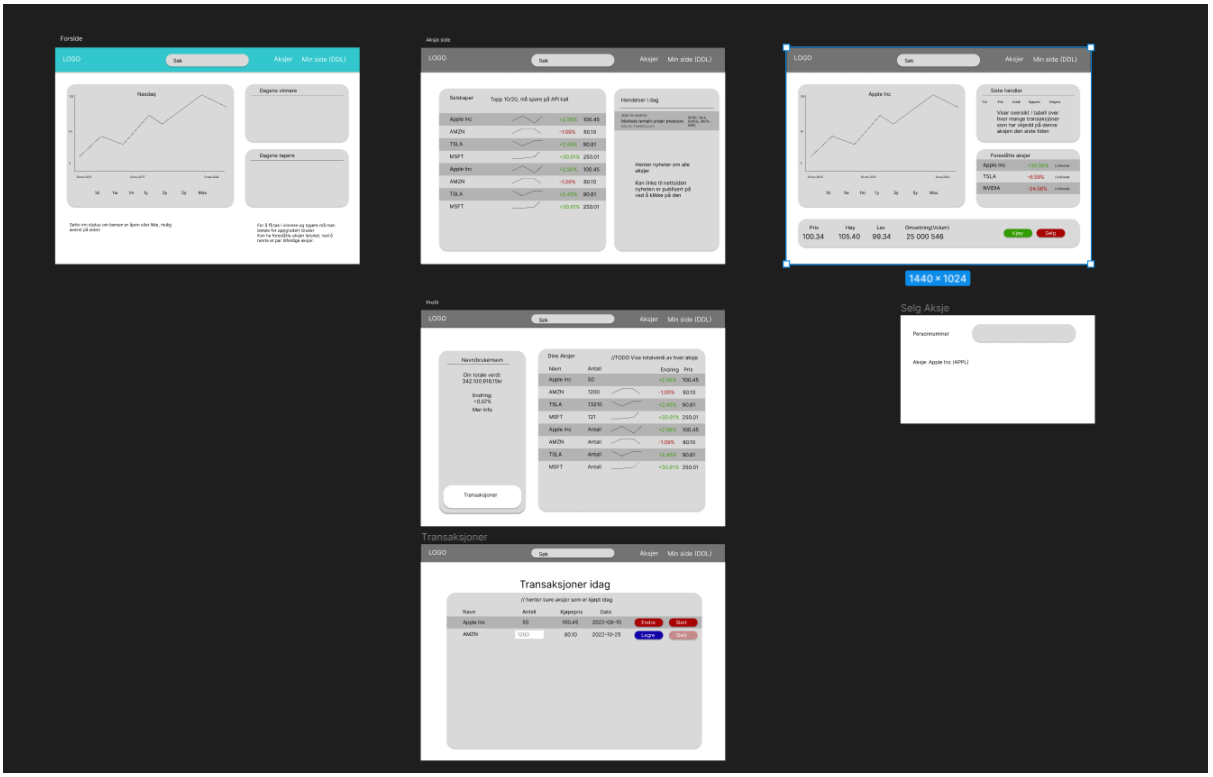


Bildet viser Nordnet sin børsen i dag side (Nordnet.no)



BIldet viser Tesla sin aksje side på Nordnet

Deretter skisset vi ned en rask prototype før vi lagde en prototype i figma som vi har fulgt mye av veien. Vi valgte å lage prototypen uten farger siden vi ikke hadde bestemt oss for hvilken fargepalett vi skulle gå for.



Funksjonalitet

Brukerhistorier

1. Som bruker ønsker jeg å kunne se alle mine transaksjoner.
2. Som bruker ønsker jeg å kunne kjøpe og selge aksjer.
3. Som bruker ønsker jeg å kunne slette og endre mine nylige transaksjoner.
4. Som bruker ønsker jeg å se alle aksjer det er mulig å kjøpe i en liste med alle aksjer.
5. som bruker ønsker jeg å velge antall aksjer jeg vil kjøpe.
6. Som bruker ønsker jeg en oversikt over aksjenyheter for relevante aksjer.
7. Som bruker ønsker jeg å kunne se oversikt over mine data som saldo, navn, personnummer og portefølje verdi.

