Høgskulen på Vestlandet

Fakultet for ingeniør- og naturvitskap

Institutt for datateknologi, elektroteknologi og realfag

Lisenskontrollsystem

Systemdokumentasjon

Versjon <3.1>

Dokumentet er basert på Systemdokumentasjon utarbeidet ved NTNU. Revisjon og tilpasninger til bruk ved IDER, DATA-INF utført av Carsten Gunnar Helgesen, Svein-Ivar Lillehaug og Per Christian Engdal. Dokumentet finnes også i engelsk utgave.

REVISJONSHISTORIE

Dato	Versjon	Beskrivelse	Forfatter
02/04/22	1.0	Påstartet arbeid	Simen og Rune
02/05/22	1.1	Laget diagrammer og utfyllende tekst	Simen og Rune
06/05/22	1.2	Utbedret diagrammer	Rune og Simen
9/05/22	2.0	Revidert dokument	Rune og Simen
15/05/22	3.0	Revidert for endelig innlevering	Simen og Rune
22/05/22	3.1	Klargjort for endelig innlevering	Simen og Rune

Innholdsfortegnelse

1 INNLEDNING **3**

2 ARKITEKTUR **4**

3 PROSJEKTSTRUKTUR **7**

3.1 Client 7

3.2 DataAccess 11

4 KLASSEDIAGRAM **13**

5 DATABASEMODELL **16**

6 SERVER-TJENESTER **17**

7 SIKKERHET **18**

8 INSTALLASJON OG KJØRING **19**

9 DOKUMENTASJON AV KILDEKODE **20**

10 KONTINUERLIG INTEGRASJON OG TESTING **21**

10.1 Kjøring av tester 21

10.2 Testdekning 21

11 REFERANSER **23**

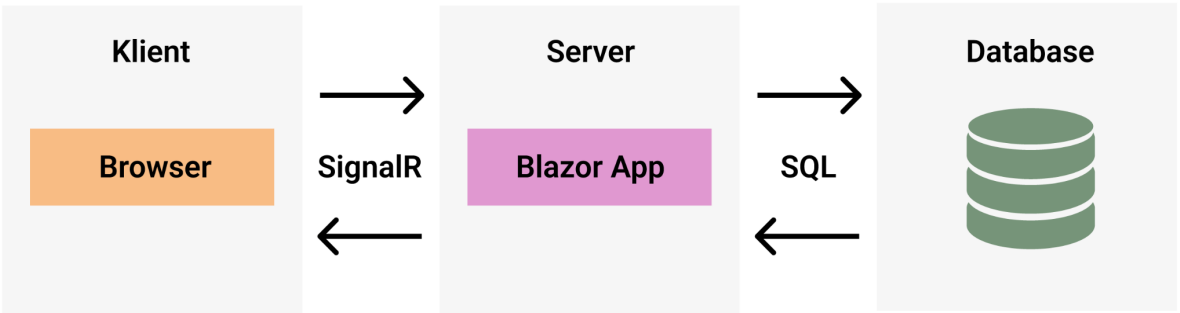
1 INNLEDNING

Systemdokumentasjon er et dokument skrevet i forbindelse med bacheloroppgaven. Hensikten med dokumentet er å gi innsikt i systemstrukturen, gjøre vedlikehold og videreutvikling av løsningen enklere. Dokumentet er primært for systemutviklere.

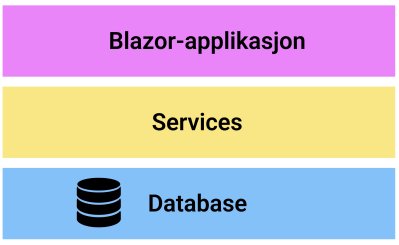
Dokumentet inneholder skisser av arkitektur, fil- og katalogstruktur, diagrammer og modeller, samt hvordan man kan installere og kjøre løsningen. Dokumentasjon av kildekoden ligger vedlagt.

2 ARKITEKTUR

Systemet består av en Blazor-applikasjon (Microsoft, 2022a) med en tilhørende database. Blazor-applikasjonen kjører server-side, som vil si at applikasjonen lastes inn på serveren, og sendes deretter til klienten som ønsker å bruke applikasjonen. Serveren er deretter ansvarlig for å holde applikasjons-state oppdatert.



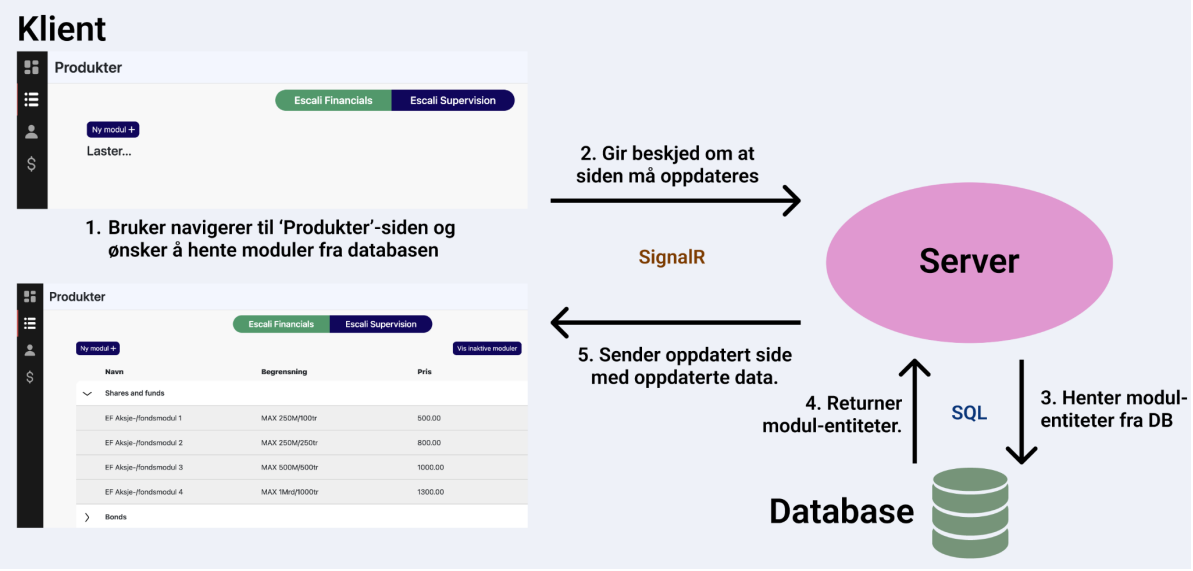
Figur 2.1:
Blazor-applikasjonen kjører på en server og holder applikasjonen oppdatert hos klienten vha. SignalR (Microsoft, 2022b). All data ligger lagret i en Azure SQL Database (Microsoft, u.å. c). Applikasjonen bruker SQL (IBM, 2021) for å hente og oppdatere denne dataen.



