Systemdokumentation

Brugerbeskrivelse:

Systemet er udviklet til Middelby Reolmarked's stab. Det vil sige, at systemet på nuværende tidspunkt betjenes af en medarbejder og 2 ejere, men systemet er ikke begrænset af dette og stabens størrelse.

Som bruger kan man trykke sig ind i systemet og nemt navigere sig rundt i menuen. I menuen kan man vælge mellem 6 forskellige punkter:

1. *Lejere:* Brugere kan administrere de nuværende lejere, ved at slette eller redigere i dem. Derudover kan man også oprette nye lejere.

2. *Reoler:* Ligesom med lejerne kan man også administrere reolerne ved at redigere eller slette i dem. Hvis ejerne vælger at udvide lokalet, kan man også oprette nye reoler.

3. *Lejeaftaler:* Ligesom de 2 forrige, kan man administrere lejernes lejeaftaler, ved at slette og redigere i dem eller oprette en ny.

4. *Produkter:* I produkter kan man, ligesom i de 3 ovenstående punkter, også tilføje, redigere eller slette produkter. Produkterne tilhører en lejer, og man kan derfor også vælge en lejer til højre i programmet, hvor man kan se hvilke produkter der tilhører lejeren.

5. *Salg:* I denne kategori vælger man de eksisterende produkter og tilføjer til kurven. Når alle produkterne er tilføjet, kan man trykke på gennemfør salg.

6. *Regnskab:* Til sidst på måneden kan den regnskabs-ansvarlige klikke ind på punktet "Regnskab" og derfra udregne regnskabet for en bestemt periode.

Som systemet er opsat i denne version, kan alle brugere (både medarbejdere og ejere), have adgang til alle funktioner i systemet. Det vil sige, at en medarbejder også ville kunne gå ind og administrere følsomme informationer som fx regnskab.

På sigt kunne det være implementering af login-system, hvor man kan give de forskellige brugere rettigheder, så en medarbejder kun har adgang til relevante daglige opgaver.

Formål med systemet

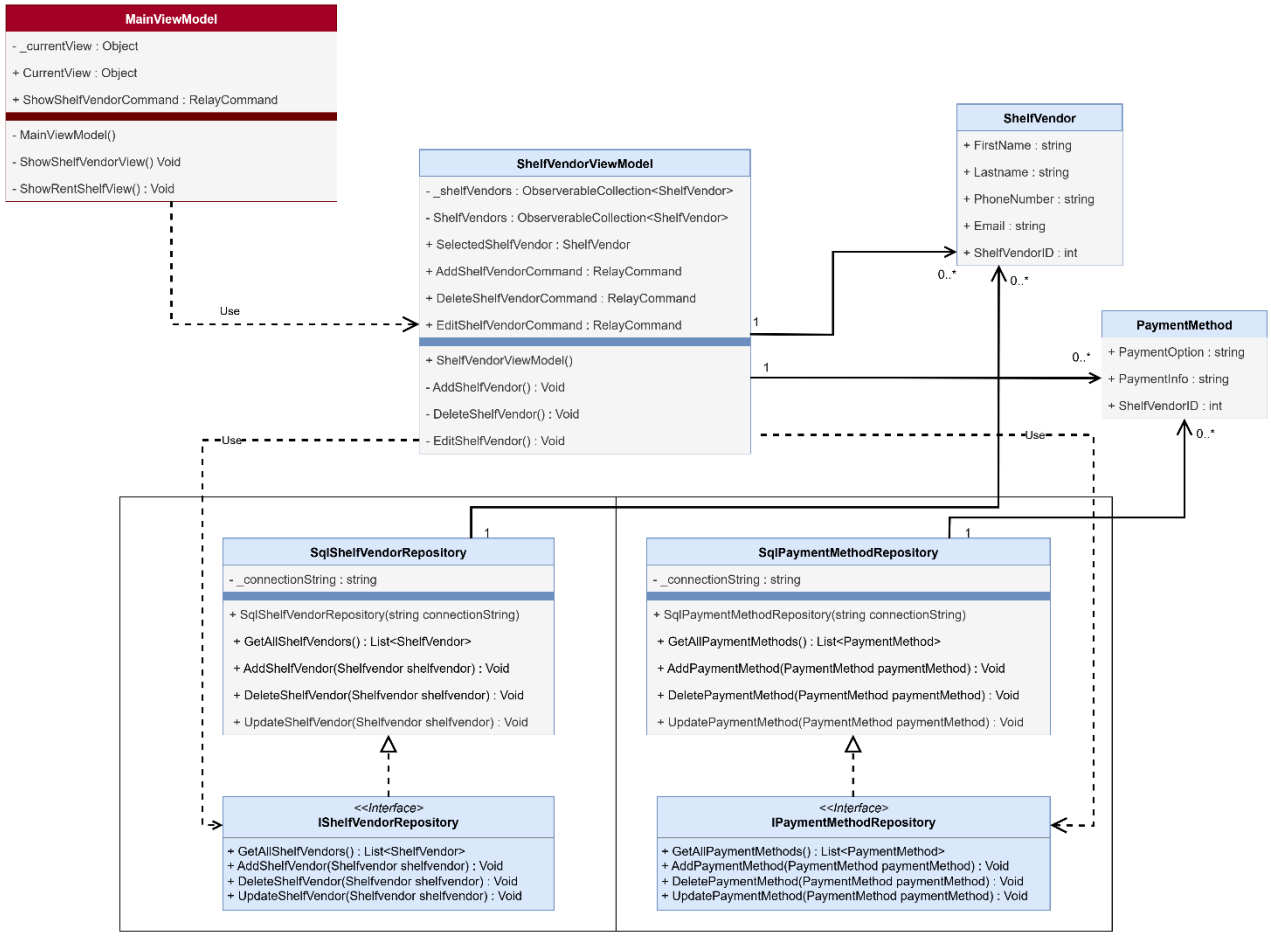
Systemets formål har været at lavet et simpelt program som hjælper Middelby Reolmarked at gå fra analoge arbejdsgange, og i stedet skifte til en simpel og digital løsning der hjælper med at reducere arbejdsbyrden. Det er både ved oprettelse af reollejere, lejeaftaler og reoler. Systemet skal samle dataen og formidle det på en simpel og læsbar måde, der giver product owner det som de ønsker, en simpel og smart løsning.