Utførelse av brukerhistorier

1. Under ser man bildet fra profilsiden til brukeren. Helt neders på siden vil man finne en tabell som inneholder alle transaksjonene

2. Etter at en bruker har klikket på en aksje de ønsker å kjøpe vil de bli ført videre til siden “single Stock Page”. Her vil brukeren kunne kjøpe aksjen og selge den hvis de har den i porteføljen

3. Brukeren vil kunne få mulighet til å slette og endre sine nylige transaksjoner og dette kan utføres inne på profilsiden. Her vil en liste med transaksjoner vises og de nyeste vil kunne ha mulighet til å slettes og endres. Når edit-knappen trykkes vil antall feltet bli gjort om til et inputfelt, og brukeren kan endre antall aksjer som de ønsker å kjøpe. Slett Knappen vil slette hele transaksjonen.

4. Når brukeren ønsker å kjøpe en ny aksje kan de trykke på “Stocks” i “Navbar’en”. De vil da bli ført videre til siden “Stocks” hvor alle aksjene vises som en liste i en komponent.

5. Når brukeren har valgt den aksjen de ønsker å kjøpe kan dem i en komponent velge antall de ønsker å kjøpe. De kan legge inn antallet og totalprisen vil endres dynamisk etter hvor mange de velger.

6. Inne på Stocks siden får brukeren oversikt over de nyeste nyhetene fra aksjeverden. Hver nyhet inneholder tittel fra artikkel og forfatter samt en oversikt over hvilke aksjer som er nevnt i artikkelen.

7. På profilsiden har får brukeren en oversikt over sine data, som bl.a. personnummer, porteføljeverdi, saldo, og navn.

Backend

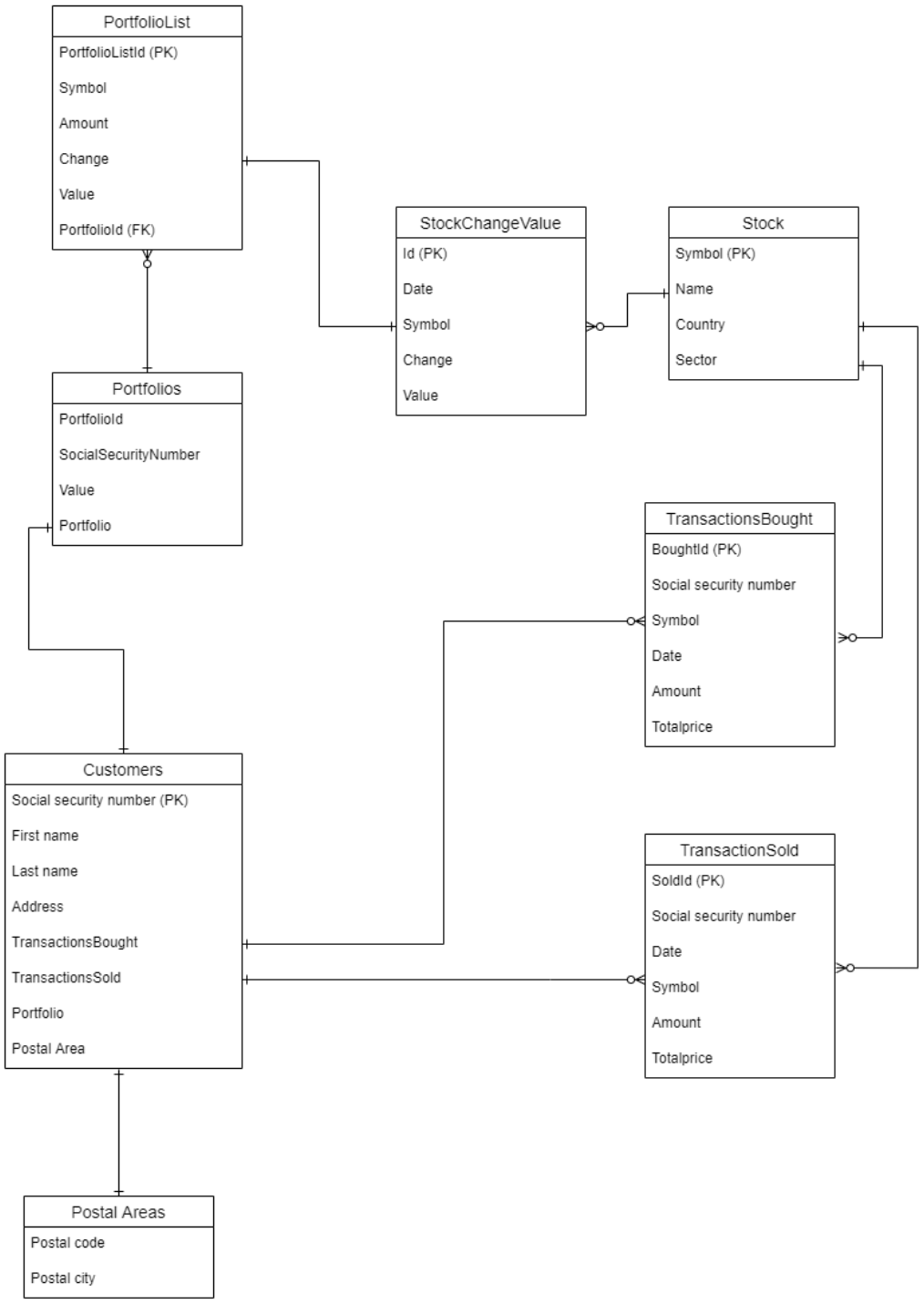
For å teste det nyeste og utfordre oss selv bestemte vi oss for å bruke asp.net core 6.0 istedet for asp.net core 3.1. Det var litt å sette seg inn i hvordan program.cs filen skal konfigureres og hvordan man skal logge til fil, siden det var annenledes fra asp.net core 3.1.

