Tværfagligt projekt

Design af administrativt it-system for EDC

Gruppe 5.
Mark Ruge, Philip Møller, Benjamin Hansen og Andreas Thomsen.

Indhold

1. Indledning

1.1 Formål

Som en del af datamatikeruddannelsen har vi de studerende på 1. semester haft mulighed for at arbejde på et tværfagligt projekt. Projekt omhandler en case fra ejendomsmæglergruppen EDC og deres behov for et administrativt IT-system.

Vores mål er at udvikle et administrativt it-system til EDC, der kan håndtere og visualisere informationer om ejendomsmæglere, boliger og kunder (købere og sælgere). It-systemet udvikles i C# og Windows Forms, og det understøttes af en database i SQL. Til at designe systemet vil vi udarbejde relevante UML-diagrammer, og arbejdsprocessen vil blive udført ud fra et agilt perspektiv ved brug af Scrum. Afslutningsvis vil vi gennem forskellige iterationer teste og forbedre prototyper af systemet.

Krav til det administrative It-system:

- Håndtere informationer om ejendomsmæglere, boliger og kunder (køber og sælger)
- Programmet skal kunne tilpasse view og beregninger ift. den enkelte afdeling
- Der skal kunne trækkes en liste over solgte boliger, som skal indeholde en gennemsnitkvadratmeterpris. Listen skal kunne afgrænses efter postnummer, kvadratmeter og boligtype.
- Ejendomsmæglere skal kunne oprette nye boliger med relevante boligdetaljer, samt tilknyttet sælger og den primære ejendomsmægler.
- Ejendomsmæglerne skal kunne opdatere salgsprisen, og den skal kunne markeres solgt til en køber med den aktuelle handelspris og dato for salget.

1.2 Afgrænsning

Vi har valgt at afgrænse systemets funktionalitet til opgavebeskrivelsen, men vi har også tilføjet ekstra funktioner, som vi anser for væsentlige for programmets funktionalitet.

I vores system er der ikke integreret it-sikkerhed. For eksempel er der ikke implementeret login for hverken admin eller ejendomsmægler til beskyttelse af kundeinformationer. Dette ville nok være at forvente i en virkelighedsnær situation. Hvis vi havde lavet en loginfunktion, ville vi også have tilføjes en bedre løsning for afdelinger. Dette ville betyde, at hver mægler kun kunne se de postnumre de

postnumre i deres afdeling, mens admin ville have overblik over alle boliger i systemet, uanset afdeling.

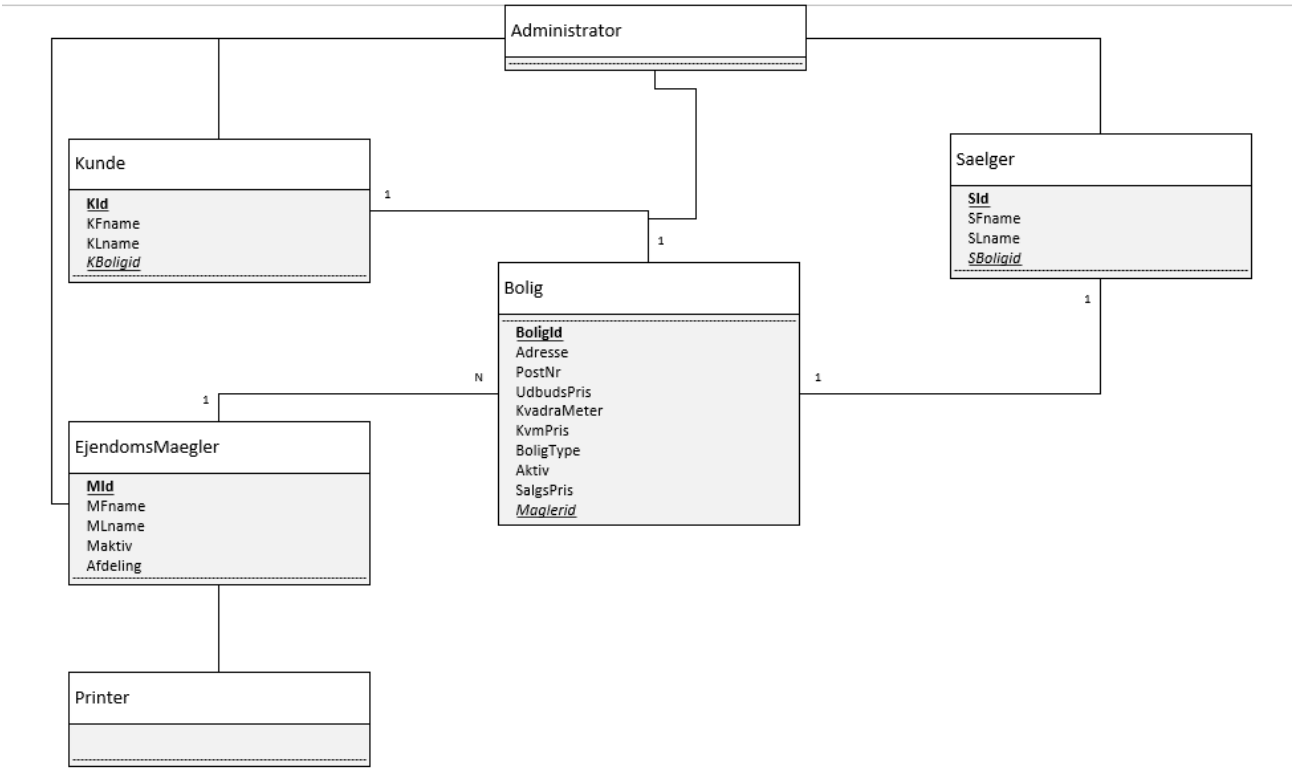
Derudover har vi valgt at begrænse mængden af brugerinputs. For eksempel er det ikke muligt at indtaste et postnummer manuelt, når en ny bolig skal oprettes. I stedet skal brugeren vælge fra en dropdown-menu. Ydermere er der kun en håndfuld postnumre, som man kan vælge imellem. Det er for at sikre at listen ikke bliver for lang med for mange postnumre med én eller få boliger. Listen af postnumre opdateres ikke dynamisk, men man skal derimod ind og tilføje nye manuelt. Alternativt kunne vi have implementeret database tabel med en liste af alle postnummer i Danmark, matchede sammen med deres tilsvarende by navne.

Der er bevidst valgt ikke at lave en opret sælger knap under sælger forms. Det er valgt, grundet vi ikke ønsker at oprette en sælger som ikke er tilknyttet en bolig. Derfor oprettes sælger i samme øjeblik som man opretter en ny bolig i systemet.

2. Systemmodellering

2.1 Domænemodel

Vi har lavet en domænemodel for at give læseren et hurtigt og klart overblik over strukturen i vores it-system. Domænemodellen giver også et overblik over aktørernes attributter og deres indbyrdes interaktioner. Alle klasser undtagen printer har fået et ID for at gøre det nemmere, at finde bestemt information i databasen. Printer er tilføjet, da vi senere skal kunne eksportere data omkring, solgte boliger.



FIGUR 1 – DOMÆNEMODEL

2.2 Use Case og use case diagram:

Ud fra opgavebeskrivelsen har vi udarbejdet en række use cases, der beskriver de forskellige aktørers krav. Vi har nummereret de forskellige use cases og illustreret dem i et use case-diagram der kan ses i figur 2 på side 4. Manage skal forstås som CRUD (Create, read, update, delete).

Håndtering af Ejendomsmægler:

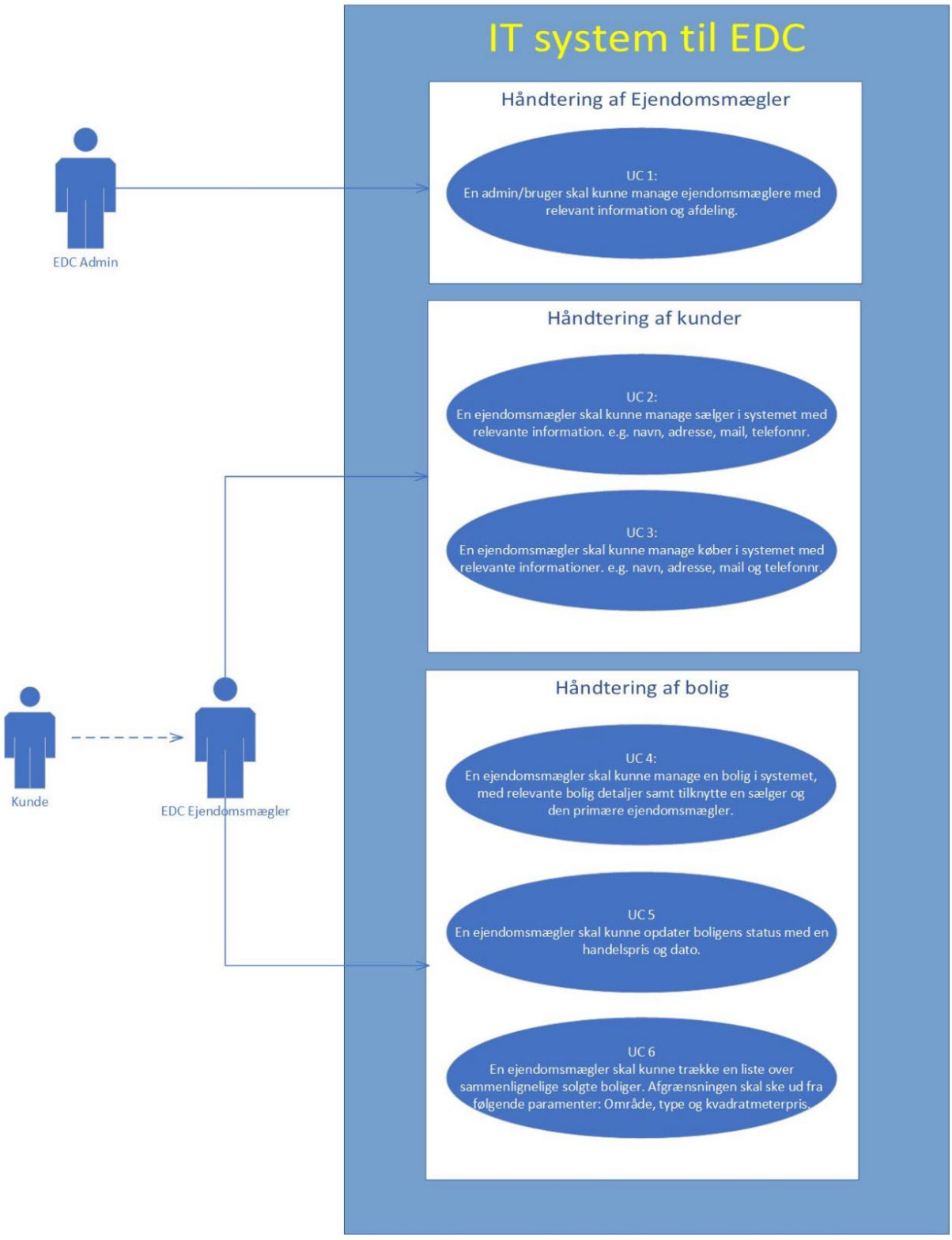
- UC 1: En admin skal kunne manage ejendomsmæglere med relevant information og afdeling.

Håndtering af kunder

- UC 2: En ejendomsmægler skal kunne manage sælger i systemet med relevante information.
- UC 3: En ejendomsmægler skal kunne manage køber i systemet med relevante informationer.