Systemarkitektur

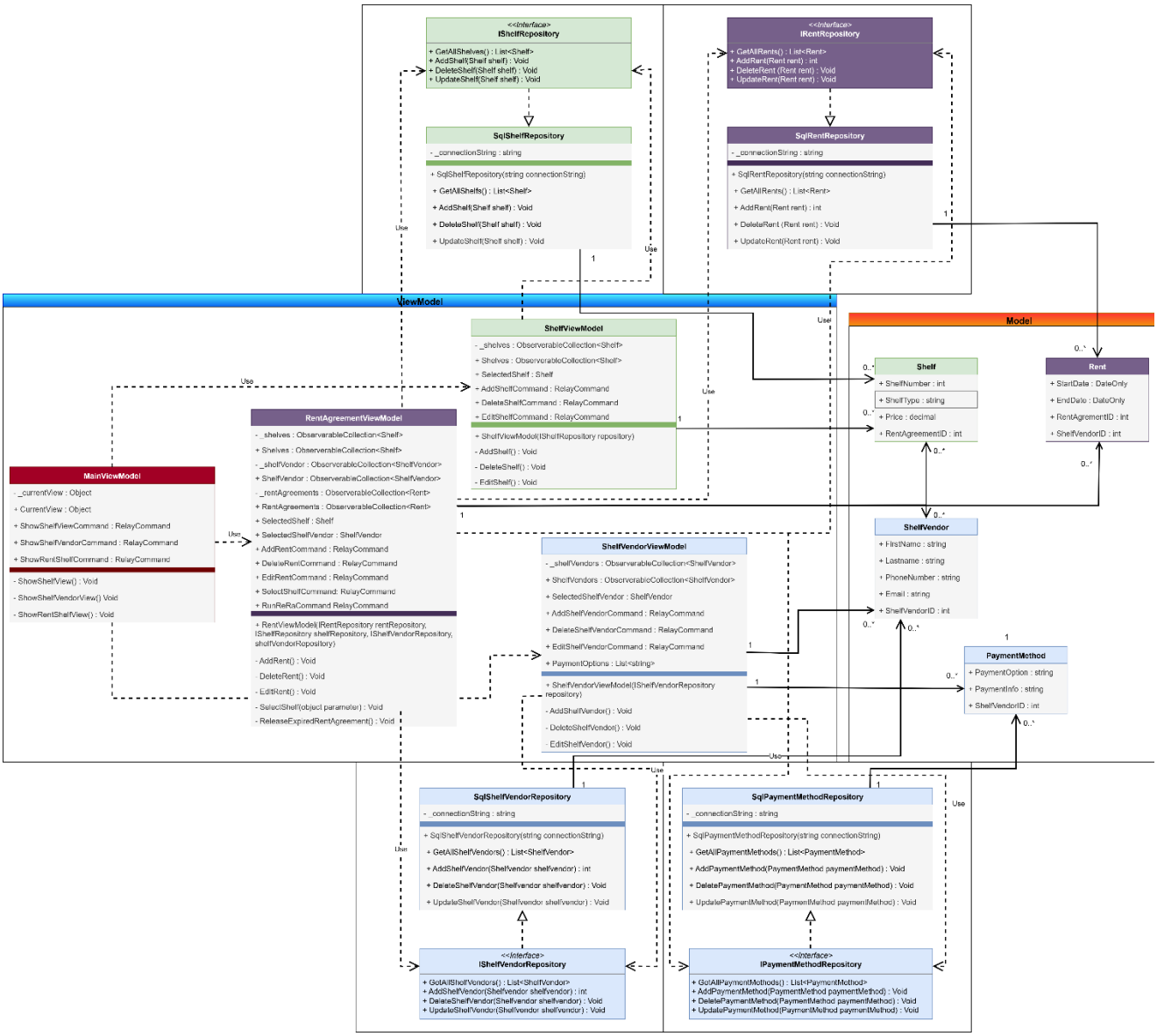
Systemet er lavet i MVVM arkitektur, ved at lagdele de forskellige dele af programmet i modules eller hvad man kan kalde legoklodser, gør det at man nemt kan udvikle på programmet samt lave om på det, på et senere tidspunkt. Man kan blot koble de her ting sammen nemt, og det gør udviklingen i grupper eller hold nemmere ved at man kan bruge Scrum f.eks. til at arbejde på forskellige klodser af programmet, og samle det efter et sprint.