Database

Vi bruker entity framework og en sqlite server til å lagre dataene som blir behandlet på backenden, dataene lagres i myDb.db. Vi måtte konfigurere databasen i program.cs med denne kommandoen.

```
builder.Services.AddDbContext<StockContext>(options => options.UseSqlite("Data source=myDb.db"));
```

Selve databasen settes opp i StockContext.cs, hvor vi definerer hvilke tabeller vi skal ha med og hvilke kolonner vi skal ha i tabellene. Når vi skal laste dataene bruker vi “lazy loading” som vil si at vi at vi ikke henter ut relasjonene, dette bidrar til at spørringene vil gå raskt siden vi kun får den informasjonen vi trenger.



ER diagram av databasen

Databasen er satt opp slik at alle transaksjoner som er kjøpt vil lagres i transactionsBought og hvor gang man selger vil det opprettes en transaksjon i transactionSold. Dette slås sammen i portfolioen, så man har en liste med totalt antall av alle aksjene man eier og den totale verdien.

Funksjonalitet

```
2 references
Task<List<Stock>> GetAllStocks();
2 references
Task<StockPrices> GetStockPrices(string symbol, string fromDate, string toDate);
2 references
Task<bool> BuyStock(string socialSecurityNumber, string symbol, int number);
2 references
Task<bool> SellStock(string socialSecurityNumber, string symbol, int number);
2 references
Task<List<Stock>> ReturnSearchResults(string keyPhrase);
2 references
Task<List<Transaction>> GetAllTransactions(string socialSecurityNumber);
2 references
Task<bool> UpdateTransaction(Transaction transaction);
2 references
Task<bool> DeleteTransaction(string socialSecurityNumber, int id);
4 references
Task<StockChangeValue> StockChange(string symbol);
2 references
Task<List<StockOverview>> GetStockOverview();
2 references
Task<Customer> GetCustomerPortofolio(string socialSecurityNumber);
2 references
Task<List<StockChangeValue>> GetWinners();
2 references
Task<List<StockChangeValue>> GetLosers();
2 references
Task<News> GetNews(string symbol);
```

Metodene i interfacet

Vi opprettet et interface til repositoriet så man enkelt skal kunne se hvilke metoder vi har implementert.

Den entiteten vi valgte å ha CRUD funksjonalitet på er transaksjoner siden det skal være mulig å opprette en transaksjon ved å kjøpe en aksje. Lese transaksjonene i tabellen nederst på profilsiden. Her kan man også endre og slette transaksjonene hvis det fortsatt er samme dato som den ble kjøpt. Her har vi tenkt at transaksjonen ligger i en avventende stadie hvor kunden kan kansellere ordren eller endre på antallet på aksjen. Dette er for at kunden har betalt en kurtasje og vil få dette refundert dersom transaksjonen ikke har gått gjennom.

Direkte data fra API

For å hente data bruker vi et API fra Polygon.io, vi så først på et gratis API fra NasDaq, men det var vanskelig å sette seg inn i hvordan man hentet ut dataene fra de ulike aksjene. Det i kombinasjon med at vi fant lite dokumentasjon valgte vi Polygon sitt API. Ulempen med dette var at vi har kun 5 API kall i minuttet, som ble et problem når vi skal hente alle aksjene sin pris i dag. Det gjorde at vi må lagre dataene i databasen hvis dataene ikke har blitt hentet den dagen. For å kunne gjøre flere kall til API-et la vi inn flere API-nøkler i en liste så den bytter på hvilken nøkkel som skal brukes.