Håndtering af Bolig

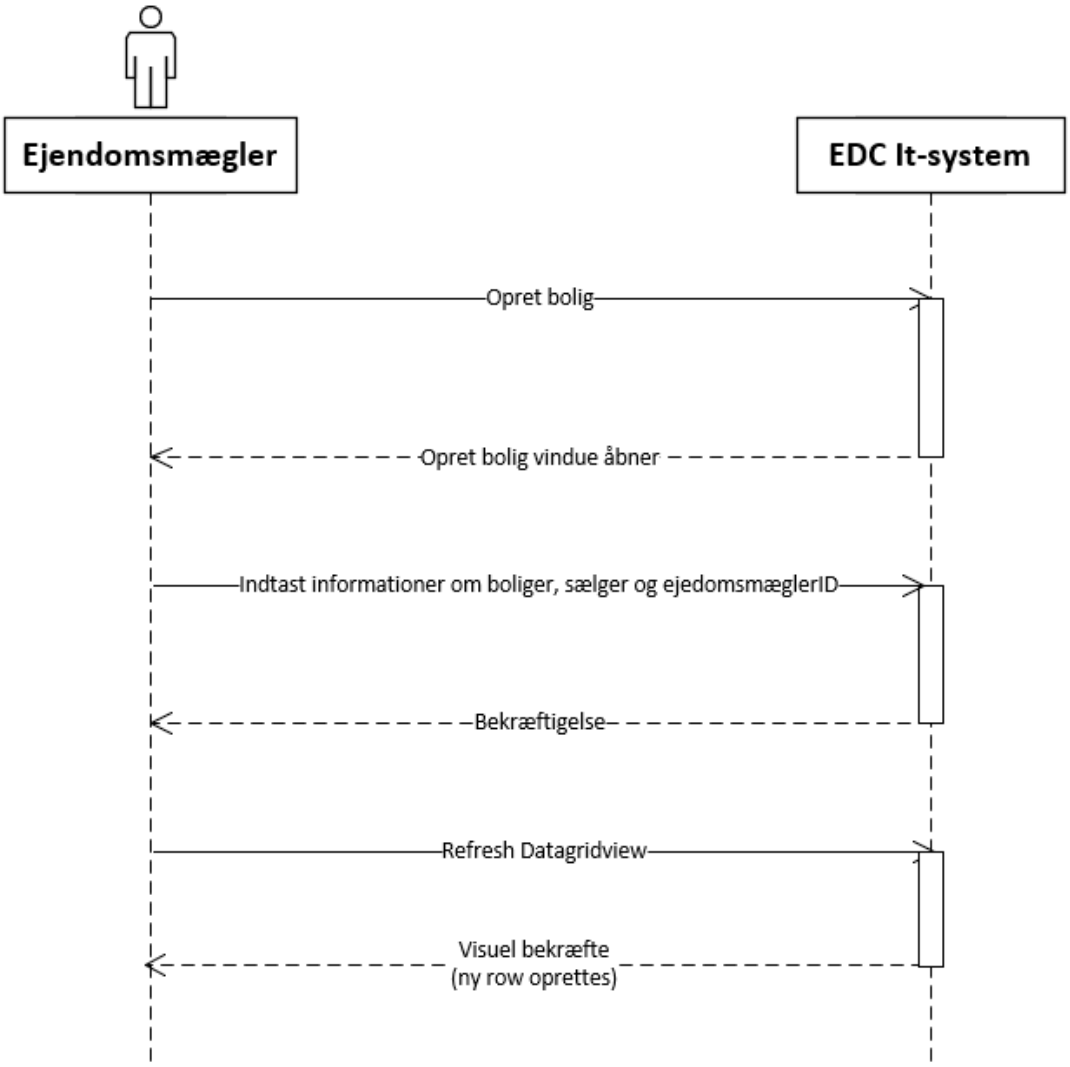
- UC 4: En ejendomsmægler skal kunne manage en bolig i systemet, med relevante bolig detaljer samt tilknytte sælger og den primære ejendomsmægler.
- UC 5: En ejendomsmægler skal kunne opdatere boligens status med en handelspris og dato.
- UC 6: En ejendomsmægler skal kunne trække en liste over sammenlignelige solgte boliger. Afgrænsningen skal ske ud fra følgende paramenter: Område, type og kvadratmeterpris.



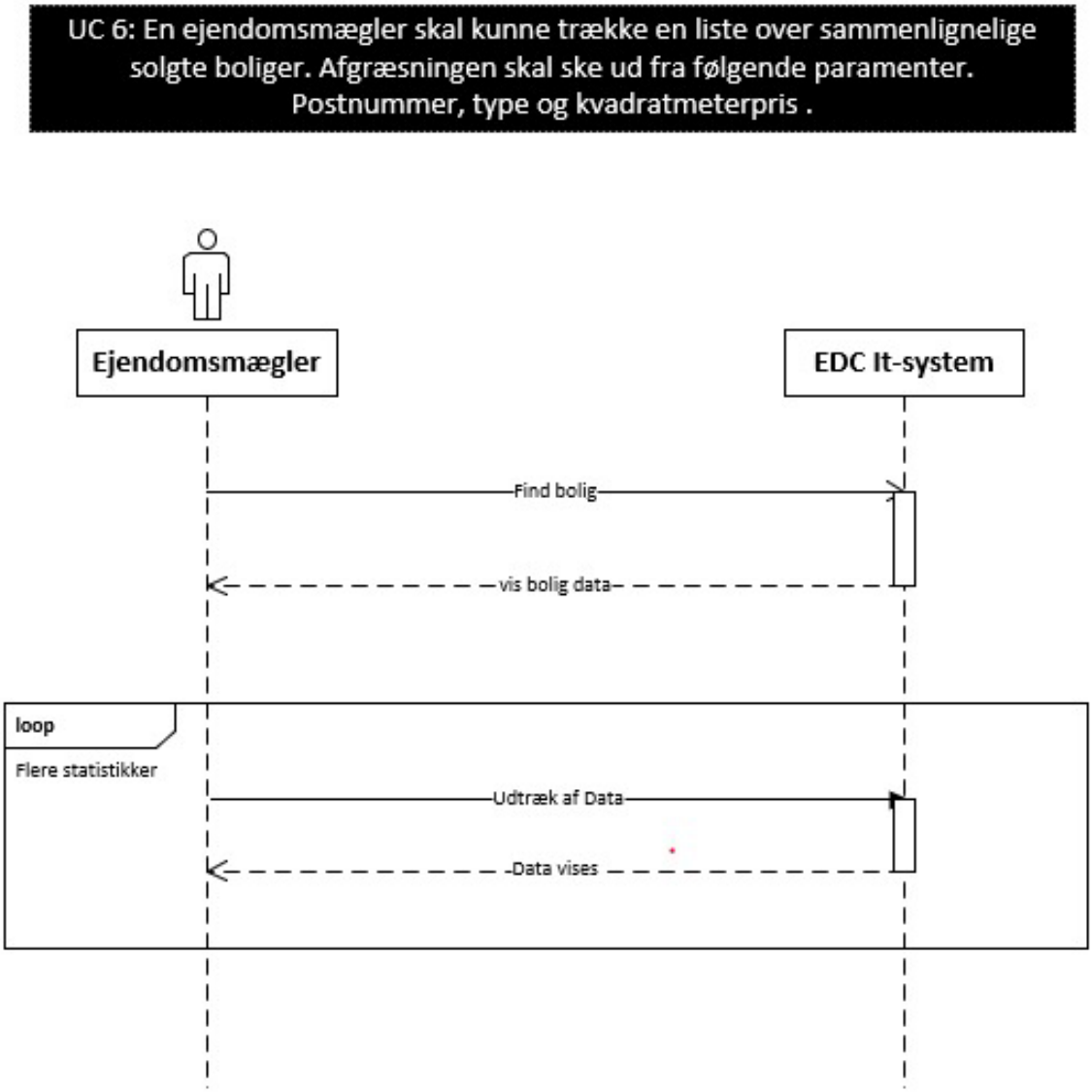
FIGUR 2 - USE CASE DIAGRAM

2.3 System Sekvens Diagrammer

For at holde en rød tråd igennem opgaven, har vi valgt at highlighte enkelte use cases. I dette tilfælde oprettelse af bolig og udtræk af data. Ejendomsmægleren skal have mulighed for at starte en session omkring oprettelse af en bolig. Når boligen er oprettet med relevante informationer, skal systemet sende en bekræftelse tilbage til bruger, der viser, at boligen er oprettet. I vores anden use case skal ejendomsmægleren kunne vise og sortere i data over sammenlignelige solgte og aktive boliger.



FIGUR 3 - SYSTEMSEKVENSDIAGRAM



FIGUR 4 - SYSTEMSEKVENSDIAGRAM

2.4 Entitetstype, Relationstype, Diagram (ERD)

Vi har udarbejdet et Entitetstype RelationsType Diagram (ERD). Dette diagram har vi brugt til at skabe overblik over hvilke entitetstyper der findes i systemet, samt deres relationer. Navngivningen er sket på dansk, med fokus på at undgå æ, ø og å, ved attributterne og entitetstyperne. Relationen mellem entiteterne er alle en 1 til mange relation. En ejendomsmægler kan godt have flere boliger på markedet, med flere sælgere af boligerne, samt potentielle kunder. Men en bolig, sælger og kunder kan kun have en ejendomsmægler.

Navngivningen på attributterne er desuden sket med forkortelser ved personerne, for at lettere kunne differentiere dem, samt sikre at SQL queries ikke møder udfordringer ved attributter som hedder det samme i forskellige tabeller.

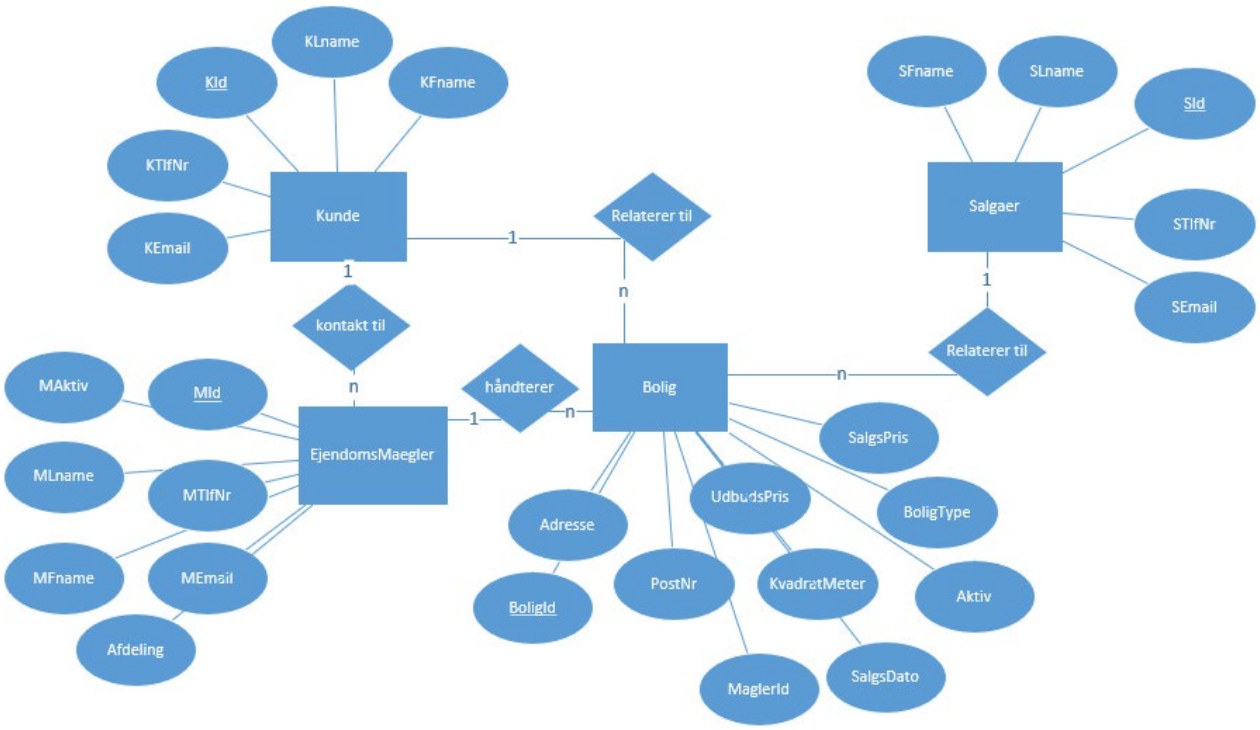
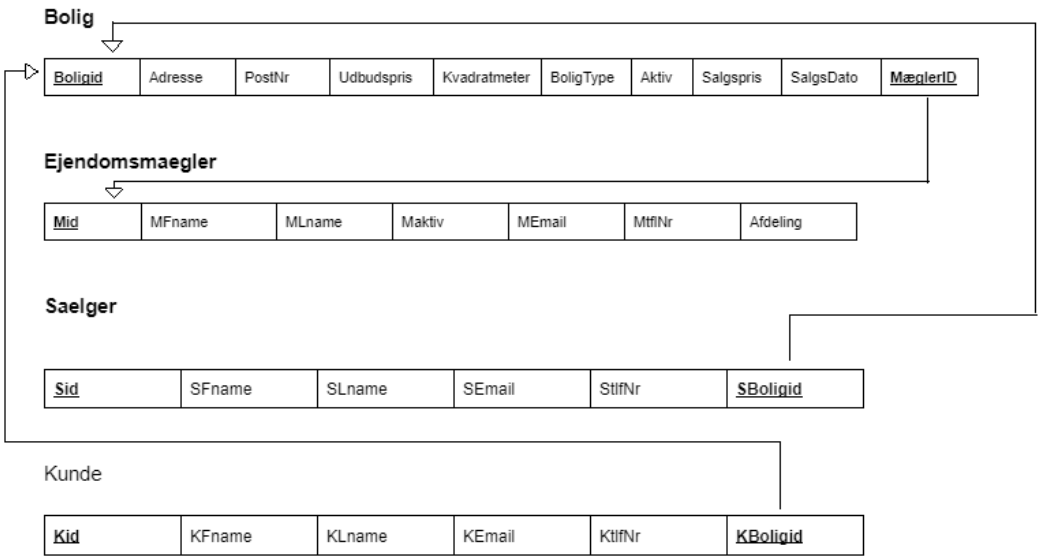


