30.11.2022

ITPE2300 – Gruppeoppgave2

Dokumentasjon for Trading Trainer applikajsonen

Bernt Olsen (s341528), Awet Teklemarian, Bastien Testeniere, Okabamicheal Elias
OSLOMET

Innhold

1 Prosjekt beskrivelse

I forbindelse med gruppeoppgave 2 i kurset ITPE2300 – webapplikasjoner er det blitt utviklet en webapplikasjon som simulerer aksjehandel. Denne applikasjonen er basert på den foregående oppgaven (gruppeoppgave 1) der det ble utviklet en MVA. De største endringene fra MVA en er at det er blitt utviklet en helt ny frontend i React (med typescript). Videre er det lagt til mer lagdeling på server siden ved at BLL (business logic layer) er lagt til. Merk at databasemodellen, AlphaVantageInterface og EcbCurrencyInterface ikke er endret fra forrige oppgave. Se dokumentasjon for gruppeoppgave 1 for mer detaljert informasjon om den delen av applikasjonen.

For å kunne teste programmet er det lagt til en test bruker i databasen. Man kan benytte følgende innloggingsinformasjon:

Brukernavn: DevUser@test.com

Passord: Password1#

Dersom databasen opprettes på nytt ved å slette TradingSchema.db og kjøre dotnet ef database update, vil passordet være: Password1

1.1 Produkt mål

I grunnlaget kommer funksjonalitets krav til gjeldende produkt fra oppgavebeskrivelse av prosjektet. De sentrale funksjonelle kravene for produktet inkluderer:

- Lage en web applikasjon som kan tilby kjøp og salg av ønskende aksjer for brukere
- Applikasjonen skal ha en Single Page Application ved bruk av React.
- Applikasjonen skal ha en innlogging og registrering form
- Input validering av både klient og tjener siden er på plass
- Applikasjonen løser Feilhåndtering og logging
- Enhetstest på valgte metoder er implementert.

1.2 Brukte teknologier/ressurser

Programmeringsspråk

- C#
- Typescript
- HTML
- CSS

Bibloteker og rammeverk

- ASP.NET core web API 6.0
- Entity framework core
- React 16.15.1
- Bootstrap 5.0
- XUnit
- Moq
- Materials UI

IDE

- Visual Studio Enterprise 2022

Andre verktøy

- Versjon kontroll: GitHub, git
- Database: SQLite, sqlite browser

Merk at node 16.15.1 er blitt brukt for å opprette react prosjektet. Malen create-react-app er brukt for å opprette prosjektet.

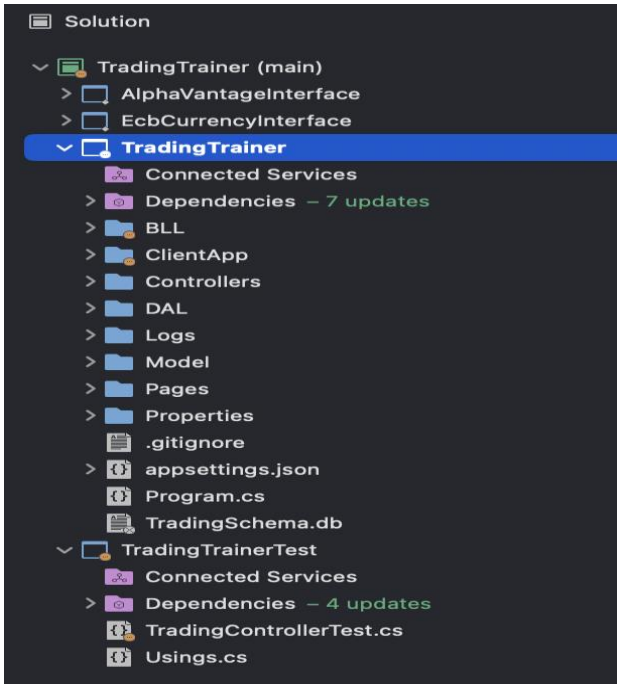
For å kunne starte spa applikasjonen sammen med .net core applikasjonen har vi benyttet SpaProxy og implementert en proxy (setupProxy.js under CLientApp/src) som omdirigerer http kall til port 5000.