Logging og feilhåndtering

Logging av feilmeldinger skjer til log-filer inne i Log mappen ved å bruke Serilog. Her vil det lages en ny log-fil for hver dag for at det skal være lettere å finne feilmeldingen dersom man vet at den har skjedd på en spesifikk dag.

```
"Serilog": {
  "Using": [ "Serilog.Sinks.File" ],
  "MinimumLevel": {
    "Default": "Information"
  },
  "WriteTo": [
    {
      "Name": "File",
      "Args": {
        "path": "./Log/webapp-.log",
        "rollingInterval": "Day",
        "outputTemplate": "[{Timestamp:yyyy-MM-dd HH:mm:ss.fff zzz} {CorrelationId} {Level:u3}] {Username} {Message:lj}{NewLine}{Exception}"
      }
    }
  ]
}
```

Konfigurering av appsettings.json filen

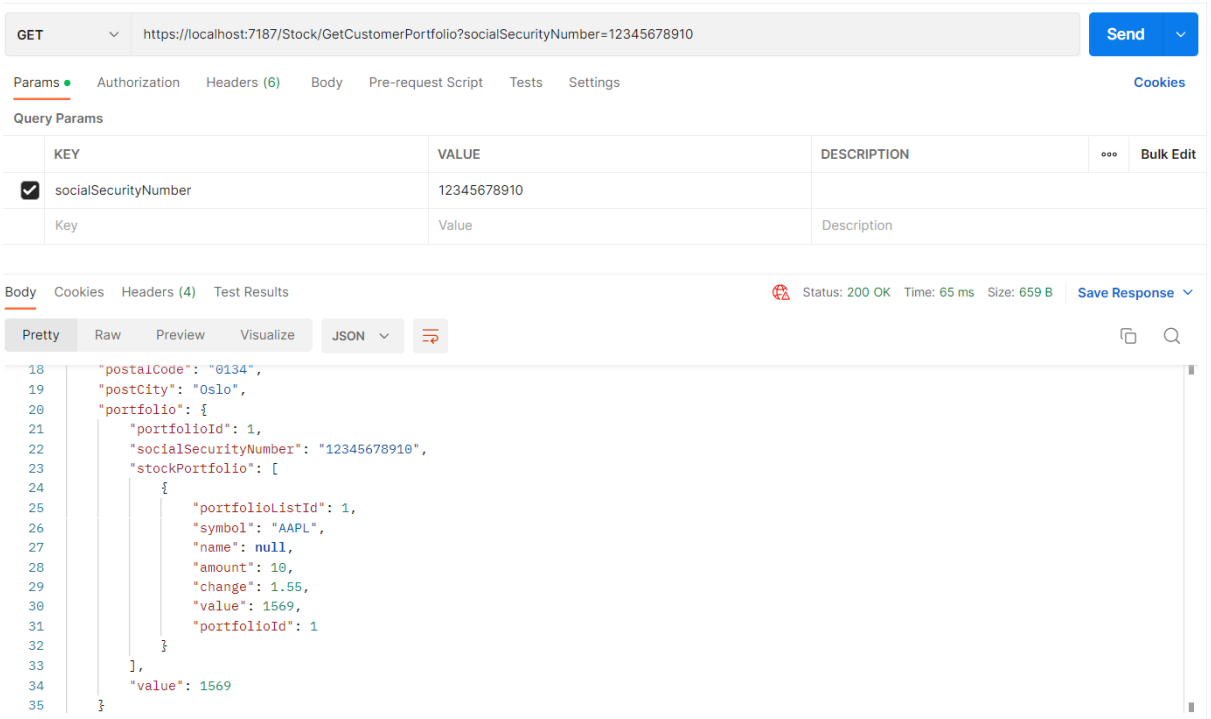
```
public async Task<ActionResult> GetAllStocks()
{
    var allStocks = await _db.GetAllStocks();
    if (allStocks == null)
    {
        _logger.LogInformation("Not found");
        return BadRequest("Not found");
    }
    return Ok(allStocks);
}
```

Eksempel fra metode med logging og feilhåndtering

Alle metodene i StockController har har logging av feilmeldinger som vil si ifra dersom det har skjedd noe feil i repositoriet. Vi har også lagt inn feilhåndtering på alle metodene. Så de vil returnere en http bad request med en kort melding dersom en transaksjon ikke har gått gjennom eller den ikke får kontakt med databasen.

Testing av API

Når vi har utviklet har vi kjørt manuelle integrasjonstester ved bruke postman, dette har vært veldig nyttig når vi ikke hadde en fungerende frontend. Det gjør også at man kan sjekke om objektet sendes som forventet.