FIGURE 5 - ERD DIAGRAM

2.5 Mapning af ERD til tabeller

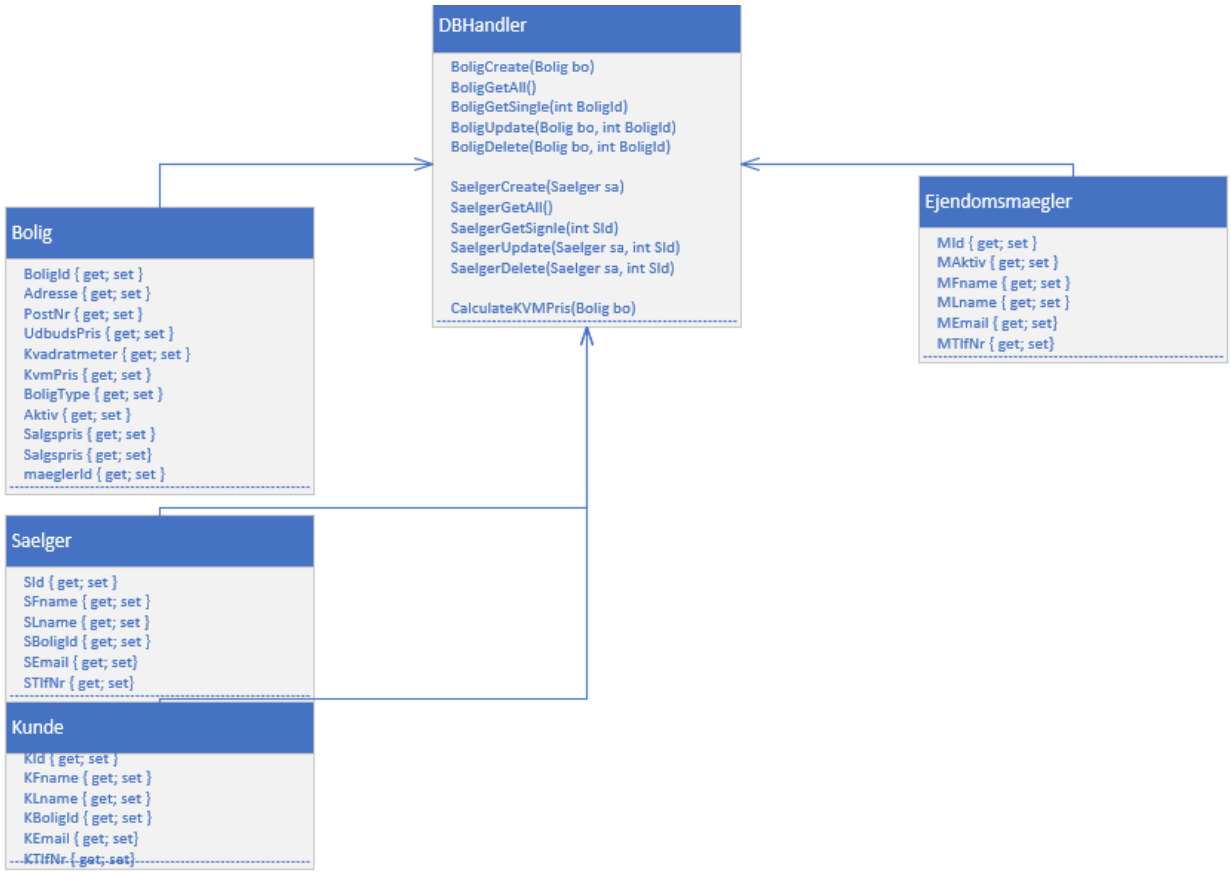
Databaseskemaet er opstillet til at simplificere databasearkitekturen. Her ses de forskellige kolonner i hver tabel. Markeret med fed er primærnøglen ved hver tabel. Længst til højre er fremmednøglerne markeret samt en grafisk repræsentation af hvilken primærnøgle de refererer til.



TABEL 1 - MAPNING AF DATABASEN

2.6 Designklassediagram

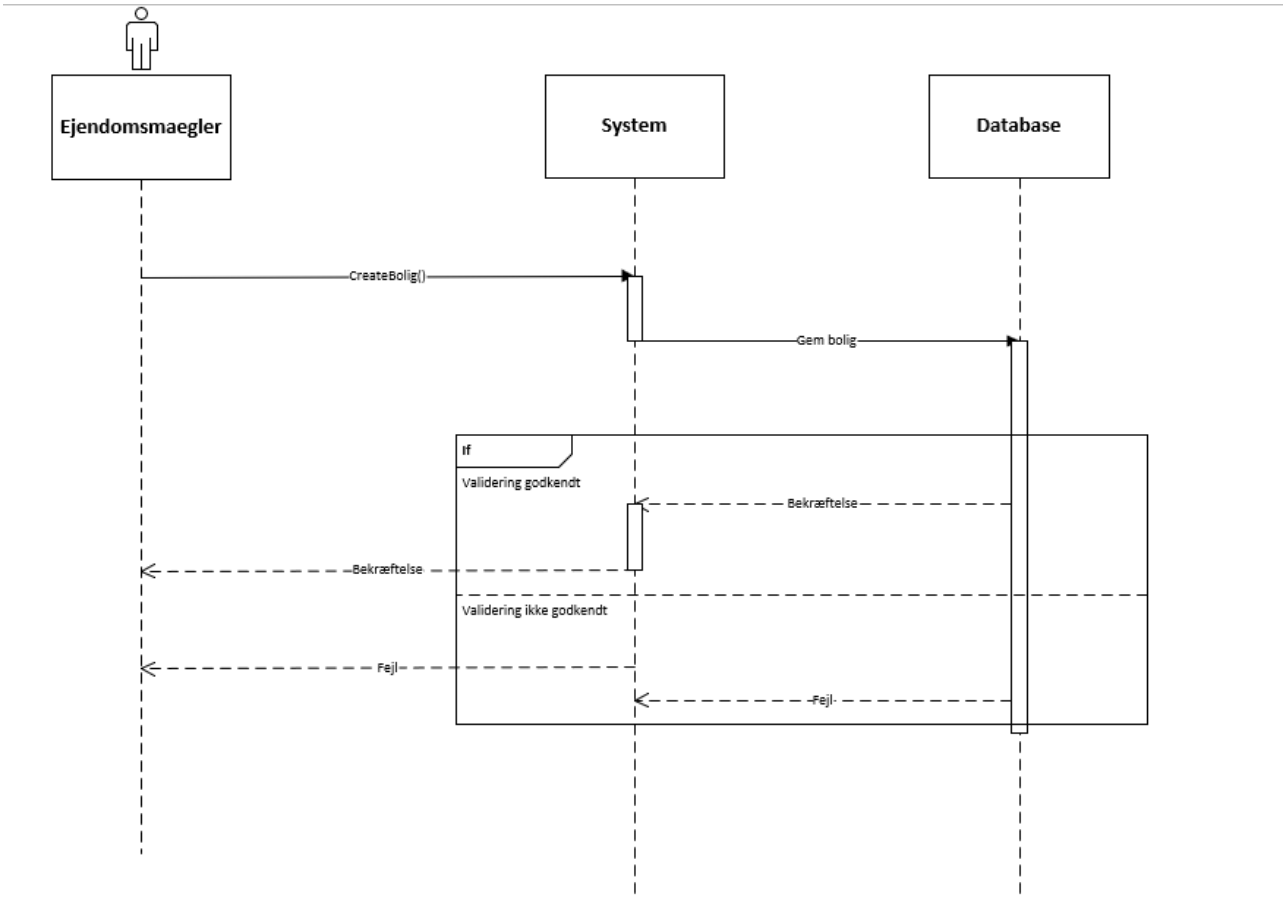
Vi har lavet et design klasse diagram for at give et bedre overblik over de forskellige klasser samt de metoder, der skal implementeres i systemet. Diagrammet visualiserer hvordan vores DB Handler kommer til at håndterer alle CRUD-operationer.



FIGUR 6 - DESIGNKLASSEDIAGRAM

2.7 Sekvensdiagram

Nedenstående sekvensdiagram viser en ejendomsmægler, der er i gang med at oprette en bolig. Diagrammet viser, hvordan opret bolig funktionen forløber i vores system. Som set på billedet kalder ejendomsmægleren createbolig() funktionen, hvor de relevante detaljer er indtastet. Systemet validerer om brugerinput er korrekte og gemmer boligen i databasen. Hvis oplysningerne er korrekte, sendes en bekræftelse tilbage i form af et popup-vindue. Hvis oplysningerne ikke er korrekte, sendes en fejlmeddelelse, som beder brugeren om at tjekke de indtastede værdier.



FIGUR 7 - SEKVENSDIAGRAM

3. Brugervenlighed og UI-design

3.1 Gestaltlove og brugervenlighed.

For at give brugerne af it-systemet en intuitiv og brugervenlig oplevelse har vi designet vores system ud fra de fem gestaltlove. I vores system har vi primært anvendt følgende tre love:

- **Loven om nærhed**

Loven om nærhed beskriver, at elementer der står tæt på hinanden, opfattes som hørende sammen. Dvs. at hvis noget tekst og et billede står ved siden af hinanden, så høre de sammen.

Vi har brugt loven om nærhed på alle vores knapper, labels, infobokse. Vi har opdelt vores menuer i venstre og højre side, så det er adskilt fra andet data. Vi har valgt, at på alle bokse og labels som brugeren skal interagere, står teksten inden i boksen som placeholder tekst.

- **Loven om lighed**

Loven om lighed beskriver, at elementer der ligner hinanden opfattes som om hørende sammen. Det kan f.eks. være form, farve eller placering.

- **Loven om lukkethed**

Loven om lukkethed beskriver, at elementer der står i samme ramme opfattes som hørende sammen. Det smarte ved at ramme elementer ind, er at man kan placere dem tættere på hinanden uden, at det bliver uoverskueligt for brugeren. Vi har f.eks. brugt loven om lukkethed under oprettelse af ny bolig. Her er Bolig informationer og Sælger af bolig grupperet i en boks hver for sig (Rolf Molich).

3.2 Brugervenlighed og funktionalitetstest.

Til at teste vores prototype 4.0 bad vi en uvillig testperson om, at foretage nogle simple operationer i vores program og ”tænke højt” om processen (Rolf Molich). For at sikre at testpersonen fik afprøvet alle væsentlige funktioner, havde vi udarbejdet en række opgaver som testpersonen skulle løse (se bilag 2). Opgaverne indebar blandt andet:

- Manage en ny bolig, sælger, køber og ejendomsmægler.
- Lav et udtræk til en csv fil.
- Test søge- og sorterfunktionerne.
- Ændre status på en bolig til at være solgt.

Feedback til funktionaliteten af systemet:

1. Testpersonen ville gerne kunne justere størrelsen på de forskellige datagridviews.
2. Testpersonen fandt en bug der gjorde, at det ikke var muligt at opdatere udbudsprisen på udvalgte købere.
3. Testpersonen fandt en bug, der fik sælg bolig til at crashe programmet.