1.3 Applikasjonsarkitektur

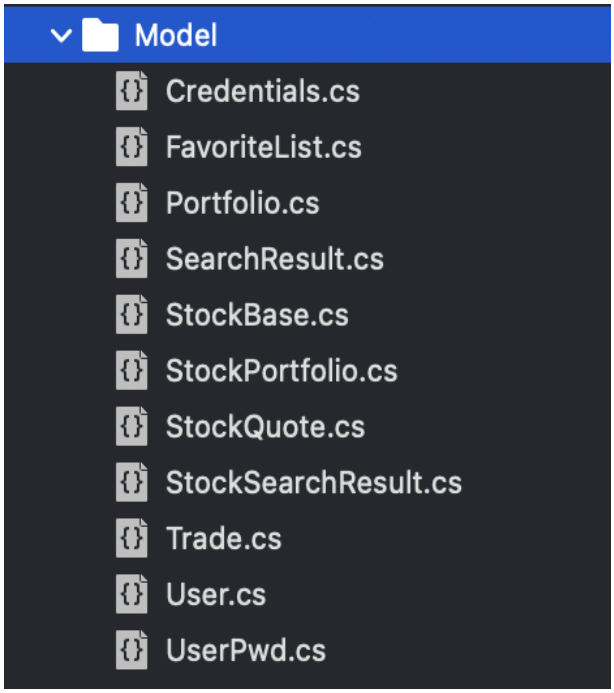
1.3.1 filstruktur

Applikasjonen består av klient og tjener side. Applikasjonen er delt in i ulike lag (lagdeling). Server siden er delt inn i Controller som utgjør grensesnittet mot klienten, Business Logic Layer som inneholder business logikken og Repository som utgjør grensesnittet mot databasen. Merk at det er TradingTrainer mappen som inneholder .NET core web api prosjektet som skal kjøres. Det er også lagt til tre andre klasse biblioteker i prosjektet: AlphaVantageInterface, EcbCurrencyInterface og TradingTrainerTeset.

I Model mappen finner vi alle entity objektene som applikasjonen skal bruke til å mappe de tilsvarende tabellene i entity frameworks. Klassene benyttes for å representere dataen fra databasen og deretter kommunisere med klientsiden.



Figur 1 - Filstrukturen til TradingTrainer



Figur 2- Model mappen med objekter som brukes i kommunikasjon med klienten

1.3.2 Frontend

ClientApp inneholder alle komponentene react applikasjonene består av (se figur 4). Vi brukte her react, Typscript , JavaScript og css kode. Merk at all css kode for applikasjonen er lagt til i index.css. Kjernen av React applikajsonen er Main.tsx. Denne filen implementerer react routing ved metoden createBrowserRouter. Videre blir RouteProvider komponenten returnert av Main funksjonen som videre implementeres av index.tsx (app root). Figur 3 viser stien til de ulike komponentene og de tilhørende komponentene som legges til avhengig av stien som det navigeres til. For å navigere mellom disse komponentene har vi tatt i bruk useNavigate hook'en. Dermed kan man navigere programmatisk ved «navigate(«path»)», forutsatt at vi har lagt til «const navigate = useNavigate(«path»)».

```
//const [isAuthenticated, setIsAuthenticated] = useState(false);
// The routing feature is created with information from https://reactrouter.com/en/main/
const router = createBrowserRouter([
    {
        path: "/",
        element: <PrimaryContainer />,
        errorElement: <ErrorComponent />,
        children: [
            {
                path: "/",
                element: <LandingComponent />,
            },
            {
                path: "/login",
                element: <LoginForm
                            SetUser={setAuthenticatedUser}
                            User={authenticatedUser}
                            SetErrorMsg={setErrorMsg}
                            //SetIsAuthenticated={setIsAuthenticated}
                        />,
            },
            {
                path: "/registerForm",
                element: <RegisterForm />,
            }
        ]
    },
```

Figur 3 - React router

Når programmet startes, vil LandingComponent rendres. For å registrereen bruker navigeres det til «/registerForm» og RegisterForm blir rendret.