Figur 2.2: Kodebasens tre-lags struktur

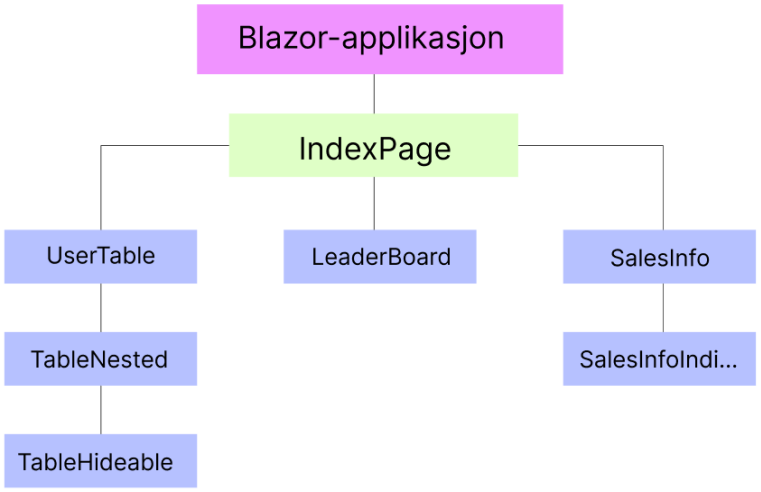
Figur 2.2 viser strukturen i kodebasen. Her inngår databasemodeller, service-klasser og en Blazor-applikasjon.

- Database - Bygget opp av entiteter lagret i tabeller i en SQL database.
- Service-klasser - C#-klasser som henter, oppretter, endrer og sletter data for en aktuell entitet.
- Blazor-applikasjon - Brukergrensesnitt. Ansvarlig for henting og fremvisning av data til bruker ved å bruke service-klassene. Kjører på serveren.



Figur 2.3: Kodebasens tre-lags struktur illustrert ved et reelt eksempel.

I figur 2.3 vises det hvordan klient, server og database kommuniserer. Klienten ønsker i dette tilfellet å se "Produkter"-siden. I det brukeren navigerer til denne siden, gir klienten beskjed om at en oppdatering av brukergrensesnitt er nødvendig. Serveren henter dermed nødvendig data fra databasen og sender tilbake oppdatert brukergrensesnitt med data til klienten.



Figur 2.4: Blazor-side, bygget opp av Razor-komponenter.

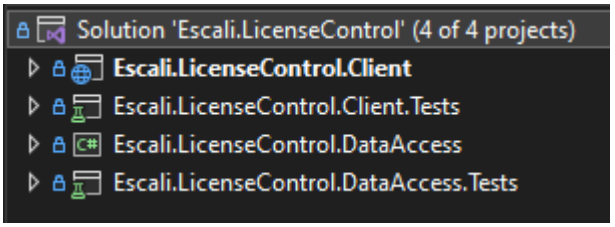
Blazor-applikasjonen er delt opp i 4 sider (Pages) som kan navigeres til vha. sidemenyen i applikasjonen. Den første er IndexPage (se figur 2.4). Alle sidene er bygget opp av mindre razor-komponenter. Figur 2.5 er en implementasjon av figur 2.4. Her vises IndexPage-komponenten, som består av tre children-komponenter: UserTable, LeaderBoard og Salesinfo.

```
@page "/"

@inject StateService _stateService
@inject UserViewService _userViewService
@inject CustomerViewService _customerViewService

<div class="grid-container">
    <UserTable />
    <LeaderBoard />
    <SalesInfo />
</div>

@code {

    protected override void OnInitialized()
    {
        _stateService.HeaderName.Value = "Dashboard";
    }

}
```

Figur 2.5: IndexPage-komponenten

3 PROSJEKTSTRUKTUR

Løsningen for bachelorprosjektet er delt inn i fire forskjellige .NET-prosjekter (Microsoft, 2022c), med to hovedprosjekt: **Client** og **DataAccess**. De to resterende er respektive testprosjekter for Client og DataAccess.