Feedback på brugervenligheden:

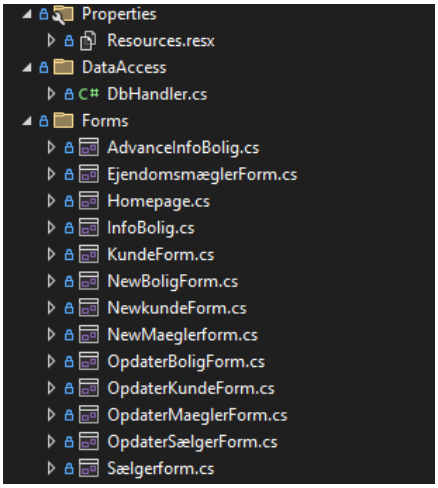
4. Testpersonen ville gerne have at overskrifterne i tabellerne var mere intuitive. I stedet for ”MFname” og ”MLname” burde der stå Mælger fornavn og mægler efternavn.
5. Teksten i visse labels var tvetydig. F.eks. er det ikke tydeligt om inaktive boliger under mægler henviser til solgte boliger eller boliger, som er taget af marked på Mæglersiden.
6. Testpersonen havde et ønske om tusindseparator i drop down-menuen ”pris” på boligsiden.

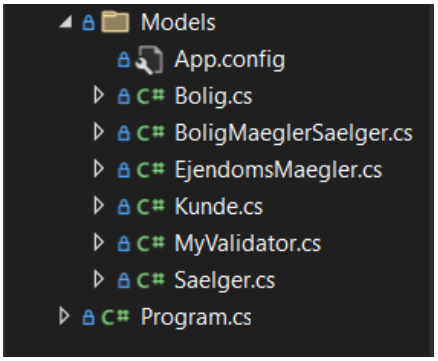
Implementeret ændringer og forbedringer efter feedback

1. Vi har ikke fundet en løsning, hvorpå brugeren selv kan justere datagridviewsene.
2. Vi havde ved en fejl slettet noget kode. Funktionen virker igen.
3. Simpel løsning. Kvadratmeterprisen udregneren var sat til at dividere med udbudsprisen. Hvis den var 0, så crashede programmet.
4. Dette var ret simpelt at implementere. I koden kunne vi ændre header teksten fra ”MFname” til ”Fornavn”
5. Vi har ændret teksten på de labels, som testpersonen fandt tvetydige.
6. Vi har ikke fundet en løsning på problemet med tusindseparator i drop down-menuen ”pris”

4. Programmering

Vores Solution Explorer er bygget op på den måde at det er nemt at navigere rundt i. Mapperne har fået tildelt navne, hvorefter vores klasser bliver placeret i den passende mappe. F.eks. så bliver alle vores modeller som kunde.cs og Ejendomsmaegler.cs placeret i modelmappen. Alle vores forms er placeret i en forms mappe. Det gør det ligeledes hurtigt og nemt at navigere rundt i, uden at skulle bladre igennem alle modeller og DBHandleren. Selve koden bag de forskellige forms kan findes ved at klikke på det det lille ikon, der står ud for de forskellige forms. På den måde kan man undgå selve designet og gå direkte til koden.





Alle vores metoder er samlet i klassen DBHandler, som er placeret i vores DataAccess mappe. Det gør det hurtigt og nemt, at benytte de forskellige metoder under programmeringen af systemet. DBHandler håndtere også vores connectionstring til alle databasekald.

4.1 Eksempler på kode

Vi har valgt at fremhæve vores boligsortering funktion. Ved hver ændring af indeks eller checkbox i sorterings kravene, kaldes metoden BoligSorting(). Hver gang der ændres i en parameter, køres metoden igen. Metoden er opbygget af en masse if statements, som bruger Lambda udtryk, der er indkapslet i en try-catch. Metoden kalder DbHandlerens GetAllBolig, som den indsætter værdierne i en liste.

Herfra checker første if statement om søgebaren indeholder noget tekst, hvis den indeholder noget så checker algoritmen om der er en adresse i listen som indeholder den tekst. Dernæst er det comboboxe med forskellige valgmuligheder i form af en dropdown-menu, som alt efter hvad der bliver valgt, sorteres der efter.

På linje 119 starter if-statement som sortere på pris. Her er der igen en dropdown-menu men som indeholder en range. Derfor gemmes minimum og maksimum værdi for den valgte combobox, som Lambda udtrykket igen tjekker for ved brug af en indbygget "Where" metode. Begge tjekbokse tjekker om den er krydset af, dernæst køres Lambda udtrykket som ser om boligen er aktiv eller solgt, så man kan danne sig et overblik blandt alle boliger om hvilke er til salg og få informationer om de solgte boliger. Se billeder forneden.

```
7 references
private void BoligSorting()
{
    try
    {
        List<Bolig> boligListe = db.GetAllBolig();

        // Søgbar ændring
        if (!string.IsNullOrEmpty(txtSearchbar.Text))
        {
            boligListe = boligListe.Where(b => b.Adresse.Contains(txtSearchbar.Text)).ToList();

        }

        // Boligtype ændring
        if (ComboBoxBoligtype.Text != "BoligType" && !string.IsNullOrEmpty(ComboBoxBoligtype.Text))
        {
            boligListe = boligListe.Where(b => b.BoligType == ComboBoxBoligtype.Text).ToList();
        }

        // postnummer ændring
        if (comboBoxPostNr.Text != "PostNr" && int.TryParse(comboBoxPostNr.Text, out var postNr))
        {
            boligListe = boligListe.Where(b => b.PostNr == postNr).ToList();
        }

        // pris ændring
        if (comboBoxPris.Text != "Pris" && !string.IsNullOrEmpty(comboBoxPris.Text))
        {
            var prisen = comboBoxPris.Text.Split("-");
            if (prisen.Length == 2 && int.TryParse(prisen[0], out int minimumPris) && int.TryParse(prisen[1], out int maximumPris))
            {
                boligListe = boligListe.Where(b => b.UdbudsPris >= minimumPris && b.UdbudsPris <= maximumPris).ToList();
            }
            else
            {
                throw new ArgumentException("Invalid price range format");
            }
        }

        // aktiv checkbox ændring
        if (Aktiv_checkbox.Checked)
        {
            boligListe = boligListe.Where(b => b.Aktiv == true).ToList();
        }

        // Solgt checkbox ændret
        if (checkBoxSolgt.Checked)
        {
            boligListe = boligListe.Where(b => b.Aktiv == false && b.SalgsPris > 0).ToList();
        }
```

KODEEKSEMPEL: BOLIGSORTER FUNKTION

På linje 145 starter endnu en if-statement, som kigger på de forskellige sorteringsmuligheder for hvilken rækkefølge de bliver udskrevet i. Her benyttes der switch, som har flere cases. Hver case har en tekst som er det den hedder i dropdown menuen, til den valgmulighed findes et Lambda udtryk igen, som afsluttes med break.

Hvis der ikke findes en eneste bolig i databasen, som matcher sorteringen, så popper en messagebox op med en besked om at der ingen boliger som matcher søgningen.

Dertil kører den efter endnu et if-statement, som checker om at listens count er større end 0. Hvis det er tilfældet, så kaldes metoden AveragePrice(), som gemmes i en variable og udskrives i en tekstboks i formatet for danske kroner.
Derefter rydder den hvilken bolig der er valgt i datagridview, for at gøre det mere brugervenligt.

Til sidst fremkommer catch funktionen, som gemmer den exception der vil melde fejl i en variabel. Som dertil bliver udskrevet i en messagebox, så brugeren af systemet kan se hvad der fejler.

```
143
144             // sorteringsdropdown menu ændring
145             if (!string.IsNullOrEmpty(comboxSortering.Text) && comboxSortering.Text != "Sortering")
146             {
147                 switch (comboxSortering.Text)
148                 {
149                     case "Adresse (alfabetisk)": boligListe = boligListe.OrderBy(b => b.Adresse).ToList(); break;
150
151                     case "Udbudspris (lav - høj)": boligListe = boligListe.OrderBy(b => b.UdbudsPris).ToList(); break;
152
153                     case "Udbudspris (høj - lav)": boligListe = boligListe.OrderByDescending(b => b.UdbudsPris).ToList(); break;
154
155                     case "Kvm (lav - høj)": boligListe = boligListe.OrderBy(b => b.Kvadratmeter).ToList(); break;
156
157                     case "Kvm (høj - lav)": boligListe = boligListe.OrderByDescending(b => b.Kvadratmeter).ToList(); break;
158
159                     case "Kvadratmeterpris (lav - høj)": boligListe = boligListe.OrderBy(b => b.KvmPris).ToList(); break;
160
161                     case "Kvadratmeterpris (høj - lav)": boligListe = boligListe.OrderByDescending(b => b.KvmPris).ToList(); break;
162
163                     case "Salgspris (lav - høj)": boligListe = boligListe.OrderBy(b => b.SalgsPris).ToList(); break;
164
165                     case "Salgspris (høj - lav)": boligListe = boligListe.OrderByDescending(b => b.SalgsPris).ToList(); break;
166
167                     default: throw new ArgumentException("Invalid sorting option");
168                 }
169
170             }
171             if (boligListe.Count == 0)
172             {
173                 MessageBox.Show("Ingen bolig matcher søgningen", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
174             }
175
176             DGVBolig.DataSource = boligListe;
177             if (boligListe.Count > 0)
178             {
179                 int gns = AveragePrice();
180                 textBoxGns.Text = $"{gns:C0}";
181             }
182
183
184
185             DGVBolig.ClearSelection();
186         }
187         catch (Exception ex)
188         {
189             MessageBox.Show($"Error: {ex.Message}", "Error!", MessageBoxButtons.OK, MessageBoxIcon.Error);
190         }
191     }
192
```

KODEEKSEMPEL: BOLIGSORTER FUNKTION

4.2 UI

I vores projekt har vi lagt vægt på at have et brugervenligt og flot UI. Vores UI fokuserer på let navigation, og forbedring af brugeroplevelse gennem blandt andet paneler. Vi har valgt en hovedform som vi har kaldt for homepage. Homepagen er vores centrale navigationspunkt, det er herfra vi vælger hvad vi gerne vil kigge på (Bolig, Sælger, Køber, Mægler). Hovedformen fungerer som en container, hvor vores underforms åbnes op i. For at det kunne lade sig gøre benyttede vi os af paneler. Panelerne indlæser og viser vores forms.

Når brugeren trykker på en af de fire knapper i navigationsmenuen, vil formen blive instantieret og den ønskede form indlæses i det relevante panel. Eksempel af kodningen ses under:

```
private void ButtonB_Click(object sender, EventArgs e)
{

    // Farve skifter på knapper//
    ButtonB.BackColor = Color.FromArgb(229, 159, 0);
    ButtonB.ForeColor = Color.FromArgb(35, 31, 80);

    ButtonS.BackColor = Color.FromArgb(35, 31, 80);
    ButtonS.ForeColor = Color.FromArgb(229, 159, 0);

    ButtonK.BackColor = Color.FromArgb(35, 31, 80);
    ButtonK.ForeColor = Color.FromArgb(229, 159, 0);

    ButtonM.BackColor = Color.FromArgb(35, 31, 80);
    ButtonM.ForeColor = Color.FromArgb(229, 159, 0);
    ef.Hide();
    sf.Hide();
    kf.Hide();
    Lbl_Title.Text = "Bolig";

    //Åbner Form inde i panelet i form1//
    bf.TopLevel = false;
    bf.FormBorderStyle = FormBorderStyle.None;
    bf.Dock = DockStyle.Fill;
    this.ScreenPnl.Controls.Add(bf);
    bf.Show();

}
```

KODEEKSEMPEL FRA FORMEN HOMEPAGE, DER ÅBNER FORMEN OP I PANELET

Så snart brugeren trykker på ButtonB som står for Button Bolig, vil boligformen blive åbnet op inde i ScreenPanelet. Dette sker ved at vi tilføjer kontrollen til panelet, så når der trykkes på knappen vil panelet blive til placeholderen for formen. Dette kan kun lade sig gøre ved at give formen et toplevel statement som er false, hvis ikke dette gøres kan den ikke gå igennem.