Når brukeren er autentisert react router sørge for at AppContainer komponenten rendres med TradingDashboard som barn (TradingDashboard injiseres i «Outlet» componenten i AppContainer). I TradingDashboard er det mulig å benytte ulike funksjoner som er fordelt på ulike komponenter under TradingComponents mappen. Ved oppstart vil Watchlist og StockQuote komponenten rendres. For å rendre StockMarket (inneholder søk funksjonen), TradeHistory eller Portfolio må det trykkes et av følgende felter:

Hver av de ulike komponentene inneholder koden som implementerer funksjonaliteten. I enkelte tilfeller er det lagt til setState props på komponentene dersom andre komponenter trenger tilgang til data fra komponenten. Et eksempel er når bruker velger en aksje i watchlist eller portfolio. Da settes følgende props:

```
    break;
default:
    // Render the wathclist component per default.
    stockList = <Watchlist
                    SetStockListTab={setStockListTab}
                    SetCurSelectedStock={setCurSelectedStock}
                    // Sending down callback to update the quote - used by the watchlist row component
                    UpdateQuoteDisplay={updateQuoteDisplay}
                    CurSelectedStock={curSelectedStock}
                    User={props.User}
```

Mappen Settings under /CleintApp/src, grupperer alle komponentene som brukes for å levere change settings funksjonaliteten i frontend. De består av hovedkomponenten MainSettings (endring av navn, email, currency), PwdReset for endring av passord og ConfirmReset for å bekrefte at brukeren skal resettes. Merk at ved reset vil portfolio, watchlist, tradehistory og verdier brukeren har opparbeidet seg, tilbakestilles. Alle de nevnte komponentene styres av Settings komponenten som igjen kan navigeres til med «/tradingDashboard/settings». Definert i router i Main.tsx:

```
        />,
    },
    {
        path: "/tradingDashboard/settings",
        element: <Settings
                    User={authenticatedUser}
                    SetUser={setAuthenticatedUser}
                    SetErrorMsg={setErrorMsg}
                />
    }
]
```

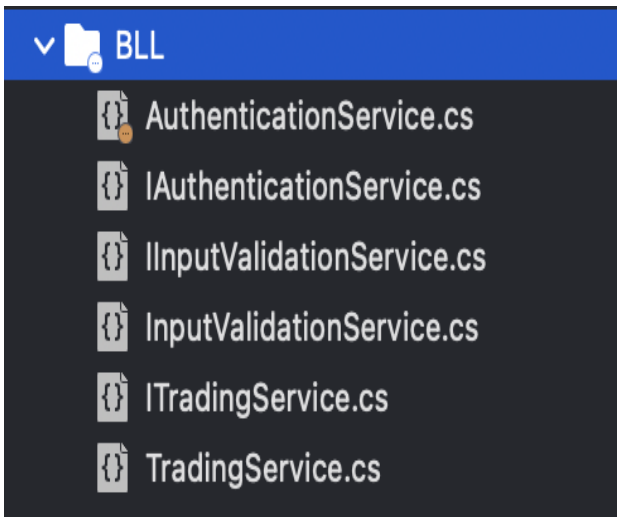
Figur 4- Filstrukturen til frontend applikajsonen (react typescript)

1.3.3 Controller

Controller laget består i vårt tilfelle av kun TradingController.cs. Denne filen inneholder alle rest endepunktene som benyttes av klientsiden. Oppgaven til denne klassen er å håndtere kommunikasjonen med klienten. Det vil si returnere data, håndtere feil som for eksempel feil input og returnering av http feilkoder (400 og 500 koder) for å indikere feil som har oppstått.

