

Project Getting real
Hold: DMOoF24
Medlemmer:
24-05-2024

Project Getting Real
Bookingsystem – Egevang Nord
Udarbejdet af
Jesper, Jeppe, Jakob, Jonathan & Rafed



Abstrakt

Dette projekt har haft som formål at udvikle et bookingsystem til booking af fælleslokaler for Boligforeningen Egevang Nord. Krav og ønsker til systemet er primært blevet formuleret i direkte kontakt med en repræsentant for Egevang Nord, og det indledende arbejde i dette projekt bestod således i højere grad af strukturering af disse informationer frem for en større analyse af virksomheden.

Dette er sket igennem udviklingen af high-level-artefakter såsom Use Cases og Objekt- og Domænemodeller. I arbejdet mod low-level-artefakter har vi udviklet System Sequence Diagrammer med tilhørende operationskontrakter.

På baggrund af ovenstående artefakter har vi diskuteret hvordan koden skulle struktureres, og eftersom UI-delen af projektet er udarbejdet i Windows Presentation Foundation, er vores primære design-mønster MVVM-mønstret.

Da det også var et krav til programmet, at det skulle kunne håndtere persistens, har vi hentet inspiration fra Repository-mønstret, og har således et lag mellem database og Models-lag, der håndterer kommunikationen mellem disse. Dette arbejde har ledt os til at udforme et Design Class Diagram, der har ligget til grund for vores endelige kode og prototype. Prototypen er funktionel, og lever op til de fleste af vores krav til programmet. Dog er vi ikke nået helt i mål med alt ønsket funktionalitet, og der er således mulighed for at arbejde videre med systemet henimod en fuldstændigt implementeret løsning, der opfylder samtlige krav til systemet.

Indholdsfortegnelse

Indledning og problembeskrivelse	4
Løsningsforslag.....	5
Virksomhedsanalyse.....	5
Proces.....	6
Design.....	7
High Level Design.....	7
Use cases	7
Objektmodel	8
Domænenmodel	10
Wireframes	11
System Sequence Diagram.....	13
Low Level Design	15
Design Class Diagram.....	15
Resultat og implementering	16
Kode-struktur	16
Konklusion, diskussion og refleksion	18
Litteraturliste	19

Indledning og problembeskrivelse

Da et af vores gruppemedlemmer havde kendskab til, at boligforeningen i Egevang Nord manglede et system til at håndtere booking af deres fælleslokaler, kontaktede vi foreningen for at drøfte muligheden for at udvikle en prototype til dette formål. Således kom en aftale i stand om, at vi i løbet af projektperioden skulle udfærdige dette system så vidt muligt. Det var i denne forbindelse vigtigt for os at tydeliggøre over for vores kontaktperson, at der kun var tale om en prototype, og at denne ikke nødvendigvis ville blive færdig eller indeholde alle ønskede funktioner, men at vi ville bestræbe os på at få lavet så komplet en prototype som muligt.

Beboerne i boligforeningen har tidligere været afhængige af e-mailkorrespondance for at tjekke ledigheden for de tilknyttede selskabslokaler. Denne metode har resulteret i manuel registrering af bookinger og en uoverskuelig proces for administratorerne. Den nuværende tilgang er ineffektiv og tidskrævende, hvilket medfører flere problemer såsom manglende overblik, risiko for dobbeltbookinger og generel frustration blandt både beboere og administratorer.

For at løse disse udfordringer har vi fået til opgave at udvikle et dedikeret bookingsystem, der kan automatisere og optimere processen for både beboere og administratorer. Ved at implementere et sådant system kan boligforeningen opnå en mere struktureret og effektiv håndtering af lokaleudlejningerne. Dette vil forbedre overblikket, minimere fejl, øge transparensen for beboerne og generelt forbedre den samlede oplevelse.

Systemet skal gøre det nemt for beboerne at reservere lokaler og giver administratorerne et klart og opdateret overblik over alle bookinger. Sammen med en repræsentant fra boligforeningen er der blevet identificeret essentielle såvel som ønskelige funktioner, som kan forbedre systemet yderligere.

For at gøre bookingsystemet fyldestgørende, er det vigtigt at have to brugerflader: en til beboerne for booking og en til administratorerne for at kunne tilgå bookinger og redigere i systemet. Ydermere var vores intention, at systemet skulle være brugervenligt for administratorerne ift. at tilføje og tilgå regler for bookinger, samt at have mulighed for skalering i fremtiden uden afhængighed af systemudviklere.

Ved at tage disse skridt vil Boligforeningen Egevang Nord kunne forbedre deres interne kommunikation og service til beboerne, reducere risikoen for fejl samt frigøre tid og ressourcer. Dette projekt vil derfor bidrage til en mere gnidningsfri og tilfredsstillende bookingoplevelse for alle involverede parter.