Det gør også navigationen af koden nemmere, da man ved hvilket lag af koden der gør hvad i systemet, og derfor også nemmere at bugfixe hvis problemer findes.

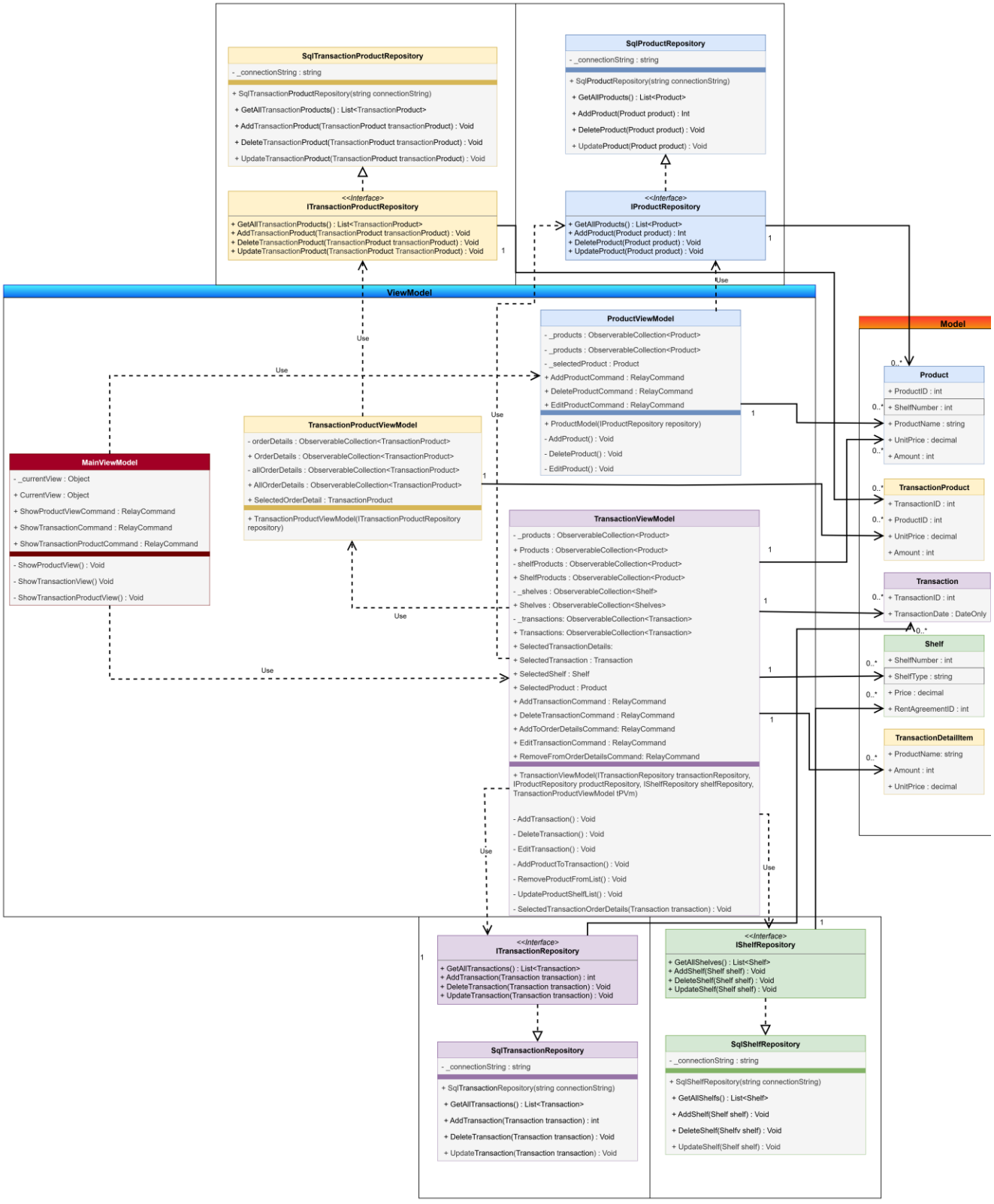
Use case 1 DCD: (Zoom for bedre kvalitet)