Udover det har vi også fjernet kanten på formen så det ikke er muligt at lukke ned for formen inde i panelet, samme tid har vi også sat dockstylen til fill, så formen skalerer op og ned efter størrelsen på panelet som justerer sig efter størrelsen af Homepage formen.

Ved at anvende paneler har vi skabt en effektiv brugergrænseflade, som gør det muligt at bladrer igennem de forskellige “hovedforms” uden at skulle manuelt lukke formsne ned.

4.3 Databasen

Databasen er oprettet i SQL Server Management Studio. Nedenfor ses et udsnit af vores script af databasen.

Der er lavet fire tabeller, som hver har deres attributter. Der er gjort brug af identity i alle ID, for at sikre at det altid vil være en unik værdi, som kan bruges til primærnøgle. Til det har overvejelserne været at bolig ID starter ved 1, ejendomsmægler starter ved 100 og køber og sælger starter ved 1.000, alle springer med 1.

Der er overvejet input værdier i oprettelsen af attributterne. Derfor er der valgt f.eks. varchar på maksimalt 50 i længden i Adresse i Bolig tabellen, varchar bliver i C# til en string, som er nem at arbejde med. Der er valgt at bruge int til udbudspris fremfor en float, da priserne er så høje at decimaltal ville være unødvendigt, samt til for at sikre at værdierne ikke skal konverteres for meget.

```
Create table Bolig
(
BoligId int identity (1, 1) not null,
Adresse varchar (50) not null,
PostNr int not null,
UdbudsPris int not null,
Kvadratmeter int not null,
BoligType varchar (20),
Aktiv bit,
Salgspris int,
SalgsDato date,
MaeglerId int not null,

Primary key (BoligId),
);

Create table EjendomsMaegler
(MId int identity (100, 1),
MFname varchar (20),
MLname varchar (20),
MAktiv bit,
MEmail varchar (30),
MTlfNr int,
Afdeling int,

Primary key (MId)
);

create table Saelger
(SId int identity (1000, 1),
SFname varchar (20),
SLname varchar (20),
SBoligId int,
SEmail varchar (30),
STlfNr int,
Primary key (SId),
foreign key (SBoligId) references Bolig(BoligId)
);

create table Kunde
(KId int identity (1000, 1),
KFname varchar (20),
KLname varchar (20),
KBoligId int,
KEmail varchar (30),
KTlfNr int,
primary key (KId)
);

alter table Bolig
add constraint Maegler_MId foreign key (MaeglerId) references Ejendomsmægler(MId);
```

Eksempel for indtastning af værdier i tabellerne.

Til denne tabel arkitektur, demonstreres der hvordan værdier ville blive indsat i ejendomsmæglertabellen. Her er varchars indkapslet i ‘’-tegn, hvorimod int står som almindelige talværdier uden decimal.

```
insert into EjendomsMaegler
values
    ('Jens', 'Jensen', 1, 'Jens@EDC.dk', 11223344, 1),
    ('Hans', 'Hansen', 1, 'Hans@EDC.dk', 22334455, 2),
    ('Bente', 'Holm', 1, 'Bente@EDC.dk', 66556655, 3);
```

4.4 Unit-testing

Der er lavet testing til opgaven. Eftersom opkoblingen med NUnit test ikke var succesfuld, har vi valgt at lave nogle få test. De test som er lavet ville fungere hvis opkoblingen var succesfuld. Test1 tester om vores input validering fungerer. Her indsættes værdien “Hus”, som forventes at returnere false, da validering tjekker, om inputtet er “Villa”, “Rækkehus”, “Lejlighed” eller "Andelsbolig". I og med at “Hus” ikke er en af de valgmuligheder, returnerer den false, som er det vi forventer.

```
[Test1]
public void Test1()
{
    //Arrange
    Bolig b = new Bolig();
    b.BoligType("Hus");

    //Act
    bool actual = b.ValidateHouseType();
    bool expected = false;

    // Assert
    Assert.AreEqual(expected, actual);
}

[Test2]
public void Test2()
{
    //Arrange
    Bolig b = new Bolig();
    b.BoligType("Villa");

    //Act
    bool actual = b.ValidateHouseType();
    bool expected = true;

    // Assert
    Assert.AreEqual(expected, actual);
}
```

I test2 oplever vi at input på boligtype er en af de valgmuligheder. Dermed returnerer den true som forventet. Det gør vi for at sikre at validering fungerer både ved rigtig og forkert input.

Test3 er ligesom i test2 at vi forventer en true værdi ved at indtaste et telefonnummer som har en længde der bliver testet på. Her sikrer vi igen at valideringen fungerer, så man sikrer at programmet ikke går i stå eller at input til databasen er korrekt.

```
[Test3]
public void Test3()
{
    //Arrange
    EjendomsMaegler ejendomsMaegler = new EjendomsMaegler();
    ejendomsMaegler.MTlfNr(11223344);

    //Act
    bool actual = b.ValidatePhonenumber();
    bool expected = true;

    // Assert
    Assert.AreEqual(expected, actual);
}
```

5. Arbejdsproces.

5.1 Scrum.

I dette projekt har vi benyttet scrum, som det primære værktøj til projektstyring, men med elementer fra Extreme Programming (herefter XP). Pair og mob programming er et af de værktøjer fra XP, vi har gjort meget brug af, især i udarbejdelsen af designet af UI'en. Her har en fra gruppen fungeret som ”driver”, mens resten af teamet har fungeret som ”observere” ved at følge med på skærmen og komme med inputs. Testing er et andet værktøj fra XP, som vi har brugt, og her har vi lavet unit-test på validering af boligtype og telefonnummer. Til sidst har vi brugt refactoring til løbende at rense op i koden og afslutningsvis til at fjerne overflødig kode, som ikke blev benyttet (XP Kent Beck).

Vi har i vores projekt fastsat sprinternes varighed til 4 dage. I starten af hvert sprint har lavet sprint Planning og opdatering af backloggen. I slutningen af hvert sprint har vi holdt sprint review og retrospektive, hvor erfaringer fra sprintet er blevet delt på kryds og tværs af gruppens medlemmer. Som en dagligt tilbagevendende begivenhed har vi afholdt daily standup, hvor vi har brugt 3-5 minutter hver formiddag på at tale om, hvad vi har lavet dagen før, og hvad vores plan for dagen er.

Da vi i denne opgave ikke har en direkte Product Owner (PO), har vi valgt at tildele rollen til undervisere, frivillige testpersoner og andre medstuderende. De har løbende givet os feedback, som vi har taget med videre i fremtidige sprits. Fordi det har svært at estimere tidsforbruget for udarbejdelsen af forskellige funktioner og modeller. Derfor har vi i hvert sprint afsat tid til eventuelle ændringer, der måtte være nødvendige.

5.1.1 Sprint 0.

Sprint Planning og Backlog:

Som en del af planlægningen har vi primært brugt Sprint 0 til at skabe os et overblik over casen ved blandt andet at udarbejde en grov udgave af produktbackloggen samt at formulere user stories. Vi har valgt at inddele bagloggen i essentielle funktioner, og funktioner som kan kunne være ”nice to have” hvis der var ekstra tid. Til at facilitere styring af vores scrum-proces har vi valgt at anvende Trello. Her har vi oprettet et kanban-board og organiseret vores user stories med farvekoder baseret på aktører og om funktionen er essentiel. De user stories vi ikke nåede at implementere kan ses under bilag 2.

Backlog: User stories

- Som bruger, ønsker jeg at kunne oprette, læse, opdatere og slette sælgere, for at sikre bedre funktionalitet af systemet.
- Som bruger, ønsker jeg at kunne oprette, læse, opdatere og delete informationer om bolig, så jeg kan tilgå det opdaterede informationer i databasen
- Som bruger, ønsker jeg at kunne oprette, læse, opdatere og delete informationer om mægler, så jeg kan tilgå det opdaterede informationer i databasen
- Som bruger, ønsker jeg at kunne oprette, læse, opdatere og delete informationer om køber, så jeg kan tilgå det opdaterede informationer i databasen
- Som programmør ønsker jeg at udarbejde metoderne til de forskellige CRUD-operationer, så når det bliver nemmere, og hurtigere når vi skal kode de forskellige funktioner
- Som systemarkitekt, ønsker jeg at lave validering på input, for at sikre at databasen kan tage imod input.
- Som bruger af systemet, ønsker jeg en velovervejet UI, med fokus på gestaltlove, for at skabe et intuitivt design.
- Som systemdesigner, ønsker jeg at teste SQL queries, for at kunne trække data ud af databasen, så jeg kan sikre det fungerer med formodet værdier.
- Som bruger, ønsker jeg at kunne udtrække specificeret data til en CSV-fil, så jeg bedre kan rådgive sælgere om udbudspris.
- Som bruger, ønsker jeg at kunne sortere data, så jeg kan finde det data jeg leder efter
- Som bruger af systemet, ønsker jeg at en database, så jeg kan tilgå informationer om bolig, kunder, sælgere og mæglere
- Som Product Owner (PO), ønsker jeg en prototype af UI-arkitekturen, så jeg kan få forventningsafstemt systemet.
- Som en systemarkitekt, ønsker jeg at lave et Domæne model, så jeg kan få overblik over arkitekturen
- Som en systemarkitekt, ønsker jeg at lave et Design Class Diagram (DCD), så jeg kan få et overblik over struktur og klassearkitekturen
- Som en systemarkitekt, ønsker jeg at lave et System Sekvens Diagram (SSD), så jeg kan få et overblik over systemets arkitektur
- Som systemarkitekt, ønsker jeg at lave et Database Skema, så jeg kan fortsat danne et overblik over database tabels

- Som en systemarkitekt, ønsker jeg at lave et Entitets Relations Diagram (ERD), så jeg kan få et overblik over databasestrukturen, med fokus på entitet og relationer
- Som bruger ønsker jeg at se gennemsnitsprisen på en bolig ud fra type og postnummer, så jeg bedre kan vurdere udbudsprisen for nye sælgere.

Sprint review:

Sprint 0 har været en kort proces, eftersom der ikke har været meget at teste på. Vi har som gruppe udarbejdet en masse user stories ud fra opgavebeskrivelsen.

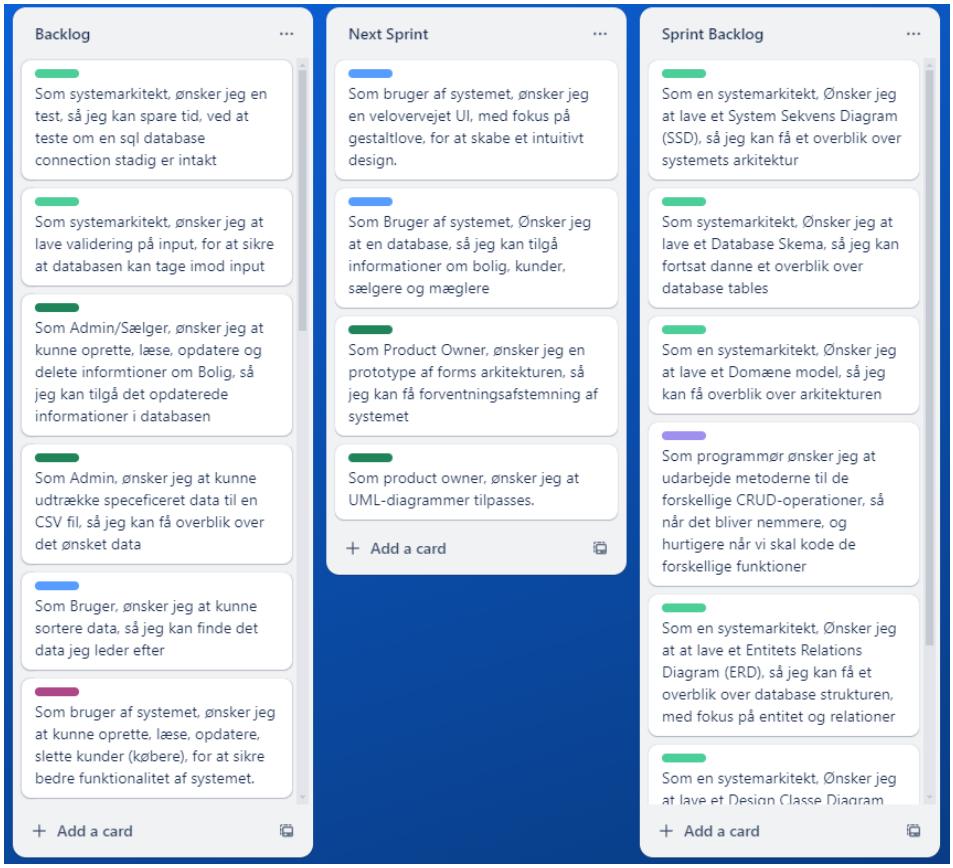
Sprint retrospektive:

Processen hjalp os med at definere projektets omfang og prioritere vores indsats i de kommende sprint. Opdelingen af produktbackloggen gav os et overblik over, hvilke funktioner der var essentielle for funktionaliteten af systemet og hvad der var ekstra.