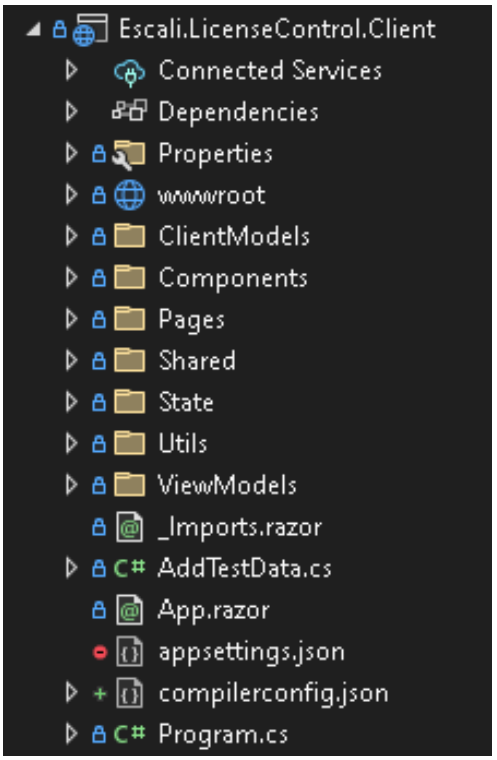


Figur 3.1: Prosjekter i løsningen

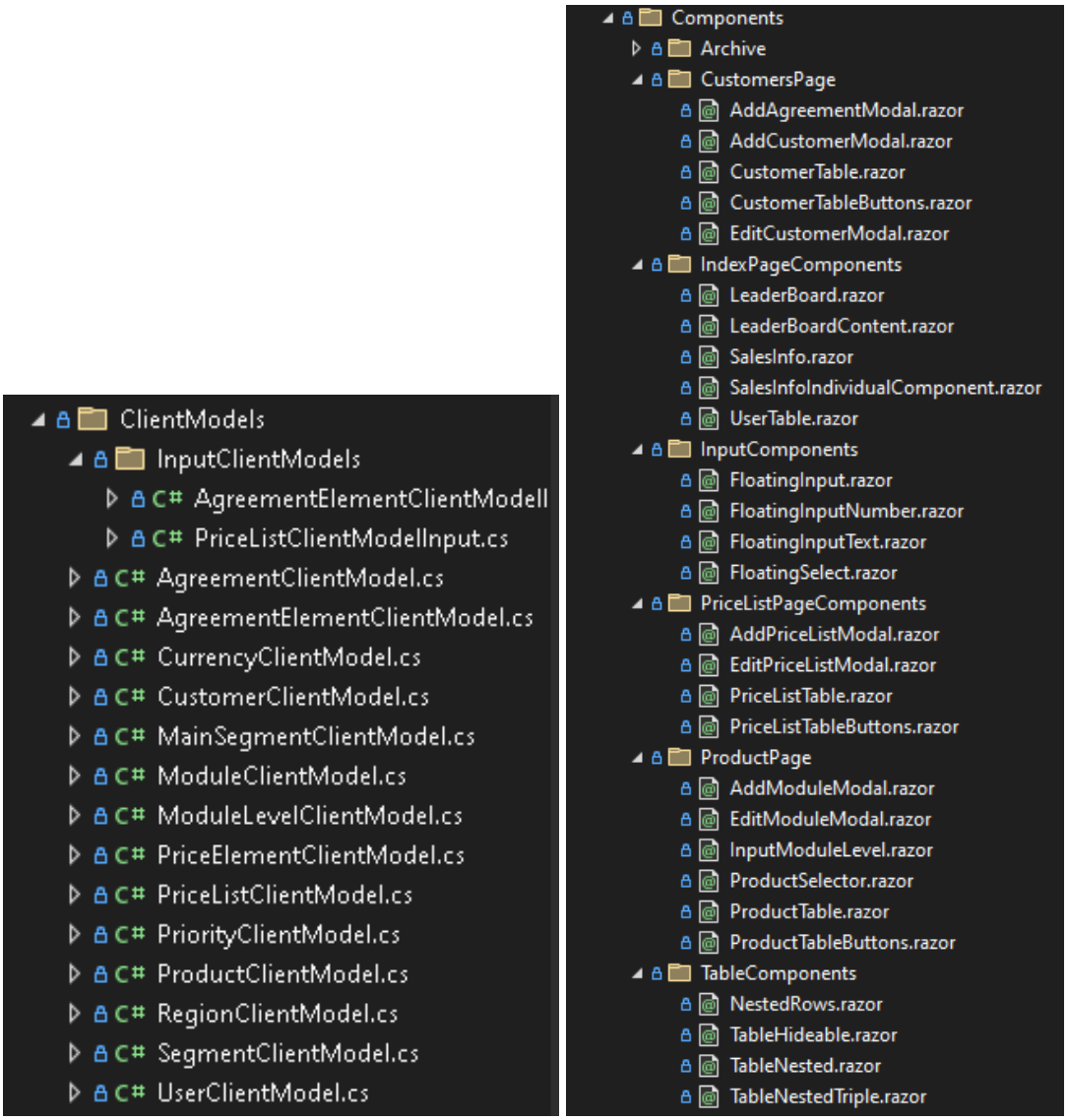
3.1 Client

I klient-prosjektet ligger kode for Blazor-applikasjonen.

Prosjektet er organisert i mappene Pages (sidene i applikasjonen), State (tilstand-lagring), Shared (felles innhold som brukes av pages), Utils (hjelpemetoder), Components (komponenter som har ansvar for å presentere data), wwwroot (CSS), ViewModels (henter og lagrer data fra database-servicene) og ClientModels (modellklasser for database-entiteter i frontend-delen av applikasjonen).
Program.cs (main) er filen som er ansvarlig for oppstart av applikasjonen.



Figur 3.2: Mappestruktur for Client-prosjekt



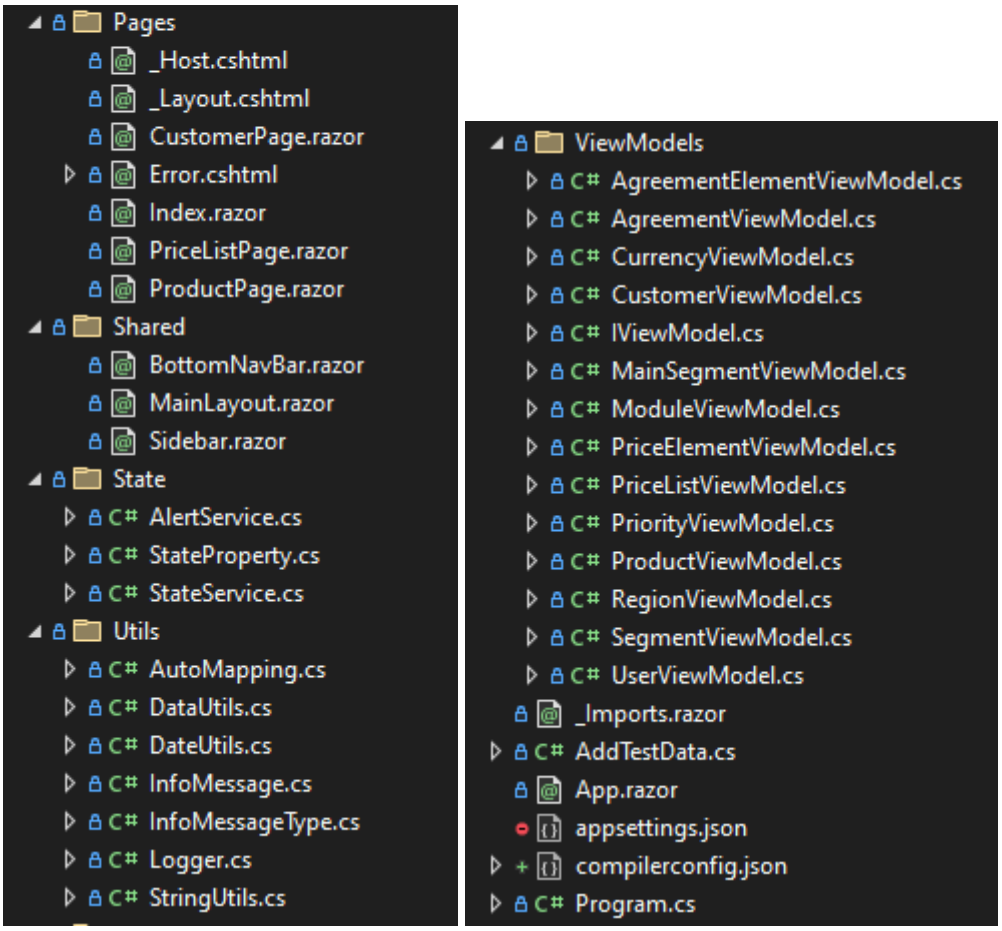
Figur 3.2:

Filer i **ClientModels**- og **Components**-mappen.

ClientModels består av C#-modellklasser som brukes for å representere database-entiteter i Blazor-applikasjonen.

Components inneholder komponenter som er ansvarlig for å presentere data. Komponentene er delt inn mapper basert på hvilken side/page de brukes på.

TableComponents og **InputComponents** er felles-komponenter som brukes på samtlige sider/pages.