Løsningsforslag

Vi havde i vores løsningsforslag opdelt potentielle funktioner i to kategorier: “Need to have” og “Nice to have”. Målet var at skelne mellem absolut nødvendige funktioner og fordelagtige funktioner for at løse Egevang Nords problemstillinger.

De nødvendige funktioner (“Need to have”) inkluderede en kalender til booking, som skulle være skalérbar og indeholde regelsæt og information om lokalerne. Systemet skulle være synligt for alle beboere gennem en kalender og have en administrator-brugerflade, hvor administratorer kunne ændre regelsættet for lokalerne og justere begrænsninger via en administrator indstillingsside. Der skulle være mulighed for at begrænse antallet af bookinger pr. beboer ad gangen, og beboerne skulle kunne slette deres egne bookinger. Systemet skulle også have en timer på booking-adgang, en tidsbegrænsning for hvor langt frem man kunne booke og en listevision af planlagte bookinger.

Ved at have viden om de nødvendige og fordelagtige funktioner, kunne vi sikre, at vi fokuserede på Egevang Nords primære behov, samtidig med at vi havde mulighed for at tilføje ekstra funktionalitet, hvis tiden tillod det.

Virksomhedsanalyse

Vi har fra start været opmærksom på, at projektets fokus ville være en meget afgrænset del af boligforeningens samlede virksomhed og værdiskabelse. Ydermere var vores første skridt i projektet at tale med vores kontaktperson i foreningen og herigennem afklare, hvad problemet bestod i, og hvad funktionaliteten skulle indebære for at opfylde ønskerne til en løsning. Vi vurderede således, at det ikke var relevant, at vi kastede os ud i en større analyse af boligforeningen, udformede et Business Model Canvas eller lavede en dybere undersøgelse af forretningsprocesser – den stillede opgave var allerede klart defineret og afgrænset.

Proces

For at strukturere vores arbejde i projektperioden, har vi benyttet os af Scrum-frameworket. Vi har ikke eksplicit defineret individuelle Scrum-roller i gruppen. I stedet har vi alle sat os ind i, hvad der kendetegner de forskellige roller og i fællesskab taget ansvar for, at disse områder blev håndteret.

Det primære værktøj fra Scrum der har dikteret vores fremgangsmetode, har været vores Scrum-board. Her har vi tilføjet de opgaver, der har været mest nødvendige.

Undervejs i projektet blev vi opmærksomme på at hvert sprint skulle dække over en use case. Denne manglende forståelse påvirkede projektets forløb markant og resulterede i ineffektiv tidsstyring og prioritering. Trods at vi ikke struktureret har benyttet Scrums iterative og inkrementelle processer med udgangspunkt i Use Cases, har vi naturligt arbejdet med visse egenskaber i Scrums ceremonier, såsom Sprint planning og Daily scrum.

Vores mest hyppige fremgangsmetode for gennemførsel af elementer i Product backlog har været par-programmering, hvor vi efter dagens arbejde har mødt til en Sprint Retrospective for at diskutere vores fremgang eller resultat.

Trods at vi ikke helt konsekvent har fulgt Scrum-strukturen og gennemgået samtlige Scrum-ceremonier i hvert sprint, har vi afholdt Sprint Review Meetings, inden vi er gået videre til at opdatere opgaverne i vores Product Backlog.

Dette har givet os mulighed for at justere vores tilgang løbende og sikre, at vi hele tiden har forbedret vores arbejdsprocesser. Ved at tilpasse os i løbet af projektet har vi sikret, at vores projekt er forløbet mere gnidningsfrit.

Design

High Level Design

Use cases

Vi har udvalgt nogle få eksempler på Use Cases:

Beboer ønsker at booke et lokale:

- Beboeren logger ind på bookingportalen og vælger ønsket dato og tidspunkt.
- Systemet viser tilgængelige lokaler baseret på beboerens afdeling
- Beboeren vælger det passende lokale og bekræfter reservationen.
- Reservationen bliver bekræftet booket.

Administrator skal redigere regelsæt for lokaler:

- Administrator logger ind på brugerfladen for administratorer og navigerer til indstillingerne for lokaler.
- Administrator ændrer reglerne, f.eks. tidspunkter for tilgængelighed eller maksimalt antal bookinger pr. dag.
- Ændringerne gemmes og opdateres med det samme i systemet.

Formålet med at anvende disse use cases er at klarlægge, hvordan systemet skal fungere fra både beboerens og administratorens perspektiv.

For “Beboer ønsker at booke lokale”, er formålet at vise, hvordan en almindelig beboer interagerer med bookingportalen for at reservere et lokale. Ved at vise denne proces, får man et indblik i beboerens oplevelse, herunder hvordan de navigerer gennem systemet, vælger en passende tid og lokation, og bekræfter deres reservation.