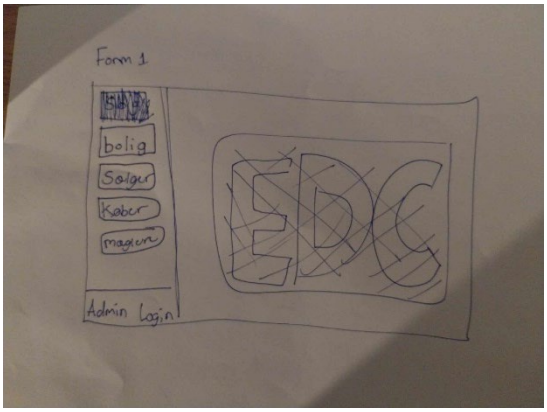
5.1.2 Sprint 1:

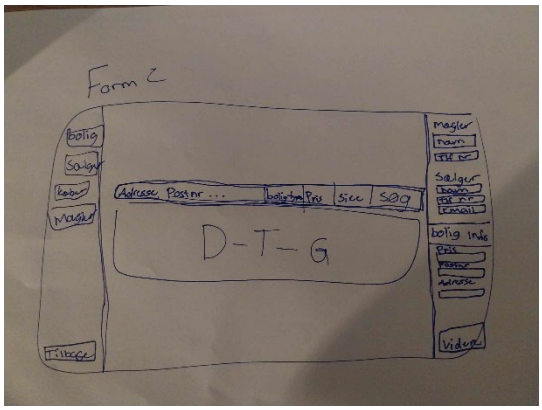
Sprint Planning og Backlog.

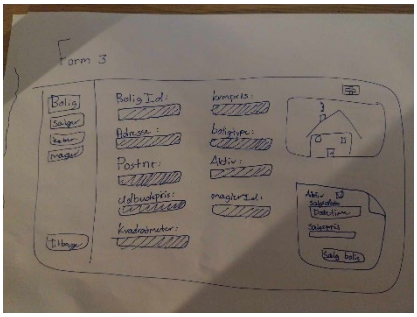
I sprint 1 er vores mål at begynde på vores model-lag og de tilhørende diagrammer. Vi vil udarbejde første udkast til domænemodellen, systemsekvensdiagrammet (SSD), design class diagrammet (DCD), EERD-diagrammet og databaseskemaet. Vi vil udarbejde metoderne til DBhandleren, så vi fremrettet kan bruge dem, når vi koder UI. Afslutningsvis ville vi gerne have et udkast til en horisontal prototype af systemet.

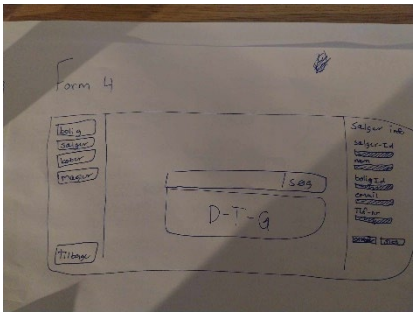


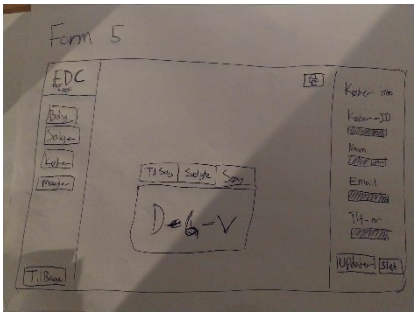
Prototype 1.0:











Sprint Review:

I dette sprint er vi blevet færdige med første udkast til diverse UML-diagrammer. Vi har også lavet en horisontal prototype 1.0 af vores UI. I slutningen af sprintet fik mundtligt feedback på vores UML-diagrammer fra vores underviser. Arbejdet med at ændre modellerne og diagrammerne har vi valgt, at tage med over i sprint 2.

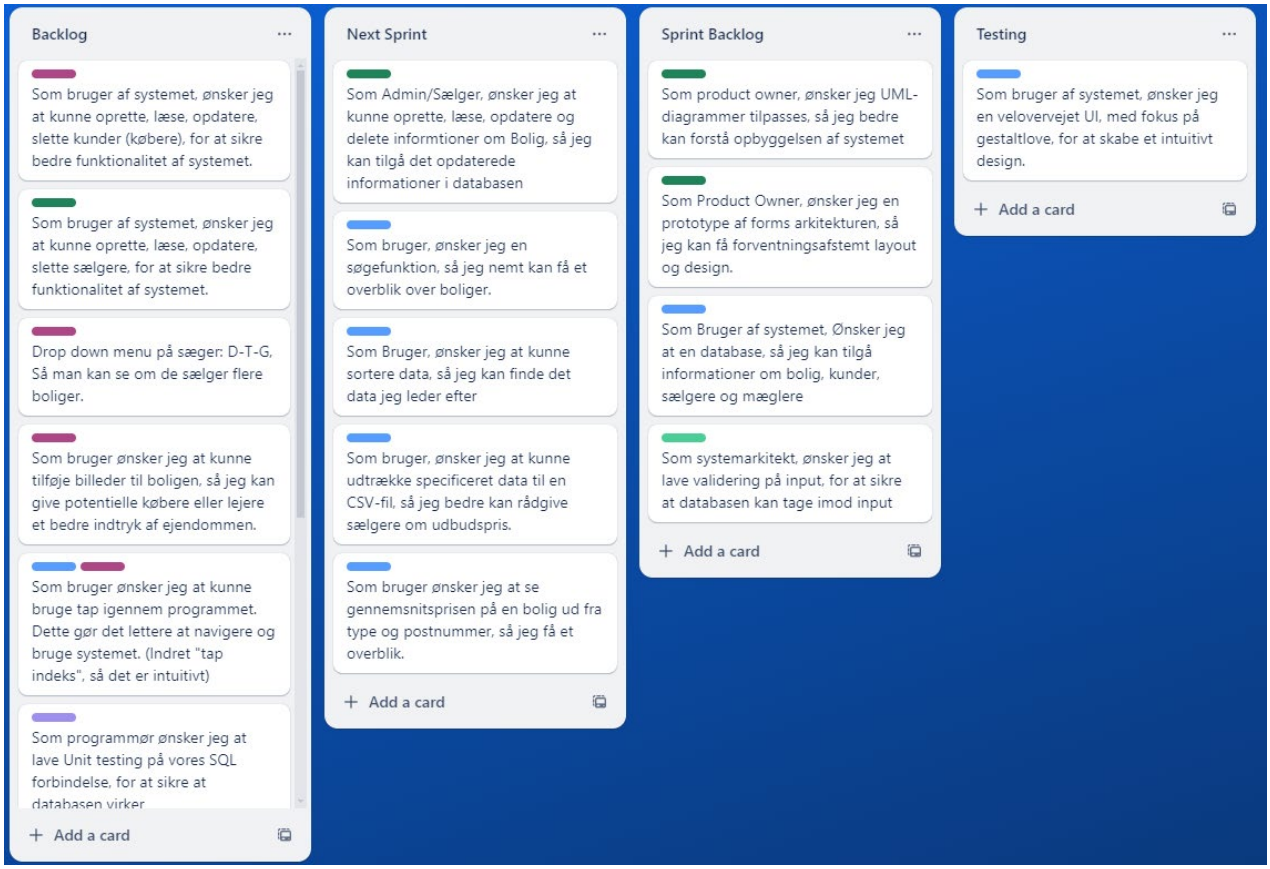
Retrospektive:

Vi har opbygget en bedre grundviden ud fra de modeller vi har lavet, samt hvilke ting der er vigtig at have i fokus når vi udarbejder systemet og UI'en.

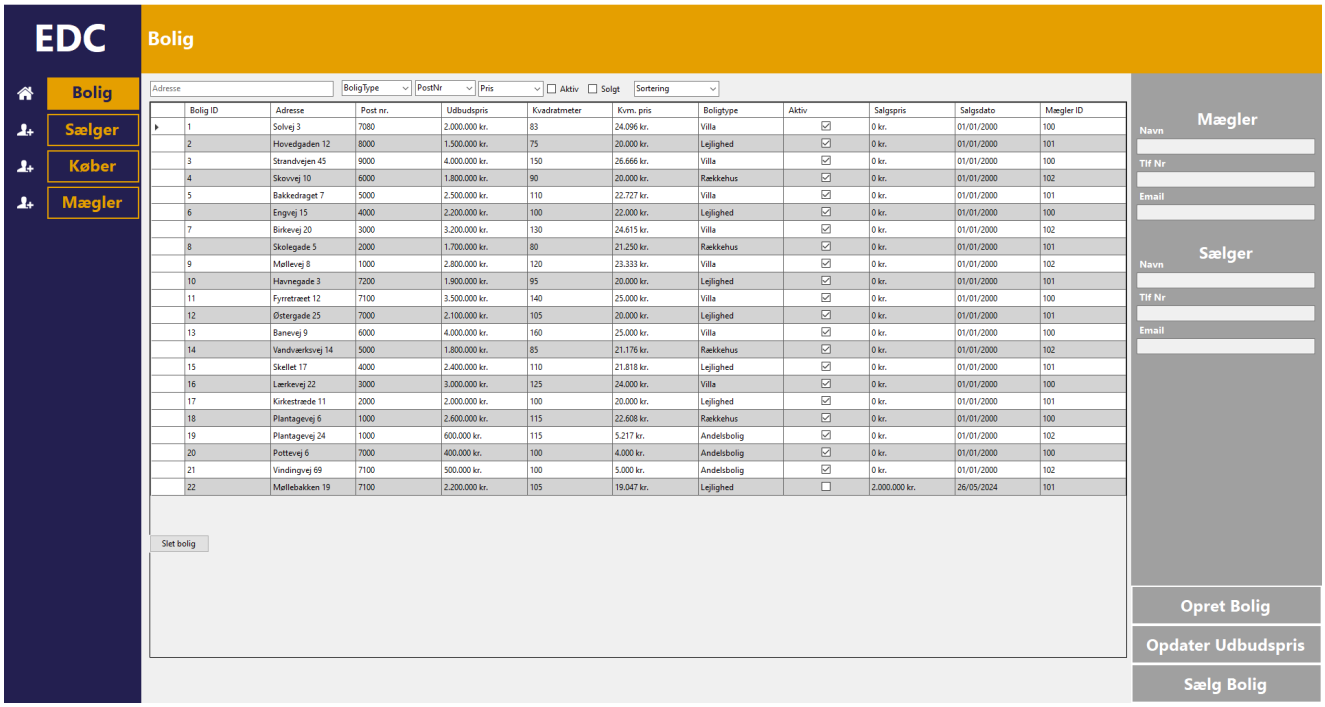
5.1.3 Sprint 2

Sprint Planning og Backlog.

I dette sprint vil vi justere vores UML-diagrammer baseret på den feedback, vi har fået fra underviser. Derudover vil vi arbejde videre på UI-designet af boligsiden. Til sidst vil vi gerne oprette og tilkoble databasen til systemet, så vi kan visualisere data.



Prototype 2.0:



Sprint review.

Vi har nået at lave rettelser til UML- diagrammerne, tilkoblet databasen og færdiggjort UI-designet på boligsiden. Datagridiewet fungere, men vi mangler at blive implementeret CRUD. Søge og sorteringsfunktionerne er sat op, men endnu ikke programmeret.