1.3.4 Business Logic Layer

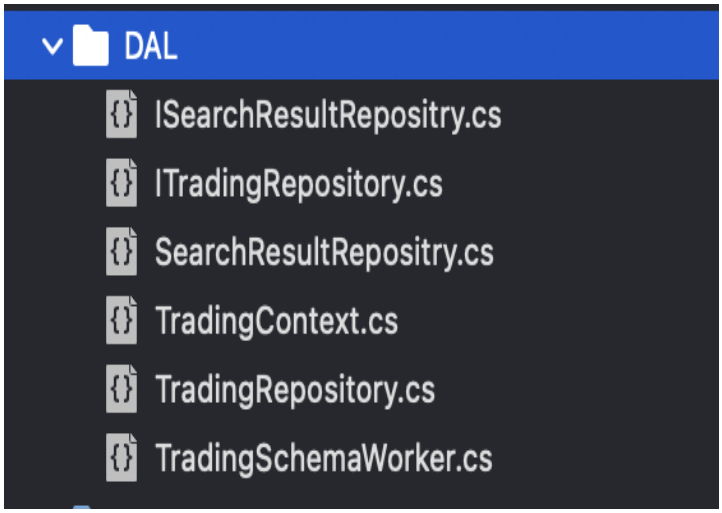
I dette laget har vi lagt til all koden som behandler data eller legger til data, mellom controller og repository laget. Det består av tre klasser i dette tilfelle ApplicationService, TradingService og InputValidationService (se figur 5). TradingService benytter metodene i InputValidationService for å validere input. Videre inneholder TradingService business logic for applikajsonen. Det er for eksempel metoder som oppretter bruker portfolio (sammendrag av verdiene brukeren eier), oprettelse av stock quotes (ved hjelp av AlphaVantage) eller metoder for salg og kjøp av aksjer.



Figur 5- Business logic layer

1.3.5 – Data Access Layer (DAL)

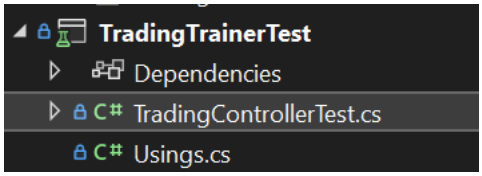
Dette laget inneholder database konteksten og repository klasser. Repository klassene utgjør ulike grensesnitt til databasemodellen. Denne delen ble utviklet i gruppeoppgave 1 og er ikke blitt oppdatert i gruppeoppgave 2 (se dokumentajson 1 for mer informajson om denne delen av applikajsonen).



Figur 6 - Data access layer

1.3.6 – Unit test Klassen (TradingServiceTest)

Det er blitt implementert litt enhetstesting på business logikken. Denne kan man finne i TradingControllerTest.cs (se figur 7). Vi har lagt til enhetstest av de metodene det virket mest hensiktsmessig å teste. Om man skulle investere mer tid i dette prosjektet hadde vi jobbet mer intensivt med dette punktet.



Figur 7- Unit tests

1.3.6 Data fra Alpha Vantage API

DENNE DELEN ER FORKLART I OPPGAVE 1. SIDEN DET ER IKKE NOE NØDVENDIG ENDRING, VURDERTE VI IKKE Å BESKRIVE IGJEN.

1.3.7 Beskrivelse av EcbCurrencyInterface

DENNE DELEN ER FORKLART I OPPGAVE 1. SIDEN DET ER IKKE NOE NØDVENDIG ENDRING, VURDERTE VI IKKE Å BESKRIVE IGJEN.

1.3.8 Databasemodellen

DENNE DELEN ER FORKLART I OPPGAVE 1. SIDEN DET ER IKKE NOE NØDVENDIG ENDRING, VURDERTE VI IKKE Å BESKRIVE IGJEN.

1.3.4 Migration og seeding av databasen

DENNE DELEN ER FORKLART I OPPGAVE 1. SIDEN DET ER IKKE NOE NØDVENDIG ENDRING, VURDERTE VI IKKE Å BESKRIVE IGJEN.

1.3.5 Hosted service for rydding av databasen

DENNE DELEN ER FORKLART I OPPGAVE 1. SIDEN DET ER IKKE NOE NØDVENDIG ENDRING, VURDERTE VI IKKE Å BESKRIVE IGJEN.

2 Beskrivelse av applikasjonens funksjonalitet

2.1 Beskrivelse av brukergrensesnittet

Vi lagde en applikasjon som folk kan bruke for å trene på kjøp og salg av aksjer med virtuelle penger. Brukere vil kunne få se utvikling av deres Portfolio gjennom tiden og hvor mye de har tapt eller vunnet totalt siden de startet. Applikasjon benytter seg av data fra finansmarked som er hentet via Alpha Vantage APIet. Brukergrensesnittet består av funksjonalitet for å søke på en aksje, kjøpe en aksje, selge en aksje eller legge til en aksje som favoritt. Man kan også finne oversikt over aksjer man eier, hvor mye man har tapt eller vunnet med hver aksje og de forskjellige transaksjon som har blitt utført. I den videreutviklede versjonen består brukergrensesnittet også av autentisering og registrering av nye brukere.