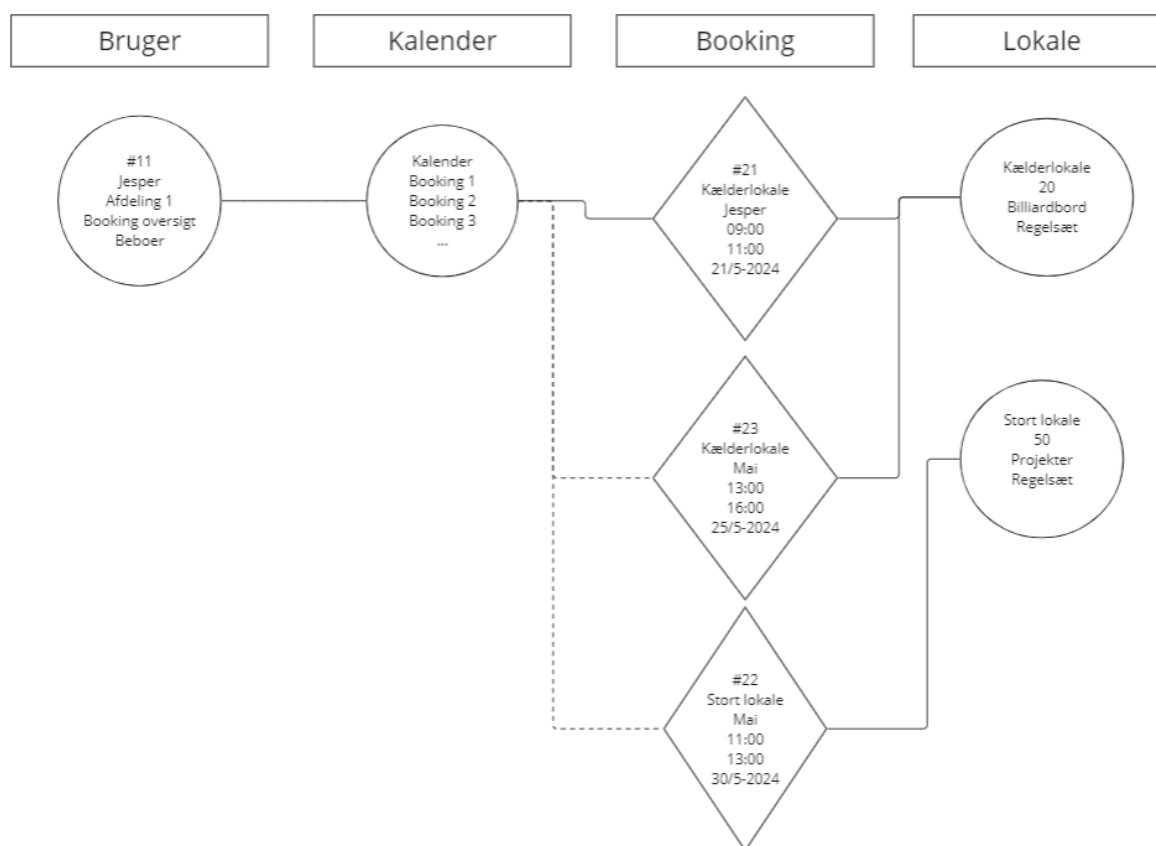
For “Administrator skal redigere regelsæt for lokaler”, er formålet at vise, hvordan en administrator har mulighed for at administrere og tilpasse systemets indstillinger og regler for lokalerne. Dette inkluderer ændringer af tilgængelighedstider, begrænsninger for antal bookinger og andre relevante indstillinger. Her får man en forståelse af, hvordan systemet administreres og vedligeholdes.

Samlet set får man ved hjælp af disse use cases et indblik i, hvordan systemet opfylder både beboerens og administratorens behov, og hvordan det letter bookingprocessen og administrationen af lokalerne på en effektiv og brugervenlig måde.