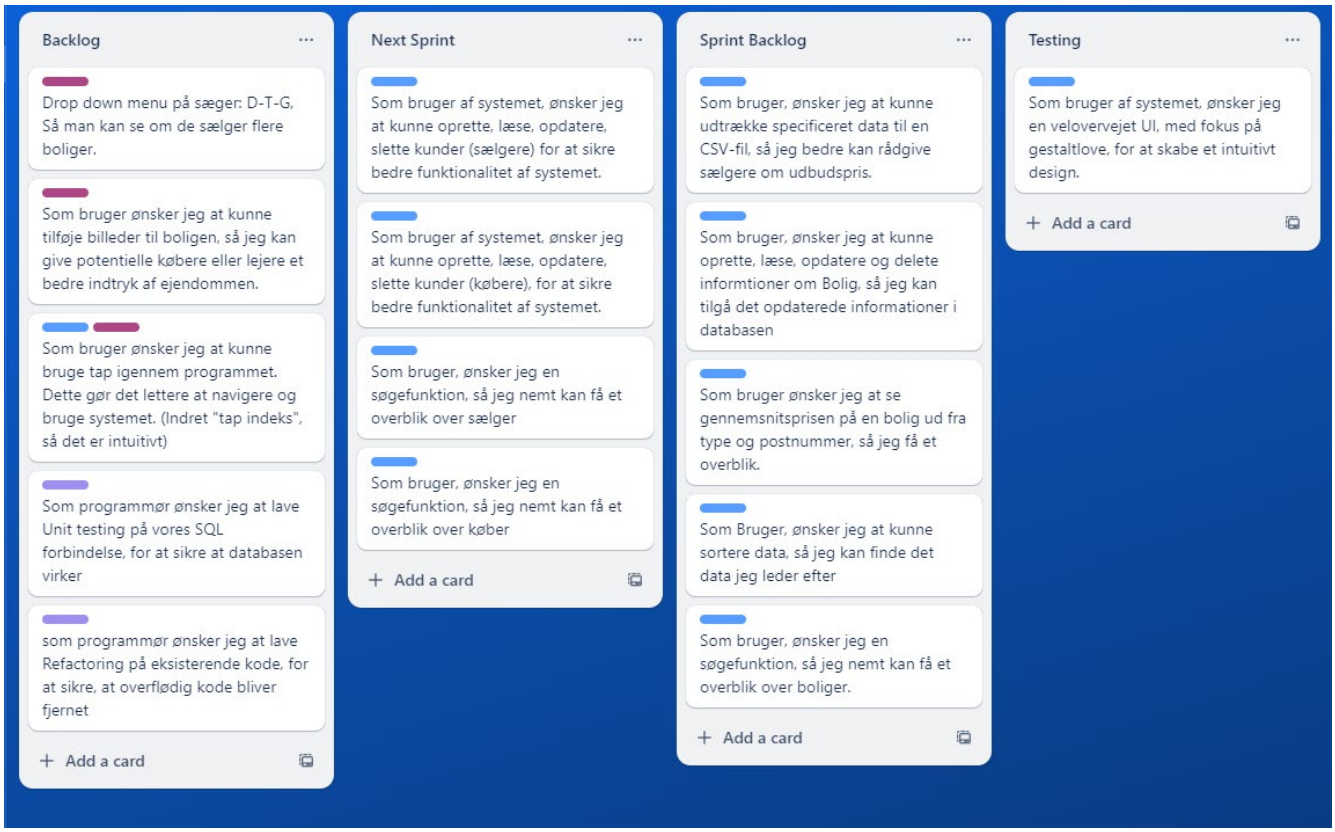
Sprint retrospektive:

Vi har i dette sprint gjort os en masse erfaringer med, hvordan design og layout skal være på boligsiden. Vi har fundet en smart løsning, der har gjort det lettere at sætte systemet op og mere navigerbart ved at åbne formene op inde i hinanden i stedet for at have dem åbne i separate faner. Dette mindsker mængden af gange vi skal lukke og åbne individuelle forms. Vi er tilfredse med vores overordnet design af systemet, og vi kan genbruge nogle af funktionaliteterne på sælger, køber og mægler siden.

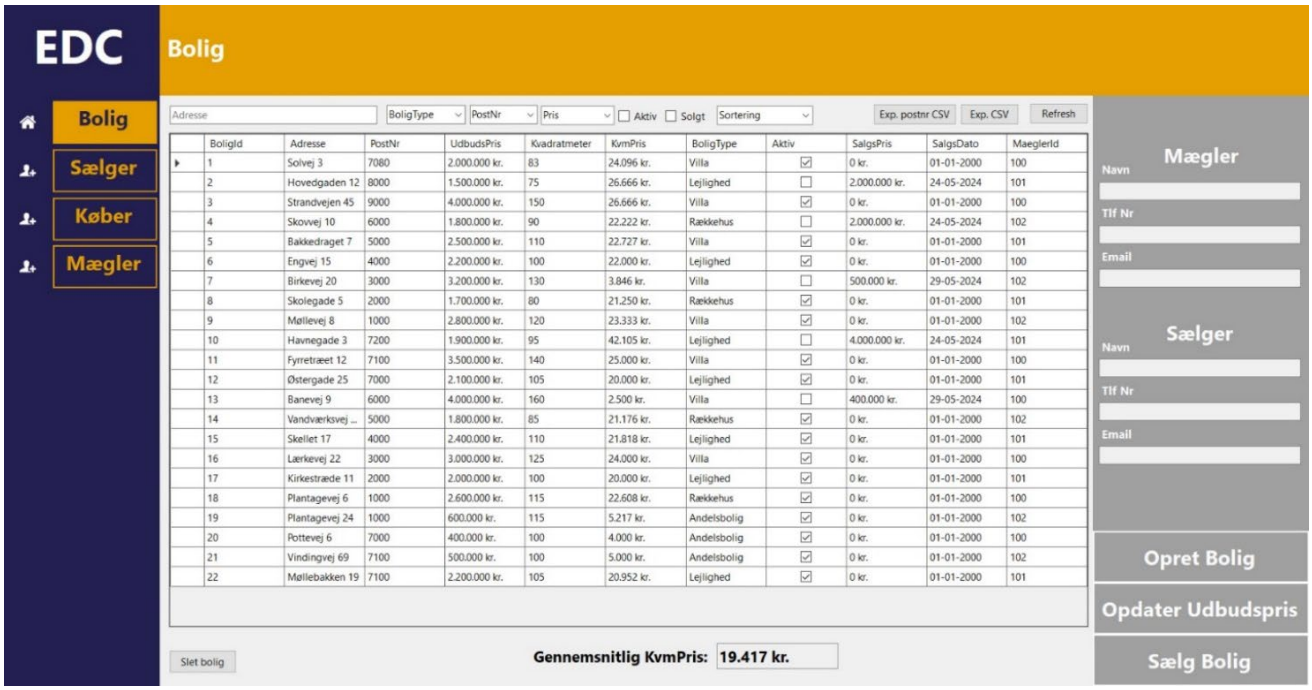
5.1.4. Sprint 3

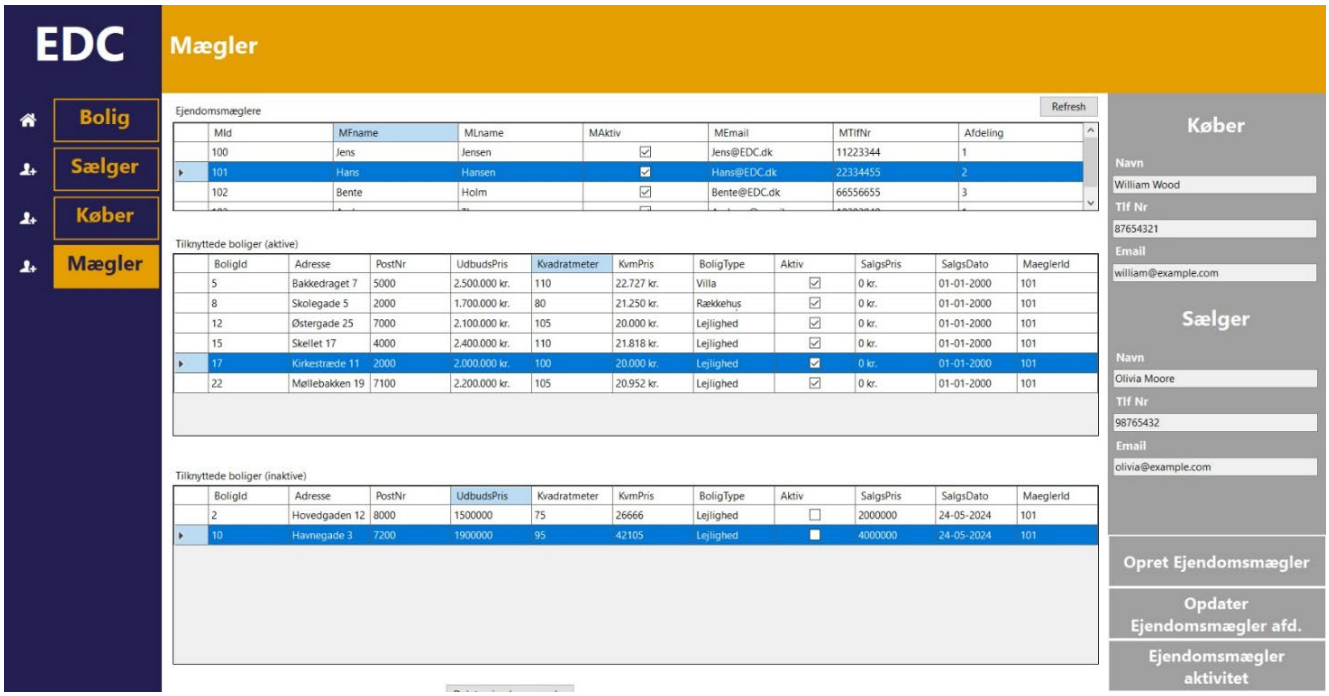
Sprint Planning og Backlog.

I dette sprint vil vi implementere CRUD funktionalitet på boligsiden og implementere søge og sorteringsfunktioner funktioner. Derudover vil vi gerne tilføje export til csv og udregning af gennemsnitspris på en bolig. Når vi har det på plads, vil vi gå videre med implementeringen af sælger, køber og mæglersiden.



Prototype 3.0:





Sprint review

I dette sprint har vi nået at færdiggøre CRUD operationer på bolig, mægler og sælgersiden. Vi har tilføjet eksport til csv fil. Derudover har vi tilføjet forskellige funktioner til datagridviewsene. For eksempel, hvis man trykker på en bolig, vises informationer om mægler og sælger i højre side. Søge og sorteringsfunktionen virker også, men de har deres begrænsninger. For eksempel skal man være meget specifik i sin søgning, da den ikke kan håndtere forskellen store og små bogstaver. Sælger og mæglersiden er også implementeret med CRUD. Vi har ikke nået at færdiggøre købersiden med CRUD operationer, så det tager vi med i næste sprint.