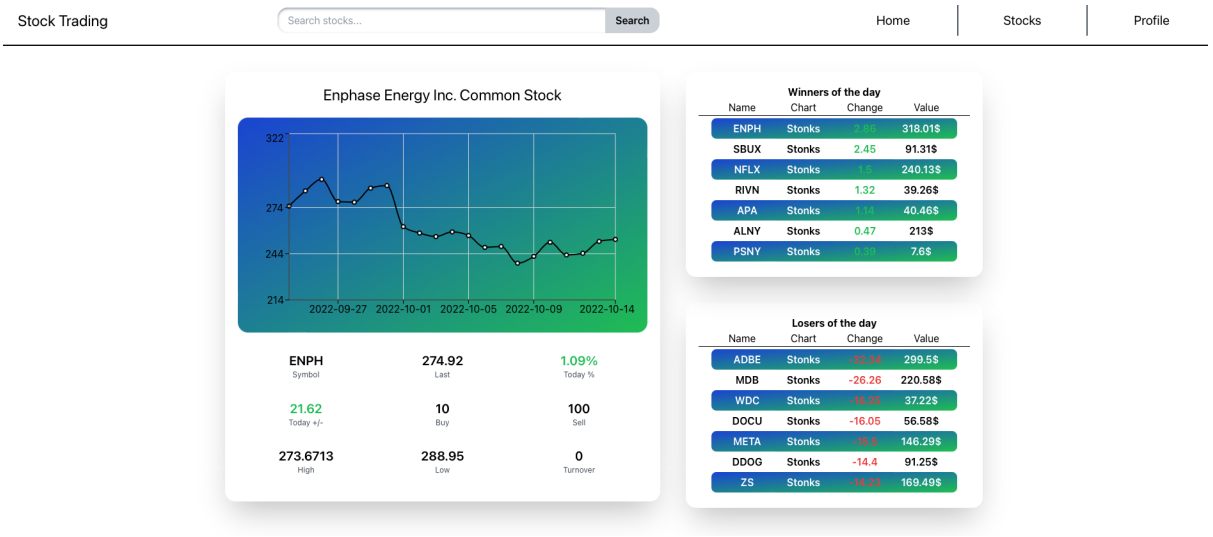
Kaller GetCustomerPortfolio i postman

Frontend

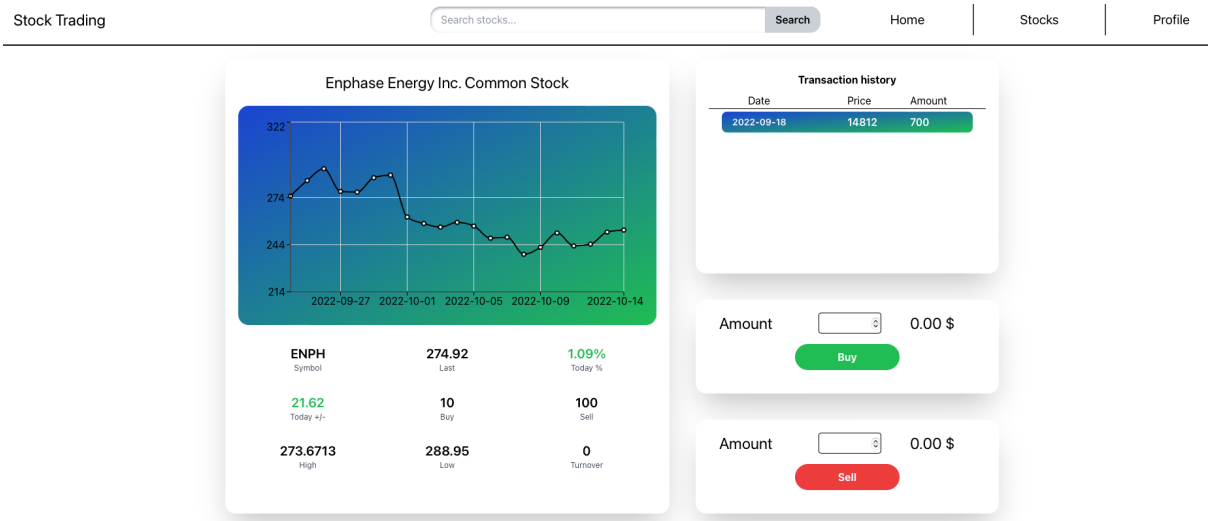
For å utfordre oss selv og ønske om å lære bransje relevante teknologier, bestemte vi oss for å gå for en TypeScript/React frontend. Vi har da med hjelp av denne teknologien laget en Single Page Application som lar brukeren se, kjøpe, selge og oppdatere sine aksjer. Vi har brukt Tailwind som CSS-rammeverk.