Figur 3.3:

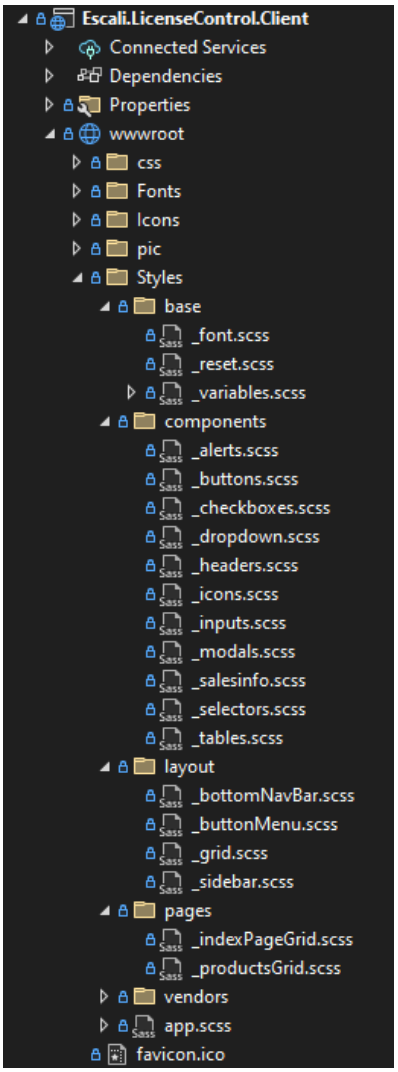
Pages inneholder de ulike side-komponentene: Index, Product, Customer og PriceList

Shared er felles kode som brukes på alle sider/pages. Her ligger blant annet kode for sidemenyen i applikasjonen.

State inneholder C#-klasser som er ansvarlig for tilstandslagring i applikasjonen.

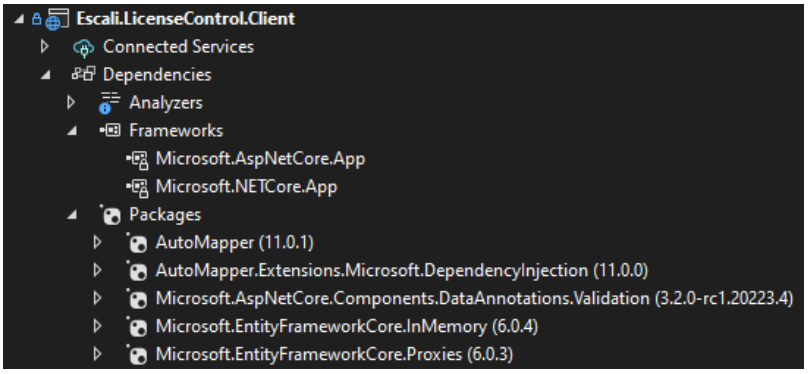
Utils inneholder hjelpemetoder og hjelpeklasser.

ViewModels. Det eksisterer én ViewModel per database-entitet. Disse klassene er ansvarlig for å hente, oppdatere og lagre data fra databasen for sin respektive entitet.



Figur 3.4:

wwwroot inneholder SCSS-filer og diverse andre filer som ikoner, bilder og fonter. SCSS-filene kompileres til én CSS-fil, som videre brukes av applikasjonen for å style HTML-elementer.



Figur 3.5:

Figuren viser avhengigheter for i klient-prosjektet: Rammeverk og installerte pakker. Disse forklares nærmere i seksjon 8.

3.2 DataAccess

DataAccess-prosjektet er ansvarlig for definisjon av database og inneholder i tillegg metoder for å hente og oppdatere data i databasen.

Prosjektets er organisert i mappene: Data (database-definisjoner), Utils (hjelpemetoder), Models (entiteter i database) og Services (service-klasser for SQL-operasjoner).

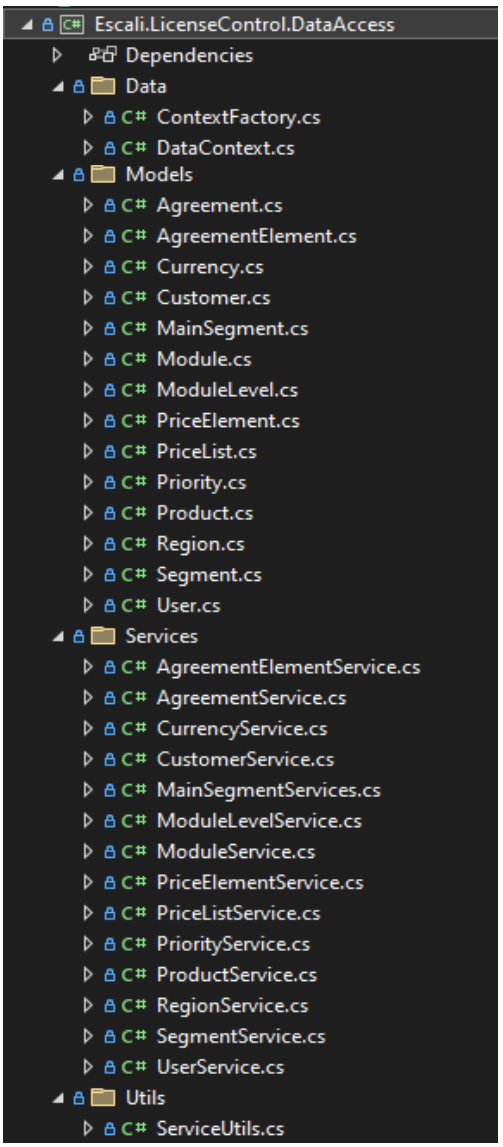
Figur 3.6 viser prosjektstrukturen.

Data inneholder filer som definerer databasen.

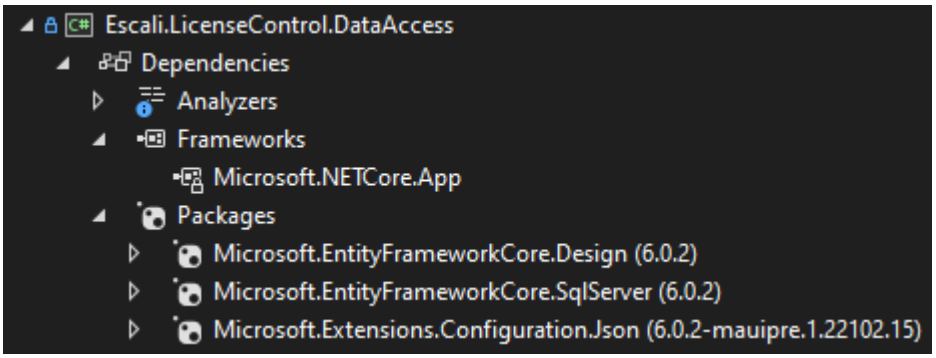
Models består av entitets-klasser for databasen.

Services inneholder én service-klasse for her database-entitet. Service-klassene inneholder metoder for å hente, opprette, oppdatere og slette data i databasen for den respektive entiteten.

Utils er hjelpemetoder.