Objektmodel

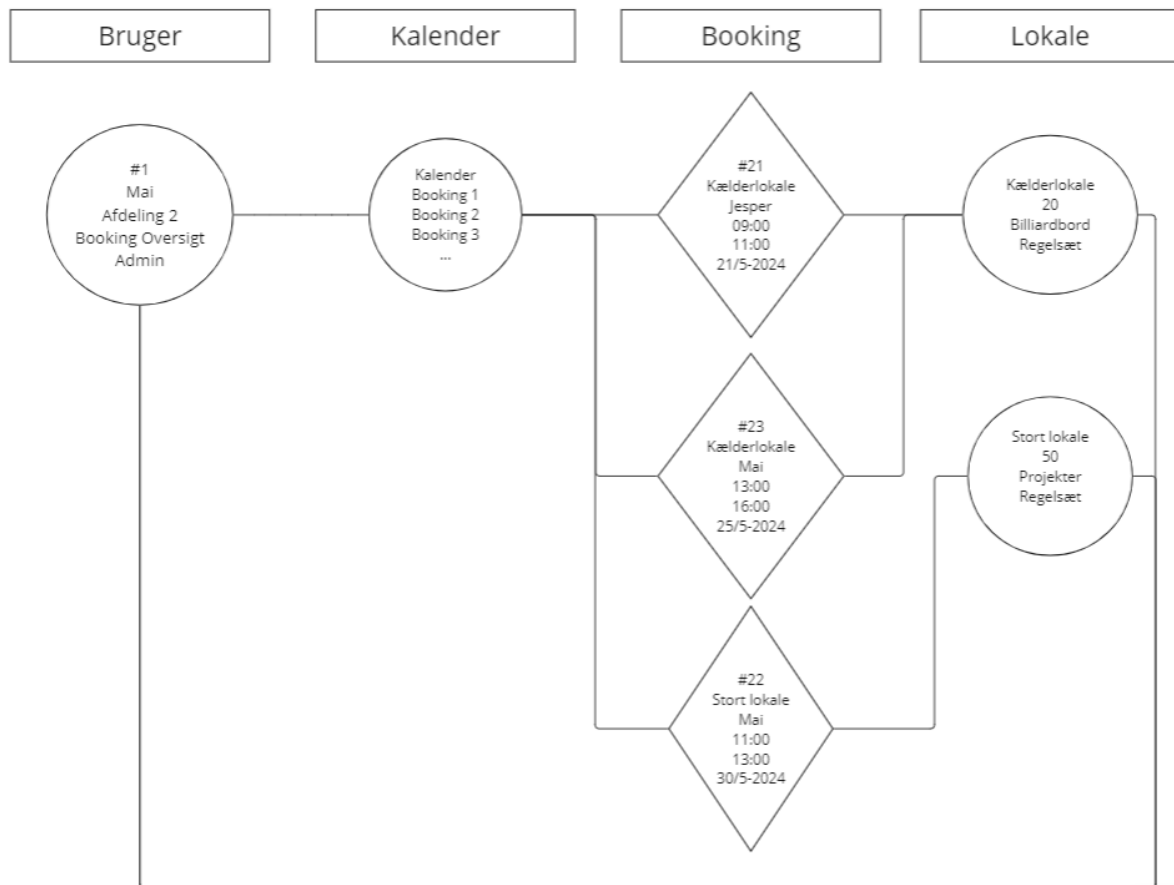
Vores objektmodel repræsenterer fire konceptuelle klasser: Brugere, Kalender, Bookinger og Lokaler. Bruger-klassen repræsenterer både almindelige beboere og administratorer i system. Disse brugere har forskellige rettigheder og adgangsniveauer afhængigt af deres rolle. Kalender-klassen fungerer som en central hub, der indeholder alle bookinger i systemet. Denne klasse muliggør effektiv styring og organisering af bookinger på tværs af forskellige brugere. Booking- og Lokale-klassen repræsenterer hhv. bookingerne og lokalerne i systemet.

En bruger kan lave en booking (samlende objekt) gennem kalenderen, dette uanset om de er almindelige beboere eller administratorer. Derudover har hver booking et lokale tilknyttet.



Figur 1 - Objektmodel for beboers brug af systemet Kilde: egen udarbejdelse

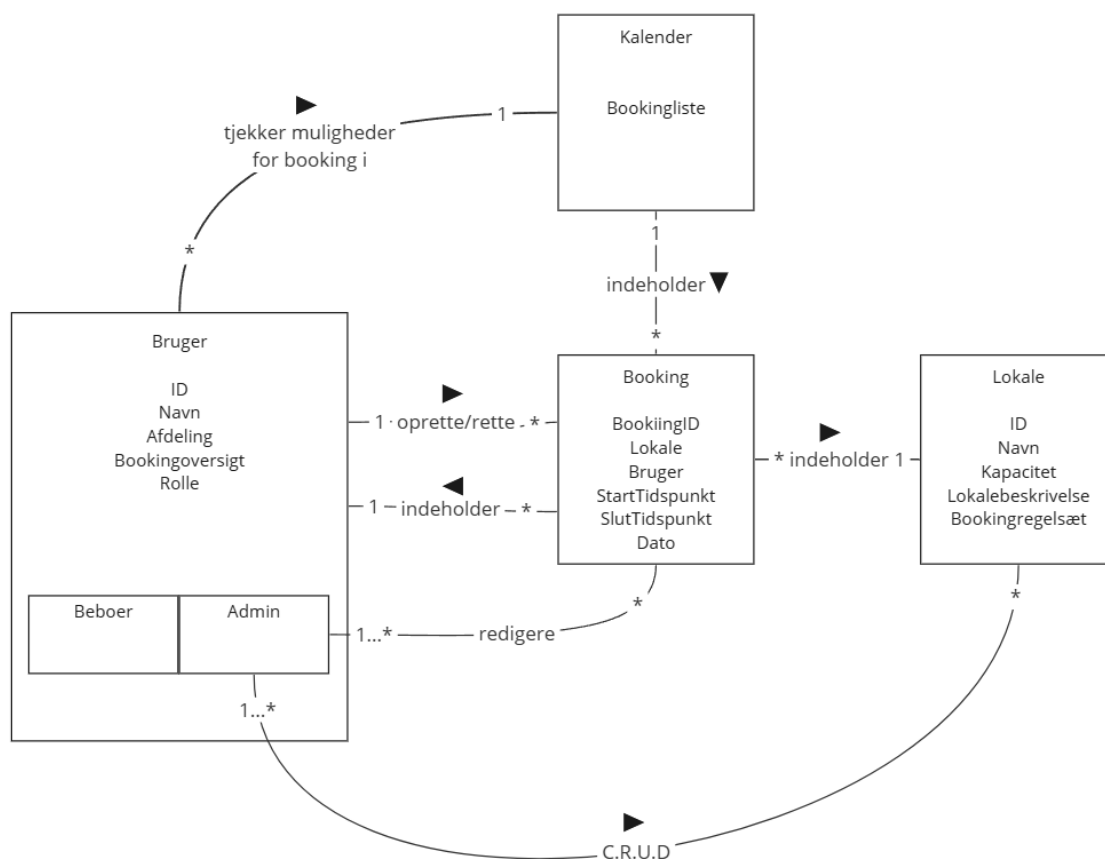
En administrator har alle de samme funktionaliteter som en bruger og har derudover adgang til alle bookinger i systemet, med det formål at kunne slette bookinger på brugeres vegne, samt ændre i lokaleoplysninger. Administratorer har desuden adgang til alle bookinger i systemet.