Use case 2 DCD: (Zoom for at se bedre)



Use case 3 DCD: (Zoom for at se bedre)



Systemet er opbygget ved at vi har én central viewmodel, som bruges som controller til at styre resten af programmet. Det er igennem den, at vores views og dermed datacontext til at bruge forskellige viewmodels, styres

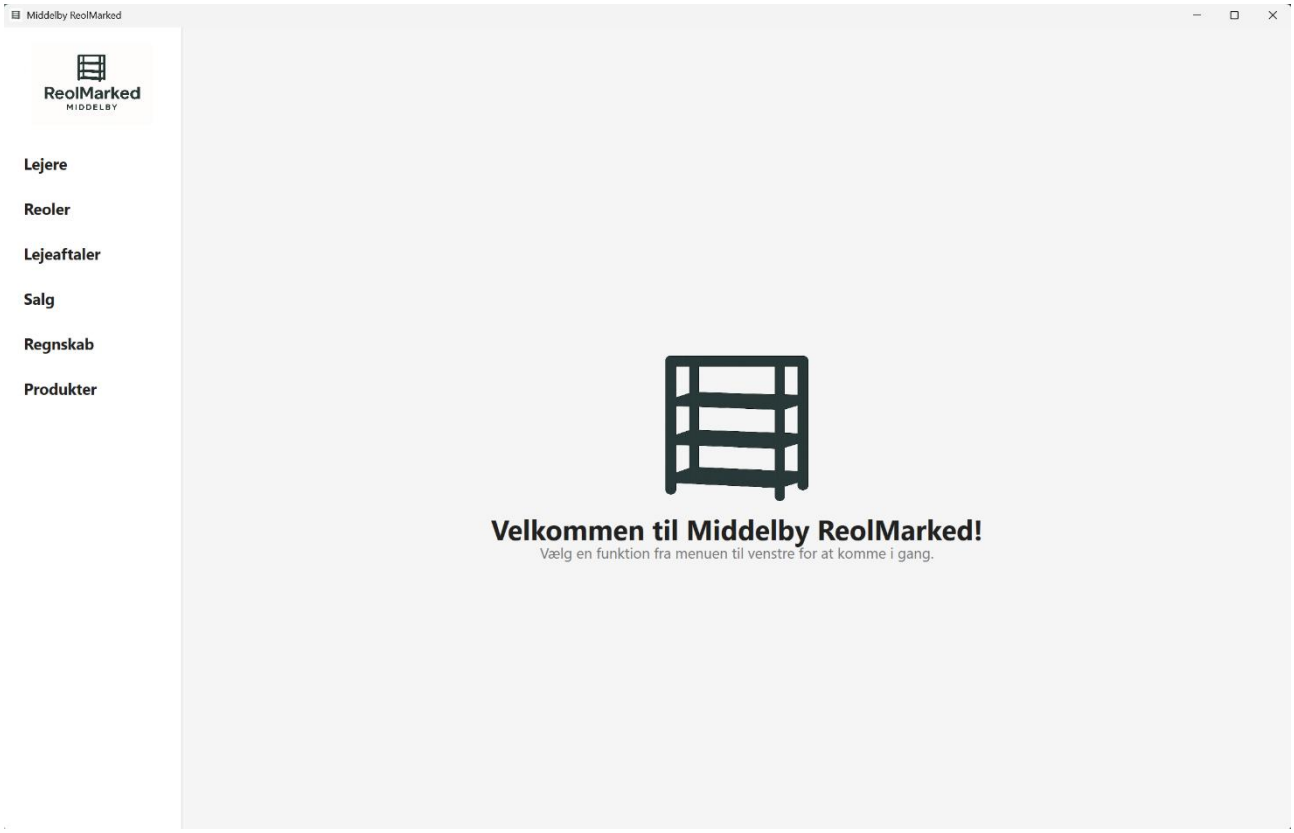
Vi har ikke modelleret klasser som RelayCommand og ViewModelBase, hvor RelayCommand er vores klasse vi bruger til at køre metoder med, som bindes i view. Alle vores viewmodels bruger den på nær TransactionProductViewModel. RelayCommand klassen implementerer ICommand.