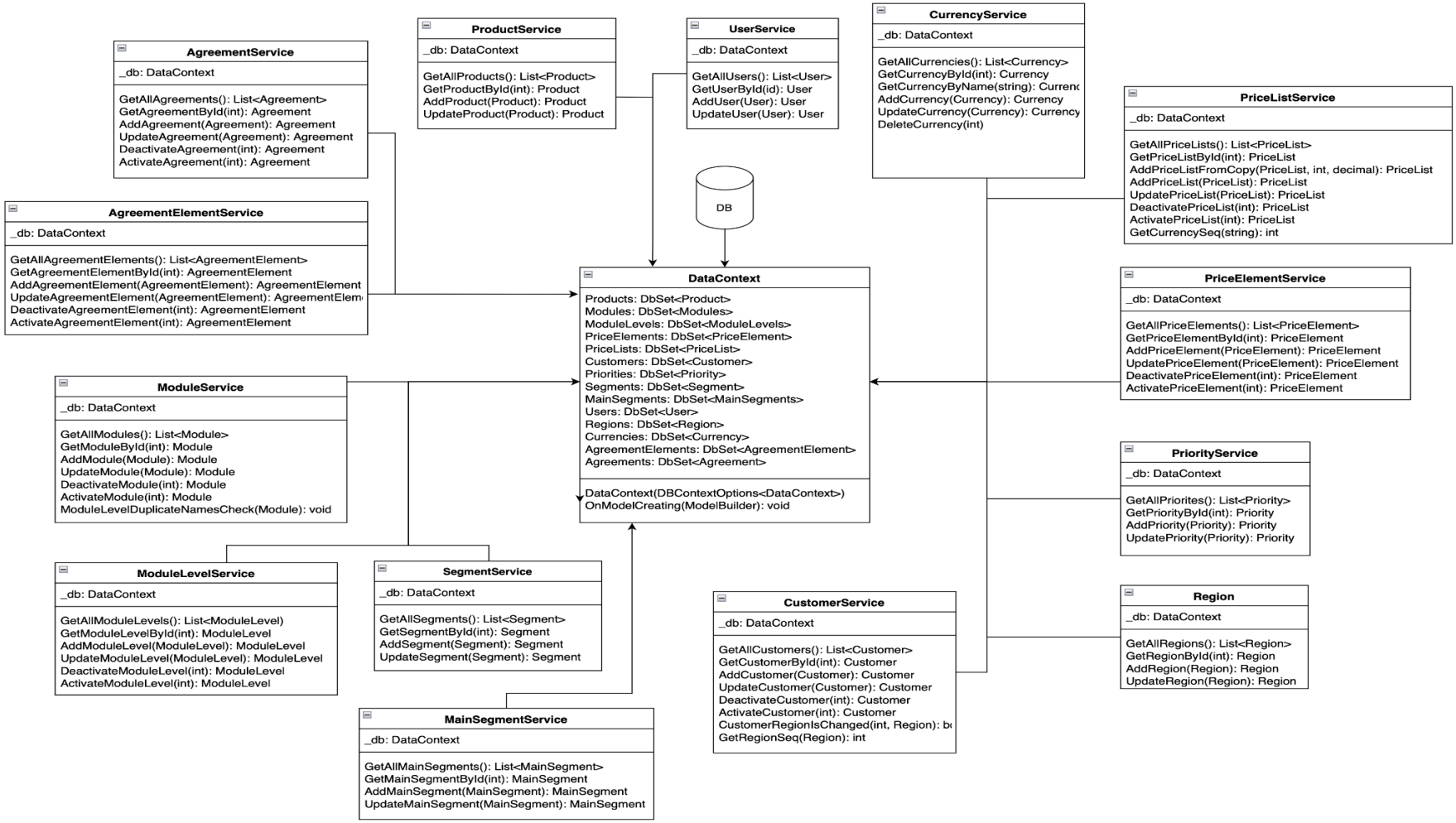
Figur 3.6: Mappestruktur og filer i DataAccess-prosjekt



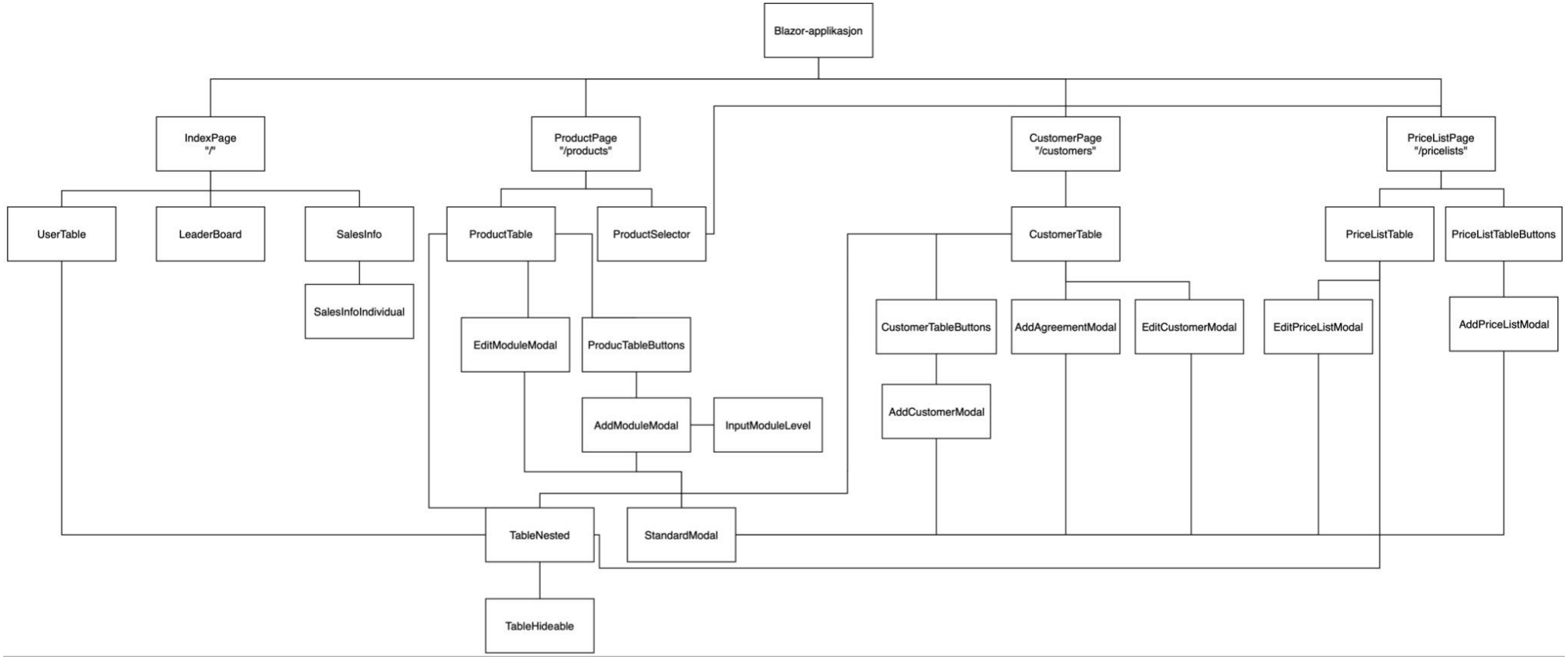
Figur 3.7:

Figuren viser avhengigheter for DataAccess-prosjektet: Rammeverk og installerte pakker. Disse forklares nærmere i seksjon 8.