Figur 2 - Objektmodel for administratorens brug af systemet Kilde: egen udarbejdelse

Domænemodel

Vi har taget udgangspunkt i vores objektmodeller og samlet dem til en domænemodel, som tydeliggør forbindelserne i systemet. Vær opmærksom på at beboer og administrator begge er brugere, men med forskellige adgangsniveauer og muligheder i systemet, hvilket vi repræsenterer ved de mindre bokse inde i bruger klassen. CRUD-akronymet står for “create, read, update, delete” og er alment brugt inden for faget.

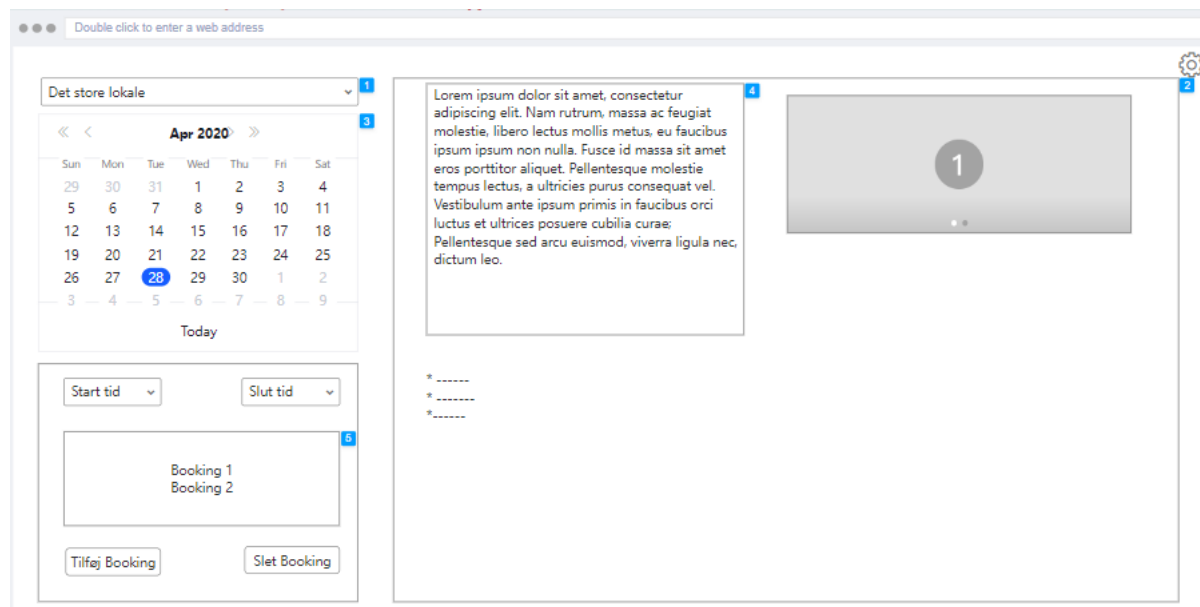


miro

Figur 3 - Domænemodel Kilde: egen udarbejdelse

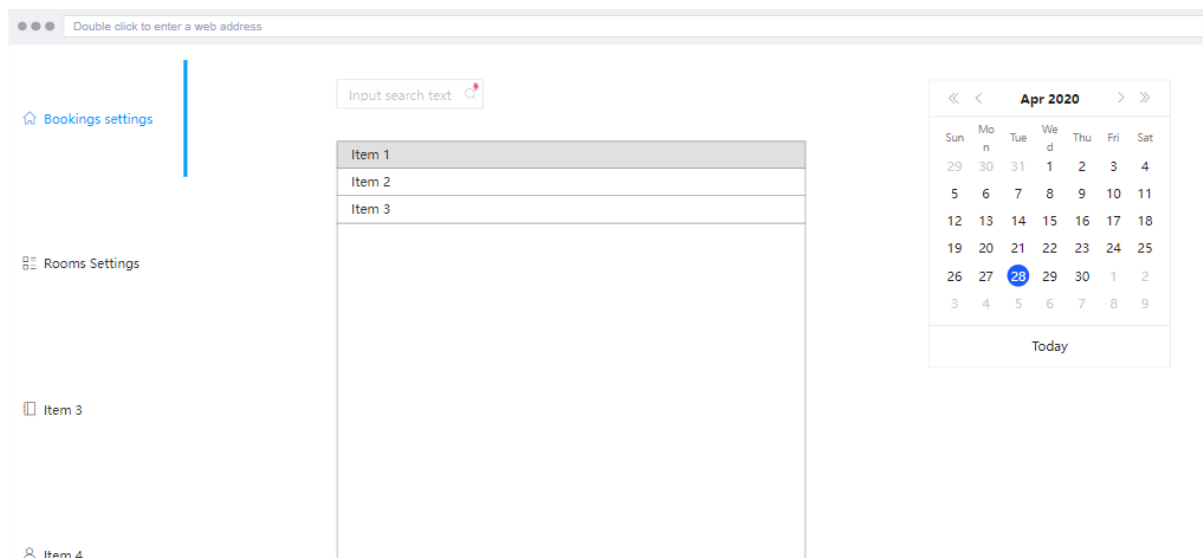
Wireframes