2.2 Register og Log inn

For dette prosjektet la vi til register og log inn funksjonaliteter. Når man starter applikasjon så havner man på en hovedside der man kan enten velge å logge inn eller å registrere en ny konto. Første gang bruker uten konto må registrere seg for kunne logge inn. Start siden register knapp bytter siden til et register form. Det oppnås ved bruk av router, register formen er en form med tekst felter for bruker informasjon. Hver felt er validert for riktig input når feltet forlates. Dette oppnås ved bruk av useState hooks, regex og error props av textfeltet. Tekstfeltene som er brukt på register kommer fra material ui, dermed tilbyr de props som error og helper text som kan manupuleres ved bruk av useState. Her settes state til hver eneste felt til true når validering feiler og dette state blir passert til error props slik at brukeren kan se at det er oppgitt feil input. All input spesifikasjonene kan man se på kilde koden. Etter utfylt form, når brukeren trykker på register, vil det sjekkes om alle feltene er utfylt, hvis ikke det skjer ikke noe ved siden og formen sendes ikke videre. Hvis det er oppgitt input som ikke består valideringen og bruker fortsatt trykker på register så er dette håndtert ved å skrive ut en melding øverst på formen som sier “formen må har feil format”. En annen tilfelle formen ikke sendes er, hvis confirm passord ikke stemmer med passord, så sende funksjonen sjekkes det om dette stemmer, hvis ikke så får brukeren en feil melding og formen sendes ikke. Hvis alt stemmer så går send funksjonen videre og mapper en bruker state objekt til bruker objekt. Dette objektet er da sendt til riktig endepunkt ved bruk av axios sin post metode.

Når man velger log in vil applikasjonen automatisk gjøre et kall til server (endepunkt "/trading/getUsername") for å sjekke om brukeren er autentisert fra før. Hvis ikke brukeren er logget in, vil man få muligheten til å logge på med brukernavn og passord. Merk at brukernavnet er email adressen brukeren er registrert på. Input i tekstfeltene valideres og sendes til server siden til endepunktet «/trading/login».

LoginForm: Metoden som brukes for å sende autentiserings forespørsel til server.

```
// This method executes the call to /trading/login through the fetch api.
const authenticateUser = async (usr: string, pwd: string) : Promise<void> => {
    const endpoint = "/trading/login";
    // Defining the payload to be sent to the server
    const credentials = {
        username: usr,
        password: pwd
    };
    // Make the authentication http patch request
    await authenticate(credentials).then((data) => {
        // Authentication was successfull
        if (authFailed) {
            //  If the authentication failed state is active, make sure to disable it.
            setAuthFailed(false);
        }
        setTimeout(() => {
            setIsWaiting({
                active : false,
                msg : defaultWaitMsg
            });
            props.SetUser(data);
        }, waitDelay)
    }).catch(errorResp => {
        // An error occured during authentication
```

På server vil passordet som sendes med brukeren sammenlignes med passordet som er lagret på brukeren i databasen. Dersom passordet ikke stemmer eller brukeren ikke blir funnet, returneres Unauthorized fra server. Sammenligning av passord skjer i AuthenticationService klassen. Her blir passordet hashet sammen med saltet som er lagret på brukeren i databasen. Dersom passordet stemmer vil «_login» flagget på bruker sesjonen settes til true og dermed indikere at sesjonen er aktiv. I program.cs er det spesifisert at sesjoner droppes etter 30 minutter inaktivitet:

```
//Adding sessions
builder.Services.AddSession(options => {
    options.Cookie.Name = "TradingTrainer.Session";
    options.Cookie.IsEssential = true;
    options.IdleTimeout = TimeSpan.FromMinutes(30);
```

Passord regler:

For at et passord kan registreres på brukeren må det bestå av minst et tall, en bokstav eller et spesialtegn (`$&+,:;=?@#|'<>-^*()%!`). Videre må passordet være større enn 9 tegn og mindre enn 32.