4 KLASSEDIAGRAM

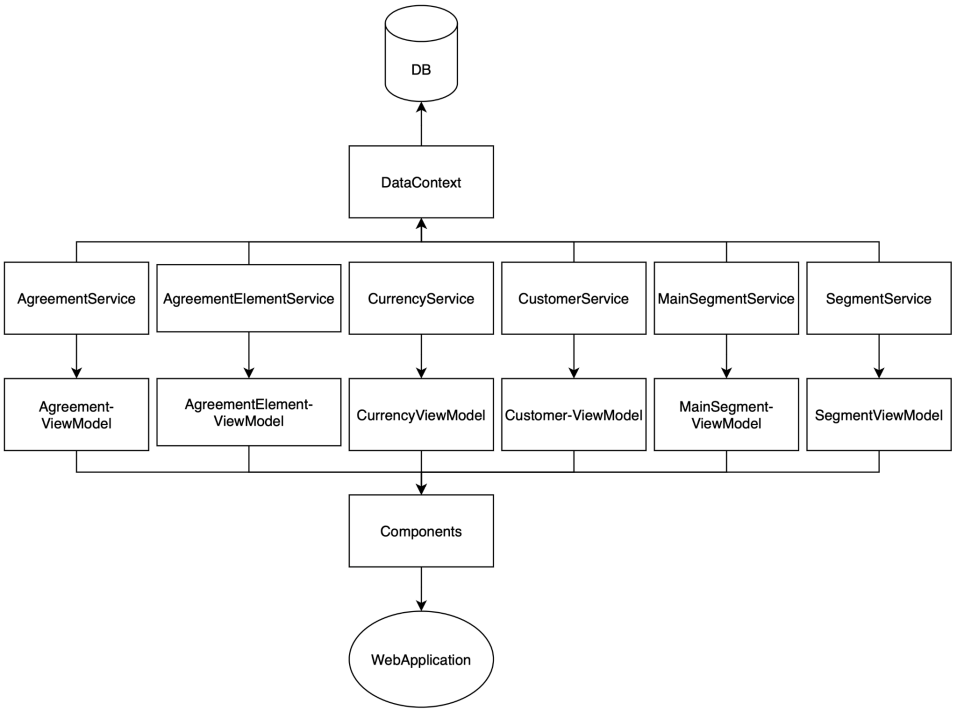


Figur 4.1: Klassediagram for DataAccess-prosjekt. DB er database. DataContext brukes for å hente og legge til data i databasen. Servicene har metoder for å hente, legge til og modifisere entiteter i entitets-tabeller, gjennom DataContext til DB.

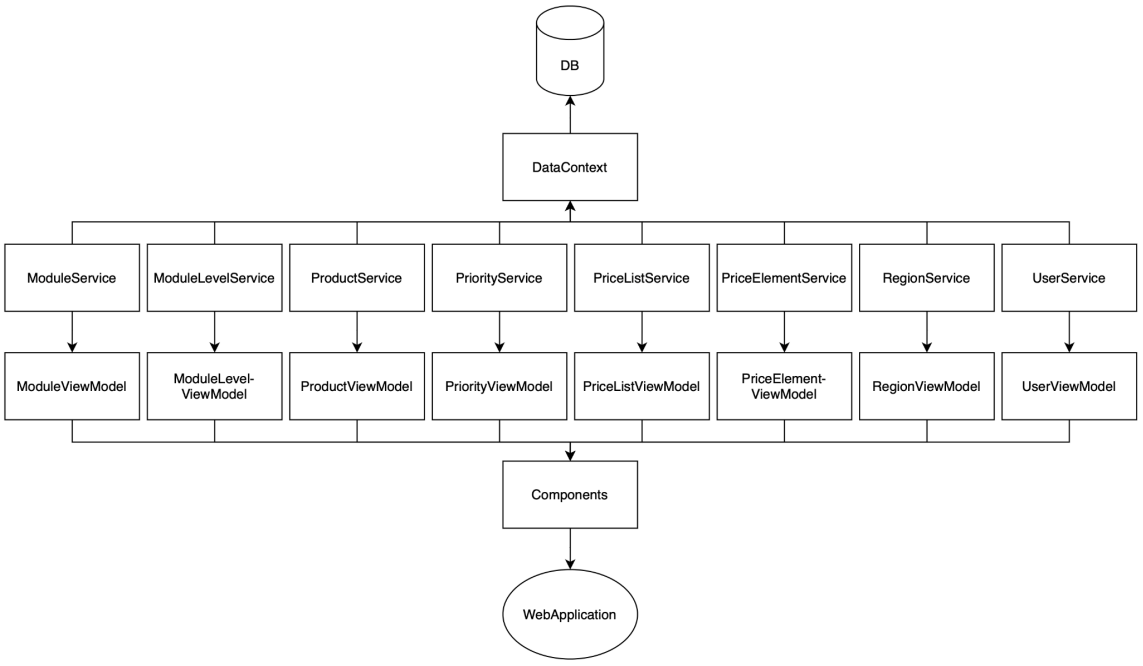


Figur 4.2: Klassediagram for Blazor-prosjekt. Blazor-applikasjonen består av fire hovedsider. Disse fire hovedsidene er bygget opp av flere komponenter. Komponentene er igjen bygget opp av andre komponenter. Gruppen har laget gjenbrukbare komponenter. StandardModal og TableNested blir gjenbrukt av komponenter fra alle sidene.

Figur 4.3 og 4.4 viser hvordan Client- og DataAccess-prosjekt er koblet sammen. DataAccess-prosjektet er lagt til som en avhengighet i Client. ViewModel-klassene i Client bruker Service-klassene i DataAccess for å hente data fra databasen. Disse ViewModel-klassene brukes igjen av de ulike komponentene i Blazor-applikasjonen for å vise data. For å få bedre oversikt, har gruppen valgt å utelate ViewModels fra figur 4.2. Components i figur 4.3 og 4.4 tilsvarer figur 4.2.