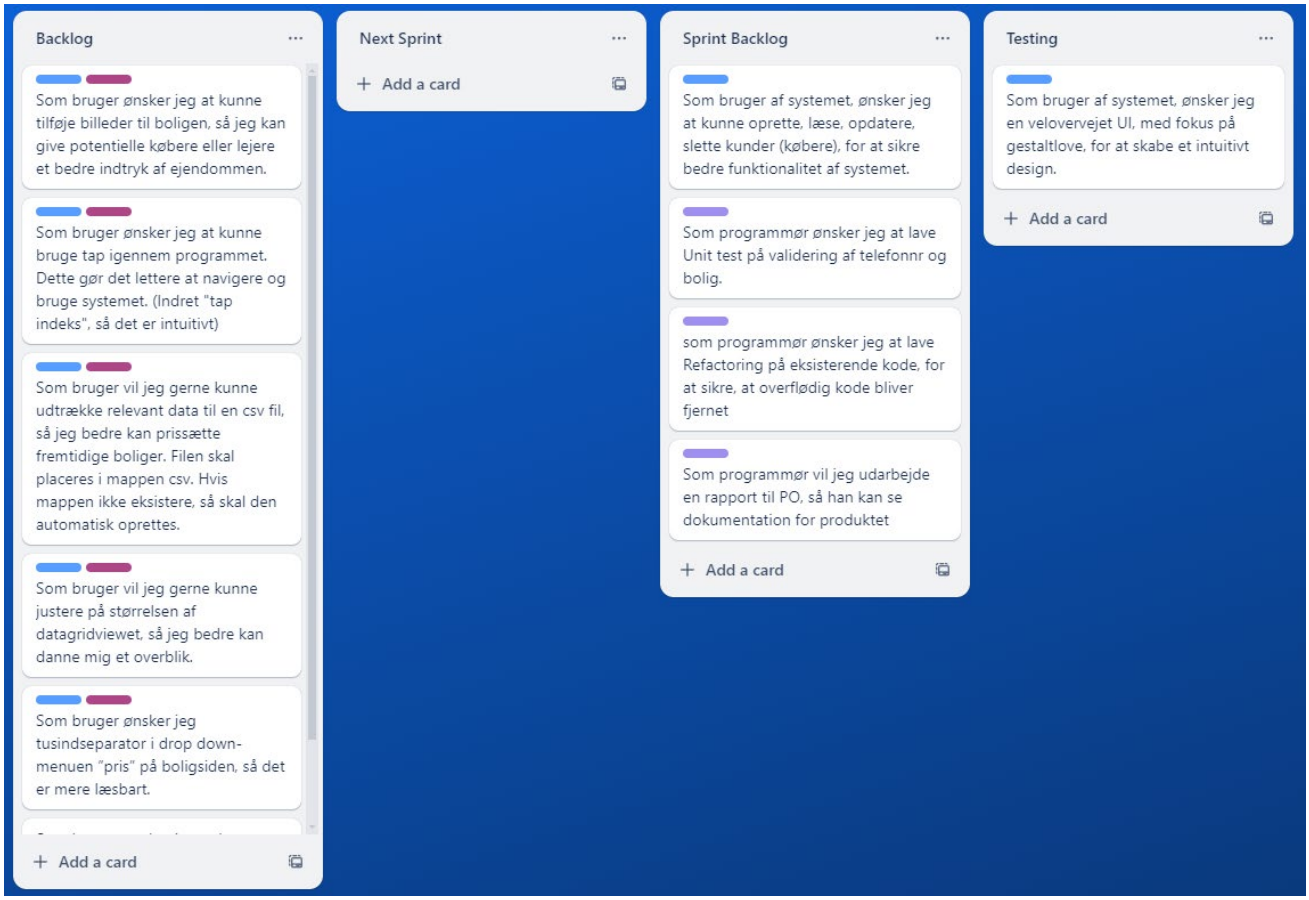
Sprint retrospektive.
Efterhånden som vi er kommet længere og længere i processen, er vi blevet bedre til at tilpasse arbejdsbyrden til længden af sprintet.

Desværre har vi fundet ud af, at der er en fejl i vores design af databasen. Fordi de forskellige ID'er er tilkoblet boligID, er det ikke muligt for en sælger at sælge mere end én bolig. Det samme problem gør sig gældende for mægler og køber. Da vi er så langt i processen, er det ikke noget vi kan ændre på nu. Vi har lært af fejlen, og det er noget, vi vil være opmærksomme på i fremtidige projekter.

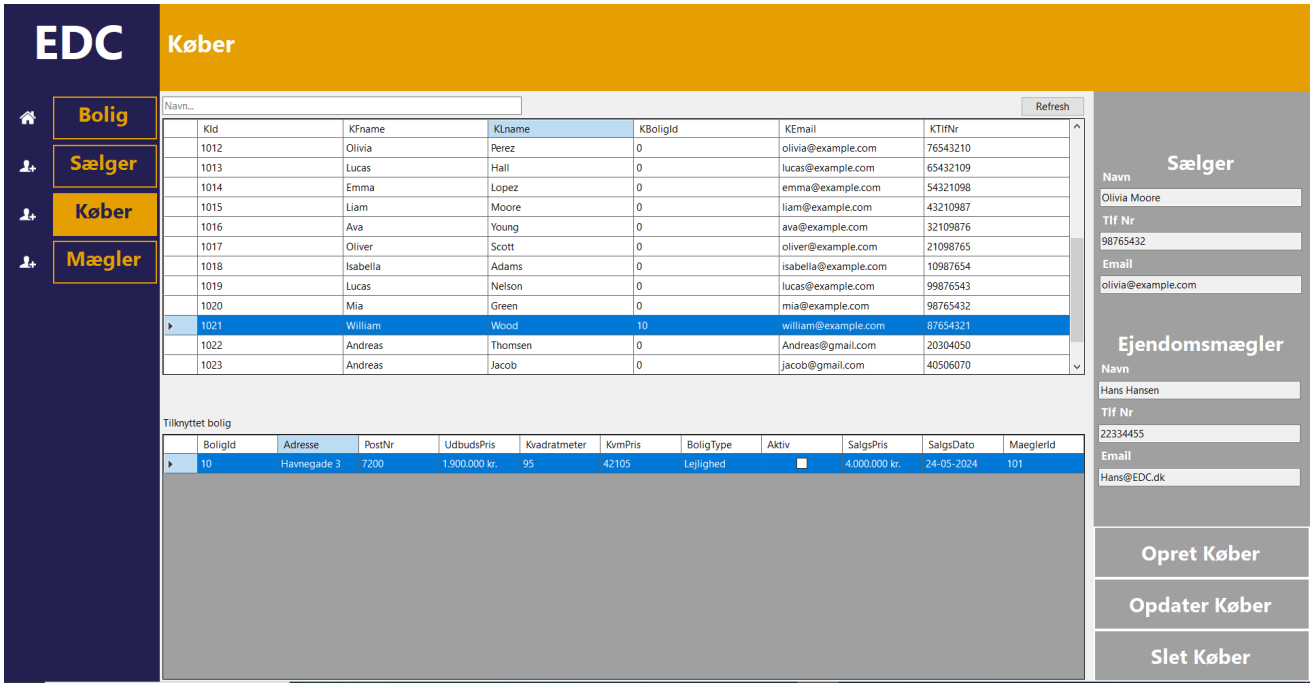
5.1.5. Sprint 4

Sprint Planning og Backlog.

I dette sprint vil vi færdiggøre rapporten, udarbejde præsentationen, færdiggøre købersiden med CRUD operationer samt udføre code cleanup. Derudover vil vi lave en unittest på validering af bolig og telefonnummer. Hvis der er tid, vil vi gerne implementere den feedback vi har fået fra vores brugervenlighedstest. De resterende "nice to have" funktioner fra backloggen kan ses under bilag 2.



Prototype 4.0:



Sprint review:

I dette sprint har vi færdiggøre rapporten, udarbejdet en præsentationen til fremlæggelsen og færdiggjort købersiden med CRUD operationer. Vi har efterfølgende lavet cleanup på koden, samt finpudset systemet ud fra den feedback vi har fået fra testpersonen i vores brugervenlighedstest.

Sprint retrospektive:

Da det meste af tiden er gået med finpudsning af programmet og rapportskrivning har der ikke været det store, at reflektere over. Arbejdsprocessen har været delt meget ud, så nogen har kigget på systemet mens andre har skrevet på rapport.

6. Perspektivering

6.1 Overvejelser

Vi har igennem hele projektet løbende haft overvejelser til design, brugervenlighed, funktionalitet mv. En vigtig ting, vi har lært fra dette projekt, er at sikre, at arkitekturen af vores program er sat rigtig op fra starten. Vi fandt sent ud af, at vi havde lavet en fejl, da vi oprettede databasen. Vi havde valgt at tilkoble SælgerID, KundeID og MæglerID til BoligID, hvilket medførte nogle uhensigtsmæssige begrænsninger. For eksempel er det ikke muligt at tilknytte flere nye boliger til samme mægler. Det er heller ikke muligt for en sælger at have flere boliger til salg på samme tid. Dette er en lektie, vi vil tage med os, når vi i fremtidige projekter skal designe arkitekturen.

I fremtidige projekter vil det nok være smart at navngive attributterne i databasen. Attributterne kan derefter arve f.eks. fra en ”Person” classe.

Scrum er udfordrende at implementere når man bliver stillet en opgave som har en meget vandfaldsarkitektur. Derfor har vores Scrum proces været mindre optimal, dels pga. opgaven men også grundet det nye kendskab til Scrum i gruppen. Dertil fungere scrum bedst over en længere periode uden en fastlagt tidsbegrænsning, som desværre ikke er tilfældet for denne stillede opgave.

6.2 Konklusion

I dette læringsforløb er vi endt ud med en masse ny viden og fået en bedre forståelse for eksisterende koncepter. Vi er ganske tilfredse med det system, vi har udviklet, og at vi har formået at implementere alle de nødvendige funktioner, som casen efterspurgte. En af de ting, vi har lært, er at når man arbejder på at færdiggøre et produkt eller prototype indenfor en bestemt tidsramme, så er det afgørende at prioritere, hvile funktioner der er essentielle, og hvad der er nice-to-have. En anden lektie, vi har lært, er vigtigheden af at gennemtænke databasestrukturen grundigt, for at undgå begrænsninger senere i udviklingen.

Processen har givet os en smagsprøve på hvad det vil sige at designe software og mange af de erfaringer vi har fået, er noget, vi kan tage med videre til fremtidige projekter.

Bilag:

Bilag 1 -Brugervenlighedstest – Tænke højt!

Spørgsmål til testpersonen:

Funktionalitet:

1. **Prøv at opret en bolig på boligsiden**
2. **Prøv at opdatere udbudspris på boligsiden**
3. **Prøv at sælge en bolig på boligsiden.**
4. **Under "Bolig" test om du kan bruge søgefunktionen. Prøv at sortere i forskellige boligtyper, postnummer, priser og om boligen er aktiv.**
5. **Under "Bolig" test om du kan eksportere en CSV-fil.**
6. **Under "Bolig" test om du kan eksportere en CSV-fil ud fra postnummer.**
7. **Prøv at opdatere sælgers informationer.**
8. **Prøv at slet en sælger på sælgersiden.**
9. **Prøv at opret en køber på købersiden.**
10. **Prøv at opdatere en køber på købersiden.**
11. **Prøv at slet en køber på købersiden.**
12. **Prøv at oprette en ejendomsmægler på ejendomsmæglersiden**
13. **Prøv at opdatere en ejendomsmægler på ejendomsmæglersiden**
14. **Prøv at gør en ejendomsmægler inaktiv på ejendomsmæglersiden**

Brugervenlighed:

15. **Er systemet nemt at navigere rundt i?**
16. **Er fejlmeddelelser tydelige og forståelige?**
17. **Er farver og layout lavet fornuftigt?**
18. **Er overskrifter og labels tydelige og forståelige?**
19. **Andre bemærkninger?**

Bilag 2. Backlog med user stories vi ikke nåede at implementere.

- Som bruger vil jeg gerne kunne udtrække relevant data til en csv fil, så jeg bedre kan prissætte fremtidige boliger. Filen skal placeres i mappen csv. Hvis mappen ikke eksistere, så skal den automatisk oprettes.
- Som bruger vil jeg gerne kunne justere på størrelsen af datagridviewet, så jeg bedre kan danne mig et overblik.
- Som bruger vil jeg gerne kunne tilføje billeder til en ny bolig, så jeg bedre kan præsentere boligen for potentielle købere.
- Som bruger ønsker jeg at kunne bruge tap igennem programmet. Dette gør det lettere at navigere og bruge systemet.
- Som bruger ønsker jeg at kunne bruge enter når man har færdiggjort indtastningen af oplysninger og skal gå videre.
- Som bruger ønsker jeg tusindseparator i drop down-menuen ”pris” på boligsiden, så det er mere læsbart.
- Som bruger ønsker jeg et brugerlogin til systemet, så jeg kan tilgå min afdeling og se relevante oplysninger.
  Som admin ønsker jeg et login til systemet, så jeg kan oprette mæglere.

Litteraturliste:

- Brugervenligt webdesign Rolf Molich (8 sider) Meddelelser og THA.pdfMonic
- Rolf Molich. De 5 gyldne regler og Gestaltlove. Pdf udleveret af underviser
- https://www.jacobleander.dk/brugervenlig.html
- Kent Beck. XP. pdf udleveret af underviser.
- Craig Larman. Applying UML and Patterns. An introduction to object-oriented analysis and design and iterative development.
- ChatGPT. Brugt til at genere data til databasen (Fornavne, efternavne, emails, tlfnr osv.) og til at rette korrektur.