Alle vores ViewModels arver fra ViewModelBase, som implementerer INotifyPropertyChanged, som vi derefter bruger på alle vores lister og properties.

Vi fik ikke sat Views med i DCD, da vi var usikre på hvordan det skulle modelleres.

Database

Funktionalitet



Systemet fungerer ved at vi har en hovedmenu i venstre side, som styrer navigationen af programmet. I højre side, som fylder 90% af content, vises netop ”content”. Dvs de forskellige tabs vises her.

Som funktionalitet har vi følgende ting implementeret:

Lejere: Du kan slette/oprette/redigere lejere hvor der tilknyttes betalingsinformation (Om de have bankoverførsel eller mobilepay overførsel)

Reoler: Du kan slette/oprette/redigere reoler. Her findes kendte begrænsninger som uddybes til sidst. Når en reol oprettes, står den default som ledig og er et objekt som genbruges medmindre den ønskes slettet.

Lejeaftaler: Her kan man oprette/redigere lejeaftaler. Man kommer ind på en side først som viser listen over alle lejeaftaler i databasen, hvor man kan klikke videre og komme ind på et vindue hvor et mockup af butikken vises med ledige og optagede reoler. Man klikker så på en af dem, og opretter derefter en lejeaftale.
Derudover findes en knap som kører en metode, som frigiver alle reoler som har lejeaftale, der er udløbet. Dette er en "manuel" proces, som kører alle reoler igennem når knappen klikkes. Hvis kunden ønsker det (reolmarked), kan det automatiseres til også at køre under program opstart.

Salg: Du kan slette/oprette/redigere salg. Det fungerer ved at du vælger produkter, som hører under deres respektive reoler, som du vælger først i en liste, og så vises produkter. Man tilføjer så et produkt eller fjerner et, hvor den så enten fjerner eller tilføjer 1 på deres attribut.
Når man er tilfreds med sin indkøbskurv, så klikker man på gennemfør salg og der oprettes en transaktion, og en _product for hvert produkt der hører til transaktionen.
Som tilføjelse til dette, har vi implementeret at hvis man vælger at slette en transaktion, så cascader den alle transaktion_produkter. Dette bliver brugeren af systemet også advaret mod.