Vi har opdelt vores bookingsystem i tre forskellige 'views'. I det første view, som illustreres i figur 4, kan beboere i boligforeningen foretage en booking af et lokale. Beboeren får mulighed for at vælge et ønsket lokale, hvor der står en beskrivelse samt et billede af lokalet til højre. Derefter kan beboeren trykke på en dato i kalenderen og vælge start- og sluttidspunkt for bookingen. Når dette er gjort, kan beboeren bekræfte bookingen ved at trykke på knappen 'Tilføj Booking'. Den nyoprettede booking kan nu ses nederst til venstre. Desuden er det også muligt for beboeren at slette en eksisterende booking ved at trykke på knappen 'Slet Booking'.



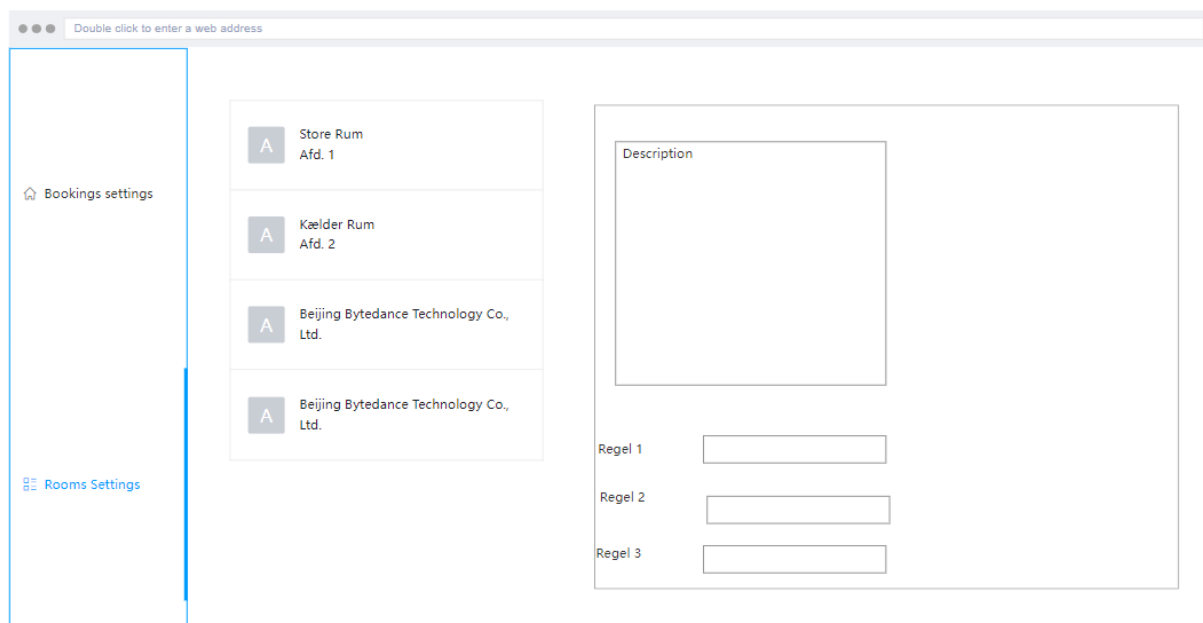
Figur 4 - Wireframe for booking af lokale Kilde: egen udarbejdelse

