**Database modellen**

I dette prosjektet har vi benyttet Sqlite databasen og Entity Framework core som ORM (object-relational mapper) for å aksessere databasen fra .net core web applikasjonen. Databasemodellen som er blitt konstruert til denne oppgaven består av entitetene Users, Stocks, Trades, SearchResults og StockQuotes. Videre eksisterer det mange til mange forhold mellom Users og Stocks. En bruker kan eie flere aksjer og ha flere aksjer i sin favorittliste. Videre kan en aksje være i favorittlisten til flere brukere og være kjøpt av flere brukere. Videre er det et mange til mange forhold mellom SearchResults og Stocks ettersom flere aksjer kan være i et søkeresultat og en aksje kan være del av ulike søkeresultater.

For å representere disse mange til mange forholdene, opprettes hjelpetabeller (join tables). I utgangspunktet vil entity framework gjøre dette automatisk dersom et mange til mange forhold er definert implisitt i DbContext programmet. EF vil da sette sammen navn på tabell og attributter ut fra de to entitetene som inngår i forholdet. I vår databasemodell er alle navnene til hjelpetabellene definert eksplisitt: FavoriteLists, StockOccurances og StockOwnerships. Merk at StockOwnerships er en tabell som inneholder flere attributter en bare primær- og fremmednøklene (SpentValue og StockCounter). Derfor er denne tabellen definert som en egen entitet i TradingContext. For at EF skal opprette tabellen med både Symbol (aksje id) og UsersId som både primær og fremmednøkkel er det viktig at StocksId og UsersId er definert som attributter i tillegg til navigation property for både Stocks og Users tabellen.

**Database migrasjoner og seeding**

Med Entity Framework finnes flere metoder for å bygge databasen ut fra DbContext. I dette tilfelle har vi benyttet migrations funksjonaliteten i stedet for å benytte EnsureCreated og EnsureDeleted. Ved hjelp av migrations kan eksisterende data i databasen bevares ved endringer av database modellen. Videre er det kun lagt til et brukerobjekt som skal legges til i databasen når den opprettes (seeding). Hver gang det foretas endringer i database modellen (f.eks legge til en attributt) kan det opprettes en ny migrasjon med «Add-Migration migrationName». Databasen kan dermed bygges på nytt baserende på den siste migrasjonen med «Update-Database». Resultatet er en database med en oppdatert struktur samtidig som data som var lagret i den forrige versjonen er tilgjengelig i den nye databasemodellen.

**Worker tjeneste for rydding av databasen**

Lagring av søkedata kan føre til at mye data i form av aksje objekter og søkeresultat objekter er lagret i databasen uten at det blir aktivt brukt av brukere. I forbindelse med dette er det blitt lagt til en metode i TradingRepository: CleanTradingSchemaAsync(). Denne metoden fjerner søkeresultater som ble lagret for over 24 timer siden. Videre fjernes alle Stocks objekter og tilhørende StockQuotes objekter som ikke har noen relasjon til en bruker (portfolio eller favorittliste) eller et søkeresultat.

For at denne metoden kjøres en gang daglig, er det implementert en hosted service, «TradingSchemaWorker», som starter når applikasjonen startes og kjører i bakgrunnen som en uavhengig tjeneste. TradingSchemaWorker klassen er basert på Worker malen i dotnet. Videre defineres det et Timer objekt i StartAsync metoden, som starter en syklus der CleanTradingSchemaAsync() kalles når klokken til maskinen som kjører programmet er 00:00:00. Merk at «CleanTradinSchemaAsync» metoden kjøres en gang når ved oppstart av applikasjonen.

**Data fra Alpha Vantage API**

Applikasjonen benytter data fra finansmarkedet hentet fra Alpha Vantage APIet. Dette er et RESTful API som i utgangspunktet er gratis, med en begrensning i antall kall mot apiet med samme api-nøkkel. Det er ikke mulig å gjøre mer en 500 http-forespørsler mot tjenesten per dag og 5 forespørsler per minutt. I applikasjonen har vi benyttet endepunktene «Quote Endpoint» og «Search endpoint» Quote endpoint returnerer verdier knyttet til aksjens verdi og volum for forrige dag. Det er «Price» verdien vi benytter når brukere selger og kjøper aksjer i applikasjonen vår. Videre har vi benyttet Search enpoint for å gi brukere muligheten til å søke på aksjer og velge aksjer man eventuelt vil se nærmere på med quote endpoint.

Vi har valgt å lage et C# klassebibliotek for koden som benyttes for å håndtere kommunikasjon med Alpha Vantage. Fordelen med dette er at det er enklere å manuelt teste koden gjennom et testprosjekt og generelt gjenbruke koden i andre prosjekter. Videre er dette et tiltak som kan føre til høyere kohesjon og lav kobling i applikasjonen. Klassebiblioteket heter AlphaVantageInterface og består av klassen AlphaVantageConnection som inneholder metoder for å sende http forespørsler til Alpha Vantage, håndterer respons data og sørger for at begrensningene knyttet til apiet er håndtert.

AlphaVantageInterface biblioteket benyttes i applikasjonens TradingController klasse. Der opprettes det nye AlphaVantageConnection objekter hver gang det er nødvendig å hente data fra finans markedet. Api nøkkelen brukt i dette prosjektet er registrert på en student bruker og er lagt til i appsettings.json under AlphaVantageApi. Api nøkkelen er tilgjengelig gjennom et IConfiguration objekt som inneholder all nøkkel verdipar definert i appsettings filen. Objektet blir tilgjengelig i Kontrolleren gjennom dependency injection ved at en IConfiguration variabel spesifiseres som parameter i konstruktøren. Merk at api-nøkkelen må lagres på en annen måte før prosjektet eventuelt publiseres.

Grunnet begrensningene knyttet til http forespørsler, har vi valgt en økonomisk tilnærming til innhenting av data. Både quotes og søkeresultater blir mellomlagret i databasen. På denne måten unngås det at unødvendige forespørsler gjøres mot Alpha Vantage. Dette vil ikke fullstendig løse problemet knyttet til begrensing og det vil dukke opp en feilmelding når man for eksempel gjennomfører 6 forskjellige søk etter hverandre. Dersom applikasjonen skal støtte flere bruker sesjoner samtidig i fremtiden, vil det være viktig å la brukere registrere hver sin api nøkkel, og lagre api nøkkelen i databasen.