Ved å bruke React sitt komponentsystem kunne vi enkelt bygge modulære komponenter, som kunne gjenbrukes ved behov. For eksempel blir komponenten som viser aksjer i en liste brukt til å vise flere forskjellige data. Dette fungerer ved at vi kan sende inn hvilken data vi ønsker inn i komponenten som følger en satt struktur, og den vet hvordan den skal behandle og vise denne.

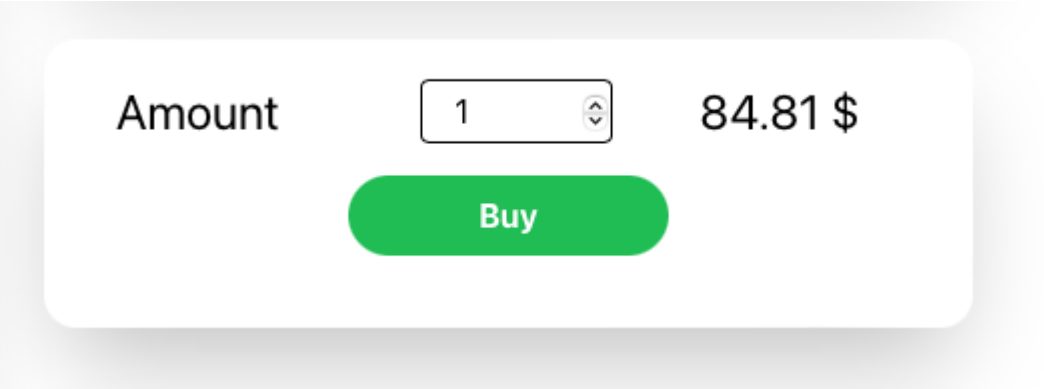
Første siden brukeren møter er hovedsiden viser 3 komponenter: dagens vinnere, dagens tapere og den største komponenten inneholder data om aksjen som ligger øverst på dagens vinnerlisten.



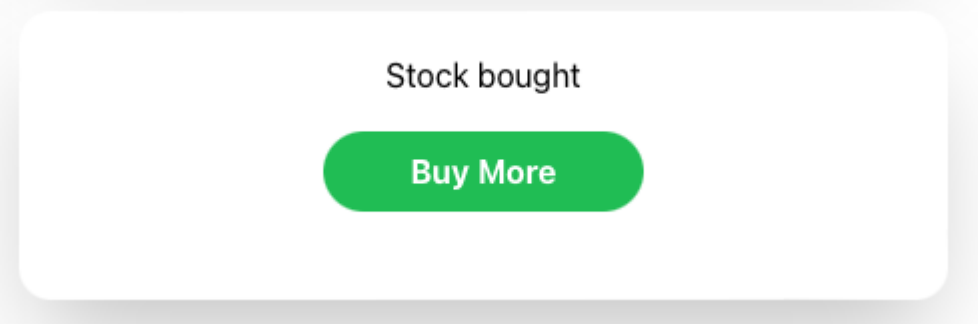
Hver aksje i begge komponentene, dagens vinnere og dagens tapere, er en link som tar deg til den aktuelle aksjens side. Her inne vil man se den aksjens data i en lignende komponent som er brukt på hovedsiden samt at man har mulighet til å kjøpe og selge aksjen. Man vil også kunne se den aktuelle aksjens transaksjonshistorikk.



Her kan brukeren legge inn ønsket antall aksjer de vil kjøpe. Totalprisen vil oppdatere seg etter hvor mange aksjer som blir lagt inn i input-feltet, og når brukeren trykker “*Buy*” vil denne transaksjonen bli lagt inn i brukerens transaksjonshistorikk.