I det første af to administrator-views, som illustreres i figur 5, får administratorer et overblik over alle bookings på tværs af beboere. Bookinger bliver vist midt på siden, hvor administratorer kan se detaljer om lokale, dato, start- og sluttidspunkt. Til højre på siden er der en kalender, hvor administrator kan vælge en specifik dato for at filtrere bookinglisten. Derudover er der et tekstfelt ovenover bookinglisten, hvor der kan søges på andre kriterier for at finde specifikke bookinger. På venstre side af visningen er der mulighed for at skifte mellem forskellige administrator-views, som vist i figur 6



Figur 5 - Wireframe for administratorens oversigt over bookinger Kilde: egen udarbejdelse

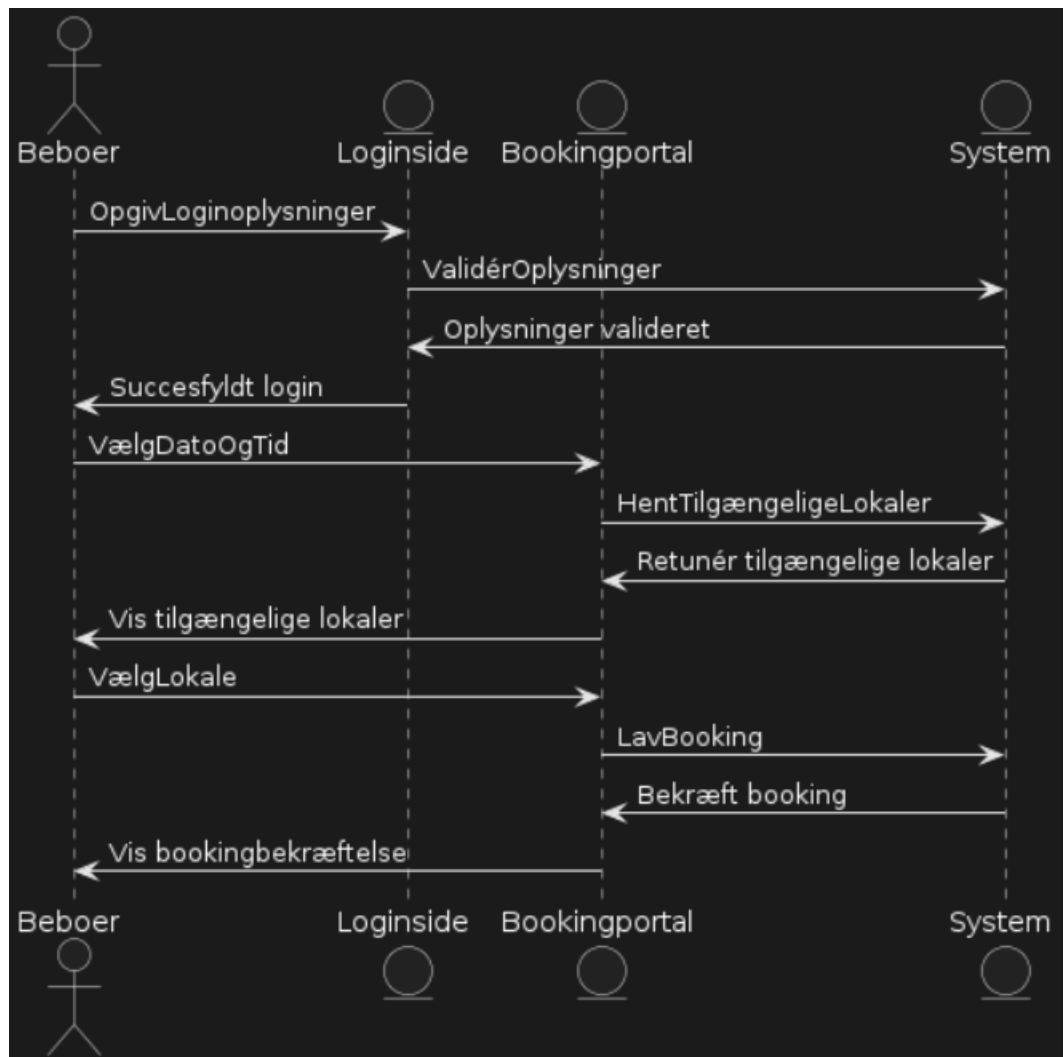
I det andet administrator-view, vist i figur 6, får administratorer en oversigt over alle lokaler i systemet. Her kan lokaler tilføjes, redigeres og slettes efter behov. Når en administrator vælger et specifikt lokale ved at trykke på det, vises detaljer om lokalet til højre, hvor de kan foretage ændringer. Ligesom på figur 5 kan man yderst til venstre skifte mellem administrator-views. Dette giver administratorer mulighed for nemt at navigere mellem forskellige administrative views og opnå det ønskede overblik over lokalesystemet.