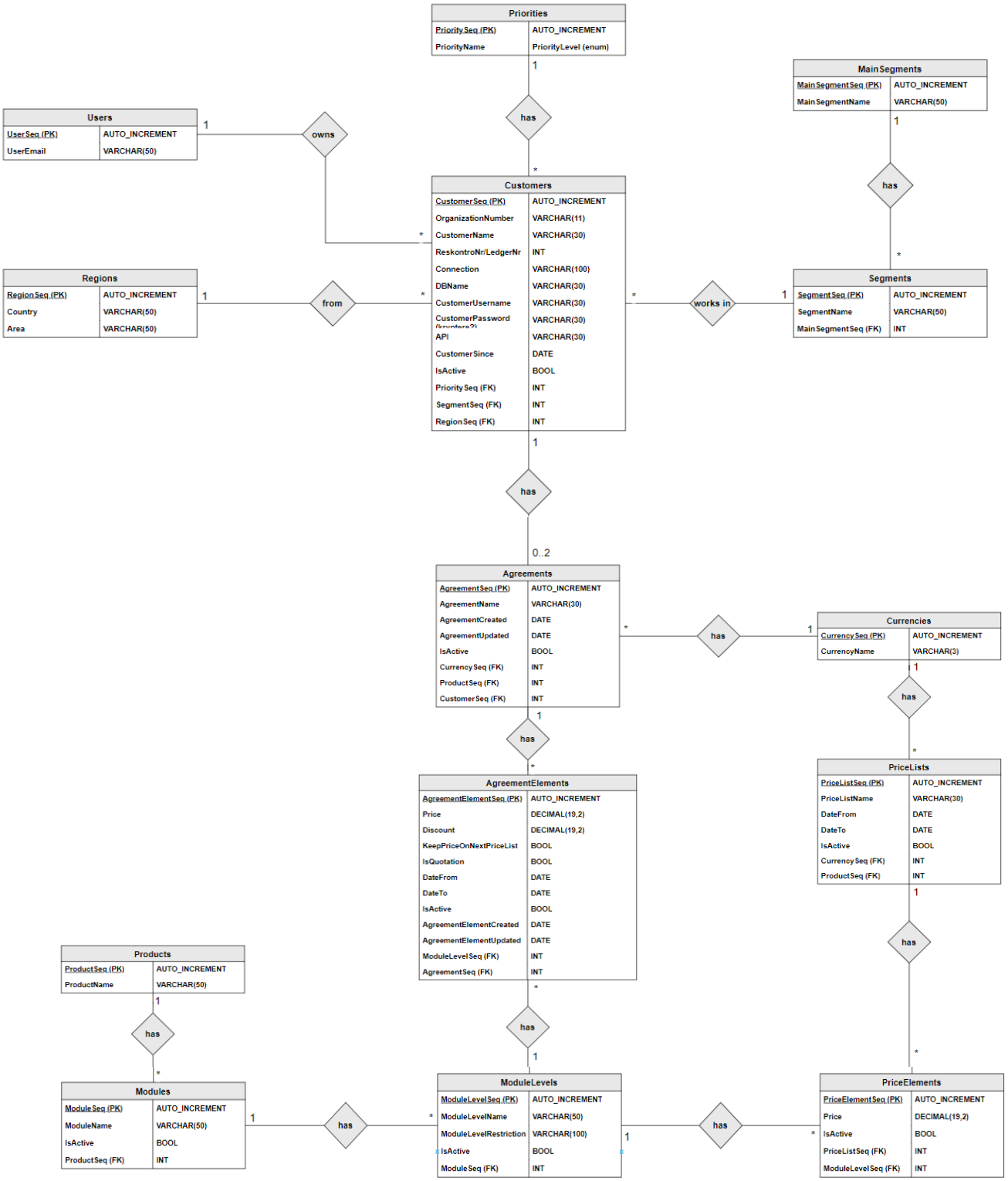


Figur 4.3: Del 1 av hvordan Client og DataAccess er koblet sammen.



Figur 4.4: Del 2 av koblingen mellom Client og DataAccess.

5 DATABASEMODELL



Figur 5.1: Databasemodellen viser entiteter m/attributter og relasjoner mellom disse. Den er utviklet av gruppen i samråd med Escali. Entitetene er generert ut i fra klassene som ligger i Models-mappen i figur 3.6.

6 SERVER-TJENESTER

Ikke relevant for prosjektet.

7 SIKKERHET

I løsningen gruppen har levert, har det ikke vært fokus på innlogging av brukere, og dermed ikke blitt implementert autentisering. Det har vært en alternativ løsning for innloggingen i webapplikasjonen. Dersom gruppen hadde implementert innlogging, ville dette blitt gjort gjennom Microsoft Identity-plattformen (Microsoft, u.å. a), via en Microsoft-konto. Dette vil implementeres i en evt. fremtidig versjon av løsningen.

Entity Framework Core (Microsoft, 2021) står bak mye av sikkerheten i webapplikasjonen. Det gruppen har lagt til for å høyne sikkerheten mot for eksempel SQL-injection, er å benytte teknologien LINQ (Language-Integrated Query) (Microsoft, u.å. d). SQL injection er et angrep bestående av injeksjon av SQL spørring gjennom input data fra klienten i en application (OWASP, u.å. a).

8 INSTALLASJON OG KJØRING

Programvarebibliotek som er brukt i løsningen:

- **ASP.NET Core:** ASP.NET er et webutviklings-rammeverk for å bygge webapplikasjoner på .NET plattformen. Core er en open-source versjon av ASP.NET som kjører på macOS, Linux og Windows (Microsoft, u.å. b).
- **Microsoft Entity Framework Core:** er en moderne database-kartlegger for .NET. Fungerer som en objekt-relasjons-kartlegger som lar .NET-utviklere jobbe med databaser ved bruk av .NET objekter (Microsoft, 2021).
- **Microsoft.NET.Test.SDK:** brukes for å bygge .NET test prosjekter.
- **xUnit:** et open-source enhetstesting verktøy for .NET rammeverk (xUnit, u.å.).
- **bUnit:** er et testbibliotek for Blazor komponenter. bUnit bygger oppå enhetstesting biblioteket xUnit (bUnit, u.å.).
- **AutoMapper:** er et bibliotek for å konvertere objekttyper (AutoMapper, u.å.). I vårt tilfelle Model-objekter (DataAccess) til ClientModel-objekter (Client).

Løsningen har ikke blitt deployet. Det vil si at det kjøring vil skje gjennom brukers terminal. For at brukeren skal kunne kjøre løsningen trengs .NET Core. Installasjon for .NET på Windows kan gjøres ved å følge lenken på Install .Net on Windows. .NET Core kan også installeres på Mac eller Linux om det er aktuelt.

Det er blitt laget en database for selve prosjektet, men for å unngå unødvendig nedlasting av programvare for å kunne kjøre applikasjonen, har det blitt laget en “in-memory” database hvor det blir lagt inn eksempeldata.

Stegene blir da:

1. Åpne terminalen og naviger til root-folderen Escali-LisenceControl
2. Naviger til klientprosjektet ved *“cd ./Escali.LisenceControl.Client”*
3. Deretter kan man kjøre applikasjonen ved *“dotnet run”*.

9 DOKUMENTASJON AV KILDEKODE

Det er skrevet dokumentasjon for koden i DataAccess-prosjektet.