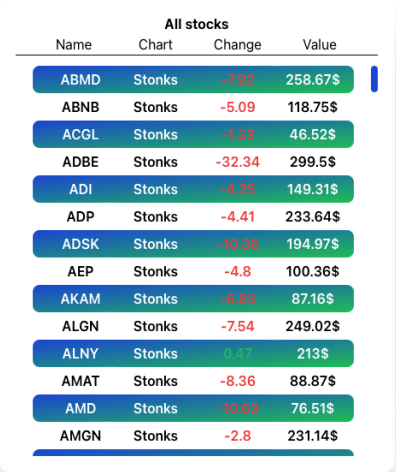


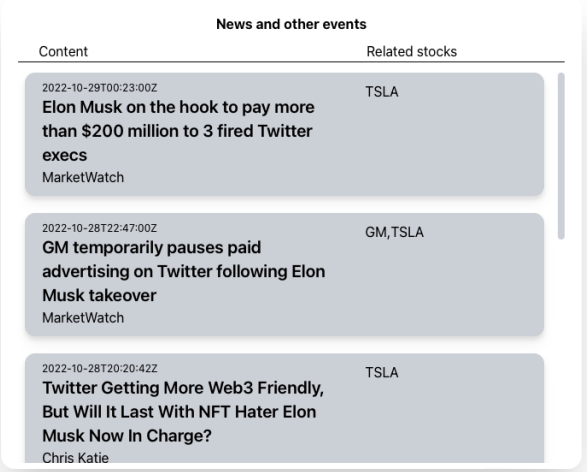
Når brukeren trykker på *"Buy"* så vil input tag`en forsvinne og det blir vist en tekst som viser at aksjen er kjøpt, og knappen vil endre tekst fra *"Buy"* til *"Buy More"*



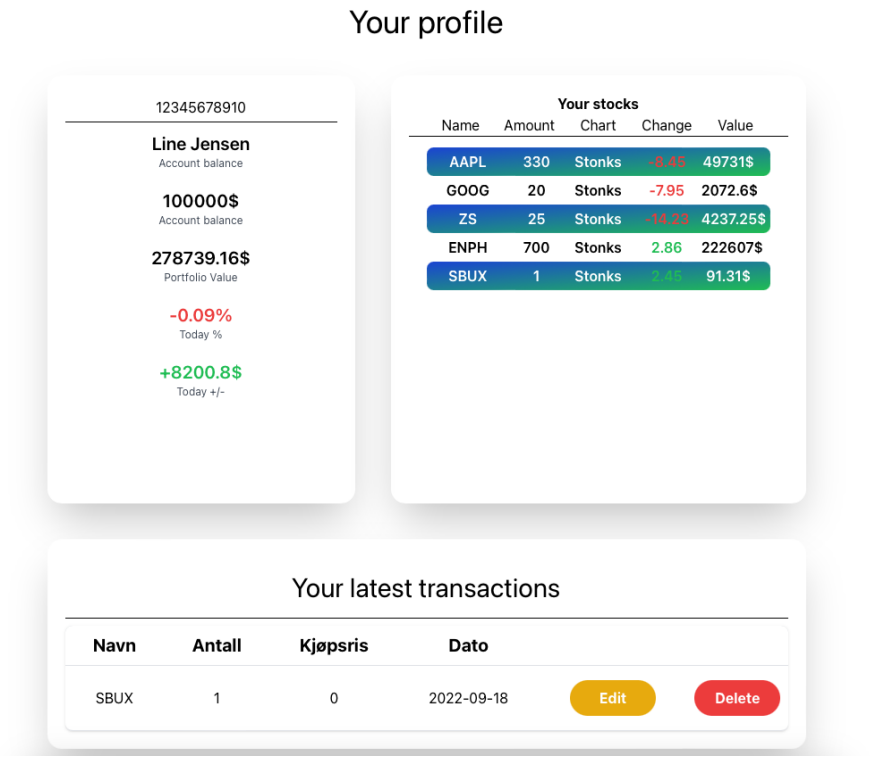
Knappen i navigasjonsbaren som heter "Stocks" tar deg til en side som har to komponenter. Den ene komponenten inneholder alle aksjene og den andre inneholder de nyeste nyhetene. Hver nyhet er en link som kan trykkes på for å åpne en ny fane med den aktuelle nettsiden.