Regnskab: Her har vi tilføjet 2 stored procedures til at starte med, som kan tilføjes til hvis kunden ønsker det.
Til at starte med, har vi vores regnskab procedure, som hjælper kunden med at lave regnskab på alle aktive kunder, i stedet for at lave det manuelt.
Det fungerer ved at der bliver valgt to tidsintervaller, som henter alle salg samt deres udgifter, og viser dem inden for perioden. Der vises så attributter som navn, lejerid, salg, salg efter kommission, udgifter og til udbetaling.
Den anden procedure, giver mulighed for at tjekke en kundes totale salg i hele deres historik som kunde. Dette kan også overwrites med "0", hvor man i stedet for så ser alle kunder.

Produkter: Under produkter findes de sædvanlige CRUD-operationer til at administrere produkter. Funktionaliteten er afhængig af, at medarbejderen først vælger en specifik lejer, hvorefter systemet indlæser dennes tilknyttede reoler og produkter fra databasen. Dette giver lejeren (eller medarbejderen) mulighed for at oprette, redigere og slette produkter for den valgte lejer. Hver handling sender en tilsvarende INSERT, UPDATE eller DELETE kommando til Product-tabellen for at opdatere databasen med de nye informationer.

Teknologier

- **Framework**: Applikationen er bygget på .NET 9.0 – i skrivende stund den seneste version af Microsofts udviklingsplatform.
- **UI**: Brugergrænsefladen er udviklet med WPF (Windows Presentation Foundation), et UI-framework til at skabe desktop-applikationer med et rigt brugerinterface.
- **Sprog**: Al kode er skrevet i C#.
- **Database**: Systemet anvender Microsoft SQL Server til datalagring. Kommunikation med databasen sker via Microsoft.Data.SqlClient-biblioteket.
- **Arkitektur**:
    - MVVM: Applikationen følger MVVM-mønsteret for at adskille logik, data og præsentation. Dette opnås ved at have separate Views, ViewModels og Models.

    - Repository Pattern: Adgang til databasen er abstraheret væk via et repository-lag. Hver entitet har sit eget repository-interface (f.eks. IProductRepository) og en SQL-implementation (f.eks. SqlProductRepository).
    - Dependency Injection (DI): Afhængigheder mellem de forskellige lag (f.eks. at en ViewModel har brug for et Repository) håndteres centralt via Microsoft.Extensions.DependencyInjection-biblioteket. Dette er konfigureret i DIContainer.cs-filen.

- **Konfiguration**: Databasetilslutningsstrengen og andre konfigurationsindstillinger administreres via en appsettings.json-fil, som indlæses ved hjælp af Microsoft.Extensions.Configuration.Json.

Unit test

Projektet indeholder et dedikeret testprojekt, **ReolMarkedWPF.Tests**.

- **Test framework**: Testene er skrevet ved hjælp af MSTest, Microsofts officielle test-framework til .NET.
- **Mocking**: For at isolere de enkelte ViewModels under test anvendes Moq-biblioteket. Dette gør det muligt at simulere afhængigheder, såsom repositories, og dermed kun teste logikken i den specifikke ViewModel uden at være afhængig af en live database.
- **Fokusområder**: Unit-testene fokuserer primært på at validere logikken i ViewModels, herunder:
    - **Kommandoers CanExecute-status**: Tester, om knapper i brugergrænsefladen aktiveres og deaktiveres korrekt baseret på applikationens tilstand (f.eks. om en "Tilføj"-knap er deaktiveret, hvis påkrævede felter er tomme).
    - **Forretningslogikken**: Tjekker, at metoder i ViewModels opfører sig som forventet (f.eks. at AddProductToTransaction korrekt reducerer lagerbeholdningen og tilføjer varen til kurven).
    - **Interaktion med repositories**: Sikrer, at de korrekte metoder på repository-laget bliver kaldt, når en kommando udføres.

Begrænsninger

Som nævnt: Vi valgte at **oprette Reoler med IDENTITY**, hvilket skaber den begrænsning at hvis man sletter en reol, så i stedet for at lave hvad der burde være reol 80, hvis 79 findes, så vil den blot give den næste i rækken. Den måde vi viser reoler, er at der kan være 1-80 reoler (som kan sættes højere hvis brug for), men dette laves ud fra reolnummer, så hvis man sletter en reol, og reolnummer 97 laves, vil den ikke vises.
Derfor er man nødt til fx at resette IDENTITY tælleren til den reol der skal indsættes hver gang, og så sætte den op til den højeste bagefter.
En løsning til dette, kunne have været at bruge en anden løsning end IDENTITY, og i stedet noget der vælger den lavest ledige værdi.