Dokumentasjonen er blitt generert av Doxygen. Doxygen genererer dokumentasjon fra kommentert C++ kilder, men har støtte for flere programmeringsspråk, som i gruppens tilfelle C#. Doxygen genererer en nettside med dokumentasjon fra kommentarer i C#-filene (Doxygen, 2022).

Vedlagt i den innleverte zip-filen er en mappe kalt “Dokumentasjon”. Åpne index.html for å vise dokumentasjonen.

10 KONTINUERLIG INTEGRASJON OG TESTING

Kontinuerlig integrasjon er ikke benyttet i prosjektet.

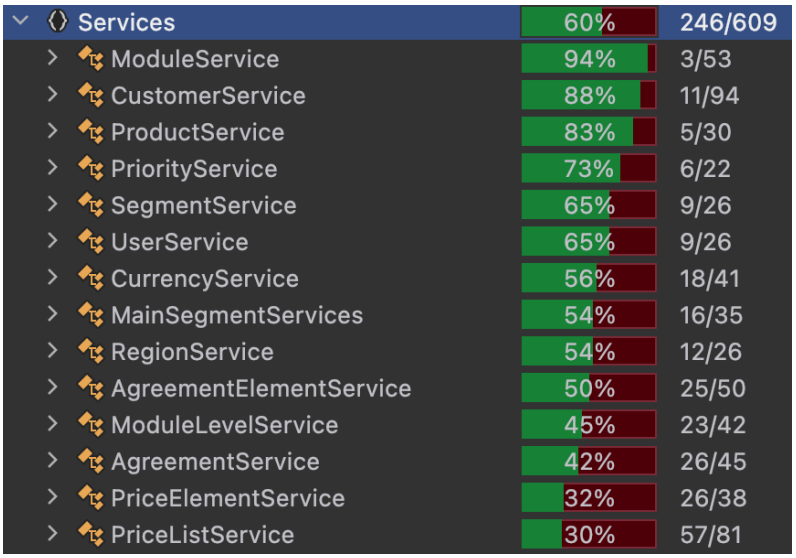
10.1 Kjøring av tester

Det har blitt gjennomført tester for både Client og DataAccess.

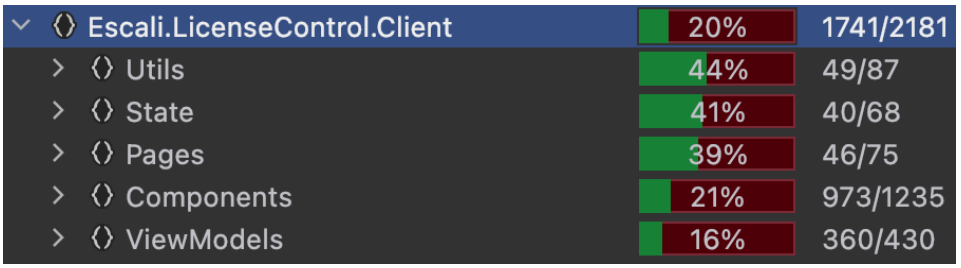
1. Åpne terminalen og naviger til root-folderen Escali-LisenceControl
2. Naviger til et testprosjekt
    a. *"cd ./Escali.LisenceControl.Client.Tests"* for Client
    b. *"cd ./Escali.LisenceControl.DataAccess.Tests"* for DataAccess
3. Testene kjøres ved å bruke kommandoen: *"dotnet test"*

10.2 Testdekning

Gruppen skrev en del tester for Service-klassene da disse var implementert. Etterhvert gikk fokuset over på å utvikle webapplikasjonen, og testing kom i andre rekke. Dette har resultert i testdekning på 60% av Service-klassene i DataAccess (figur 10.1) og 20% på Klienten (figur 10.2 neste side). På grunn av tidsbegrensningen, og at gruppen bare består av to medlemmer, har det ikke vært oppnåelig å skrive flere tester. I tillegg har gruppen og prosjekteier diskutert seg frem til at fokuset under utvikling skulle være å implementere så mye funksjonalitet som mulig, fremfor å implementere mindre funksjonalitet med stor testdekning.



Figur 10.1: Testdekning for DataAccess (services).



Figur 10.2: Testdekning for Client

11 REFERANSER

AutoMapper (u.å.) *AutoMapper.* Tilgjengelig fra: https://automapper.org/ (Hentet: 04.05.22)

bUnit (u.å.) *bUnit: a testing library for Blazor components.* Tilgjengelig fra: https://bunit.dev/ (Hentet: 04.05.22)

Doxygen (2022) *Doxygen.* Tilgjengelig fra: https://www.doxygen.nl/index.html (Hentet: 04.05.22)

IBM (2021) *What is SQL?* Tilgjengelig fra: https://www.ibm.com/docs/en/developer-for-zos/14.0.0?topic=support-what-is-sql (Hentet: 09.03.22)

Microsoft (u.å. a) *Add sign in with Microsoft.* Tilgjengelig fra: https://developer.microsoft.com/en-us/identity/add-sign-in-with-microsoft (Hentet: 04.05.22)

Microsoft (u.å. b) *What is ASP.NET Core?* Tilgjengelig fra: https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet-core (Hentet: 04.05.22)

Microsoft (u.å. c) *Azure SQL Database.* Tilgjengelig fra: https://azure.microsoft.com/nb-no/products/azure-sql/database/ (Hentet 21.05.22)

Microsoft (u.å. d) *SQL-Injections Attack*. Tilgjengelig fra: https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/sql/linq/frequently-asked-questions?redirectedfrom=MSDN#how-is-linq-to-sql-protected-from-sql-injection-attacks- (Hentet 22.05.22)

Microsoft (2021) *Entity Framework Core.* Tilgjengelig fra: https://docs.microsoft.com/en-us/ef/core/ (Hentet: 04.05.22)

Microsoft (2022a) *ASP.NET Core Blazor hosting models*. Tilgjengelig fra: https://docs.microsoft.com/en-us/aspnet/core/blazor/hosting-models?view=aspnetcore-6.0 (Hentet 22.05.22)

Microsoft (2022b) *Overview of ASP.NET Core SignalR*. Tilgjengelig fra: https://docs.microsoft.com/en-us/aspnet/core/signalr/introduction?view=aspnetcore-6.0 (Hentet 19.04.22)

Microsoft (2022c) *What is .NET? Introduction and overview*. Tilgjengelig fra: https://docs.microsoft.com/en-us/dotnet/core/introduction (Hentet 04.03.22)

OWASP (u.å. a) *Cross Site Scripting (XSS).* Tilgjengelig fra: https://owasp.org/www-community/attacks/xss/ (Hentet: 04.05.22)

xUnit (u.å.) *About xUnit.net.* Tilgjengelig fra: https://xunit.net/ (Hentet: 04.05.22)