All Stocks

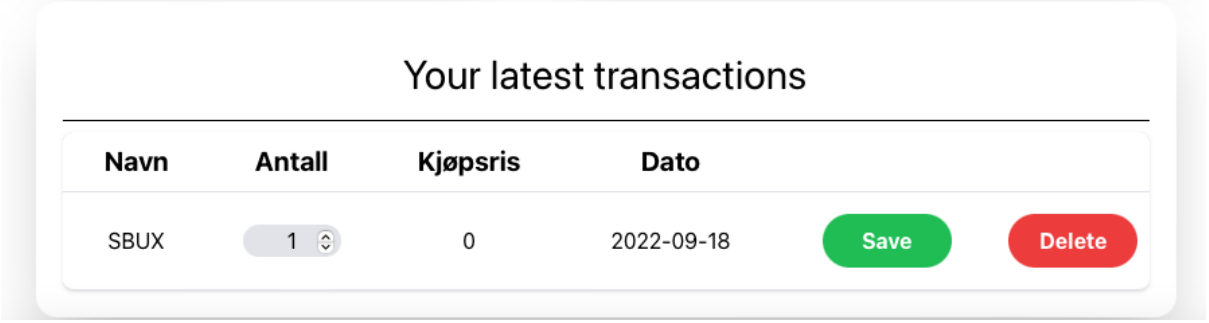




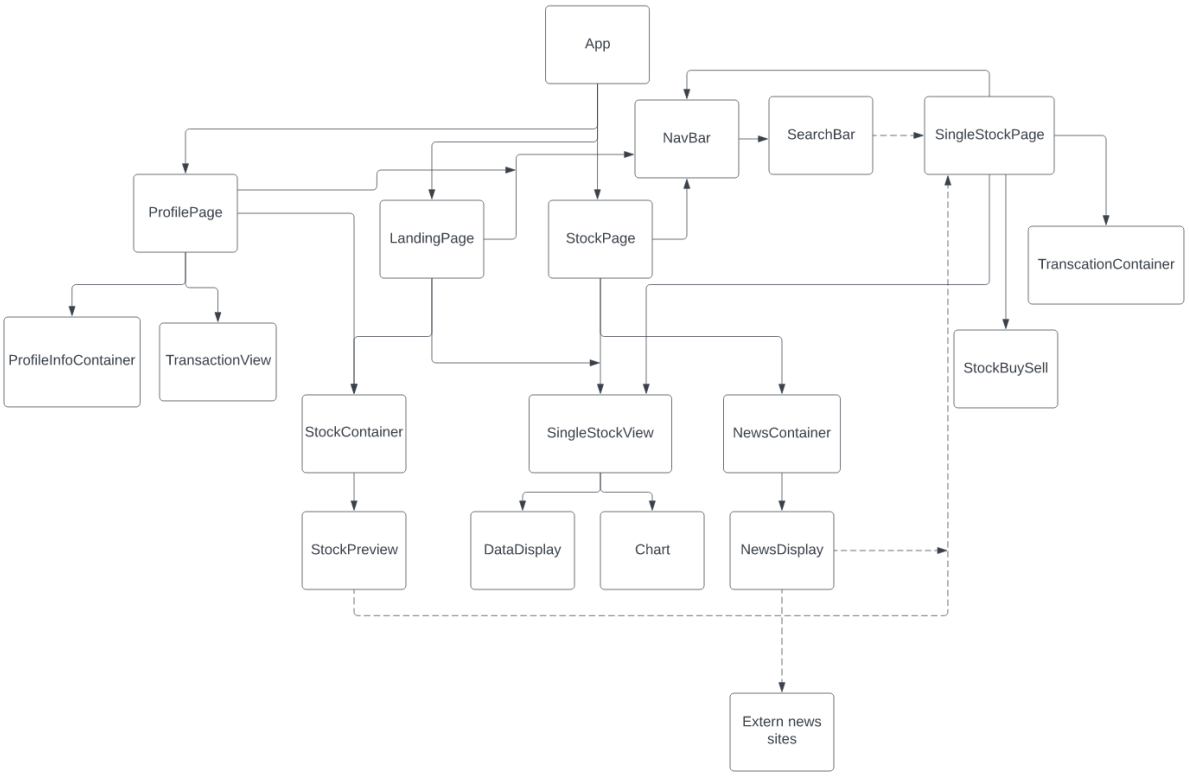
Den siste siden inneholder profilsiden. Her vil brukeren finne oversikt over alle aksjene de har kjøpt, sin egen bruker informasjon, samt den siste transaksjonshistorikken sin. I komponenten som inneholder transaksjonene har brukeren mulighet til å slette transaksjon og oppdatere den.



Hvis brukeren ønsker å oppdatere en av transaksjonene kan de trykke på edit-knappen. Da vil feltet som inneholder antall omgjøres til et inputfelt, og knappen vil skifte farge til grønn og teksten til å vise “save”.



Under har vi laget et diagram som viser hvordan React-komponentene henger sammen. En solid linje viser hvilken “child components” en komponent har, og stiplet linker viser hvordan en komponent blir åpnet ved hjelp av en link.



Konklusjon

Vi startet prosjektet med et utgangspunkt om å bruke flere teknologier ingen av oss har brukt før, med et sterkt ønske om å få mest mulig læringsutbytte og et resultatet vi så for oss.
Dette føler vi at vi virkelig har oppnådd, og vi er alle svært fornøyde, både med prosessen og sluttresultatet.
Det har vært en krevende prosjekt, med dypdykk i forskjellige dokumentasjoner, aksjer og skissering av design. Vi fikk implementert mye av hva vi ønsket oss, og ser frem til å implementere resten til neste prosjekt.