2.4 Server/Client input-validering

Vi har implementert inputvalidering på server og frontend. På server er inputvalideringen implementert i business logic layer ved klassen InputValidationService. Klassen består av metoder som kaster ArgumentExeption dersom input ikke er korrekt:

```
/**
 * This method validates the user id given as input to the trading controller.
 */
12 references
public bool ValidateUserId(int userId) {
    if (userId < 1)
    {
        // User id is not valid
        throw new ArgumentException("The provided userId is not valid. It must b
    }
    return true;
}
```

I Trading controller vil unntaket fanges det opp og Badrequest returneres til klienten:

```
catch (KeyNotFoundException userNotFoundEx)
{
    // The user was not found
    _logger.LogWarning("Endpoint GetUser: An exception has occured while trying to
        userNotFoundEx.Message);
    return NotFound(userNotFoundEx.Message);
}
catch (ArgumentException inputInvalid)
{
    _logger.LogWarning($"Endpoint GetUser: {inputInvalid.Message}");
    return BadRequest(inputInvalid.Message);
}
catch (Exception generalError)
```

På frontend valideres input som settes in i form elementer ved hjelp av onchange event og regex mønstre. Eksempel ved passord validering:

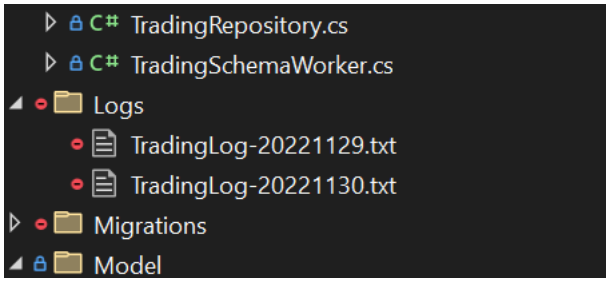
```
}

if (pwd.length < 9) {
    // The password length is invalid
    returnObj.IsValid = false;
    returnObj.ErrorMsg = "The password length is less than 9 characters!";
    return returnObj;
}
if (!(/\d/.test(pwd))) {
    // The passsword does not contain any numbers
    returnObj.IsValid = false;
    returnObj.ErrorMsg = "The password does not contain any numbers (0-9)!";
    return returnObj;
}
if (!(/\w/.test(pwd))) {
    // The password does not contain any word characters
    returnObj.IsValid = false;
    returnObj.ErrorMsg = "The password does not contain any word characters!";
    return returnObj;
}
if (!(/[\$\&\+\,\:\;\=\?\@\#\|\'\<\>\-\^\*\(\)\%\!]/.test(pwd))) {
    // The password does not contain any special characters
    returnObj.IsValid = false;
    returnObj.ErrorMsg = "The password does not contain any special characters!";
    return returnObj;
```

2.5 Enhetstester på server

I denne delen av enhetstesting ble det opprettet enhetstester for de relevante metoder med hensyn av tiden som finner i Klassen "TradingService" og vi har testet da om de absolutte returverdiene var i likhet med de foreventede returverdier for funksjonen/objektet. Når det er snakk om testmiljø, så vi har brukt for enhetstesting Visual Studio Enterprise som testverktøy. For å utføre enhetstestene ble det brukt også rammeverket Xunit og moq som er bibloteker.

2.6 Feilhåndtering og logging

I program.cs filen registrerte vi at logging av feil skal bli registrert i filen Logs/TradingLog.txt .



Feilhåndtering og logging er implementert i TradingController methodene med hjelp av try catch funksjonalitet. Videre forekommer det logging i business og data access laget. Merk at vi har benyttet LogInformation der det logges informasjon om program flyten (starten av håndtering av forespørsel, avslutning av håndtering av forespørsel). Videre er LogWarning benyttet for å logge håndterbare feil (eks feil input) og LogError for uventede feil (internal server error):

```
catch (Exception generalError)
{
    _logger.LogError("Endpoint GetUser: An exception has occured with getUser.\n" + gene
    return StatusCode(StatusCodes.Status500InternalServerError, generalError.Message);
}
```

Videreutvikling av applikasjonen

Dersom det skal investeres mer tid i denne applikasjonen er det en rekke ting vi ønsker å forbedre. For eksempel ønsker vi å gjøre programmet mer tilgjengelig for mindre skjermstørrelser. Videre ønsker vi å investere mer tid i testing (integrasjonstester og enhetstester), ettersom testene nå kun tester med forventede input verdier.