Figur 6 - Wireframe for administratorens oversigt over Lokaler Kilde: egen udarbejdelse

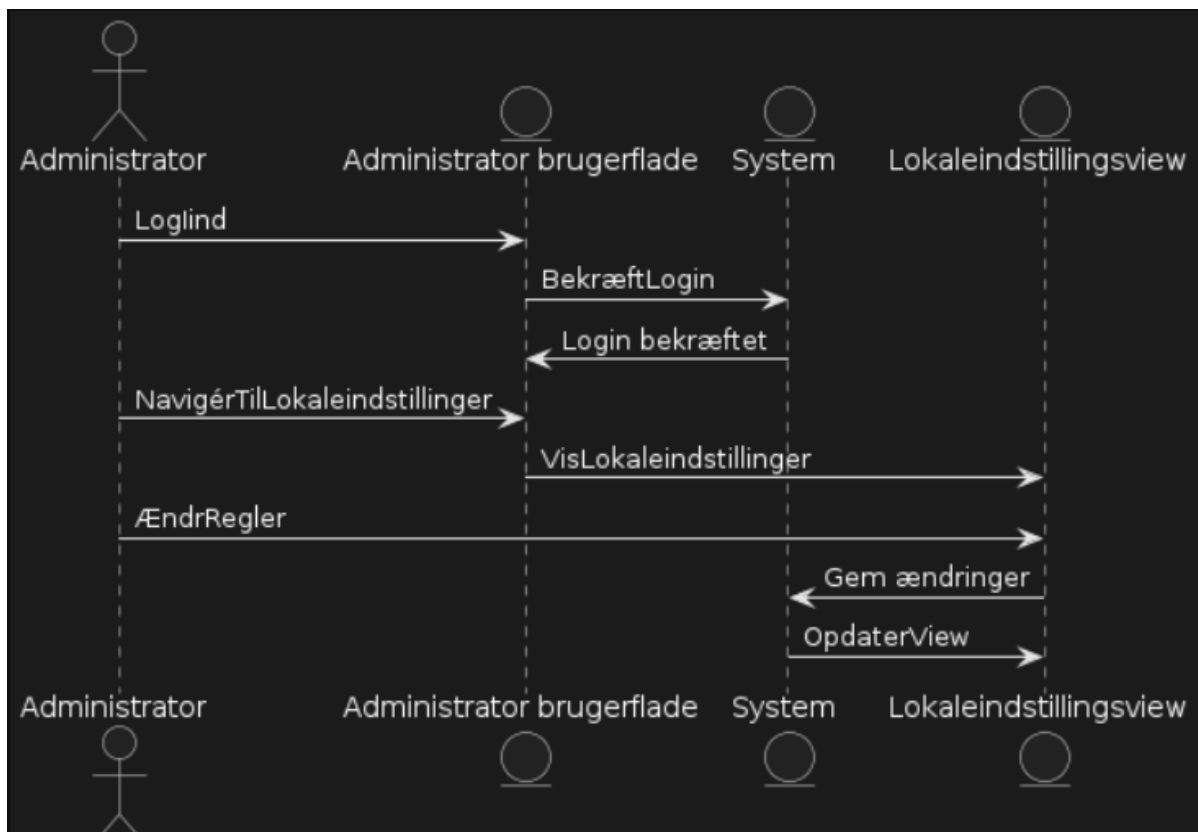
System Sequence Diagram

Diagrammerne og de tilhørende operationskontrakter tager udgangspunkt i vores Use Cases (side 7).



Figur 7 – sekvens diagram for bruger interaktion med systemet, Kilde: egen udarbejdelse

Operation:	LavBooking(lokal: Room, bruger: User, dato&tid: DateTime)
Cross References:	Lav booking
Pre-conditions:	<ul style="list-style-type: none"> - Lokale findes - Bruger(beboer) findes
Post-conditions:	<ul style="list-style-type: none"> - Et bookingobjekt med dato, tid og lokale blev oprettet (instance creation) - Bookingobjektet blev associeret til brugerobjektet (association formed) - Bookingobjektet blev associeret til lokaleobjektet (association formed)



Figur 8 – sekvens diagram for administrator interaktion med systemet, Kilde: egen udarbejdelse

Operation:	ændreLokaleRegler(lokal: Room, bruger: User)
Cross	
References:	Ændre Lokale Regler
Pre-conditions:	<ul style="list-style-type: none"> - Lokale findes - Bruger(administrator) findes
Post-conditions:	<ul style="list-style-type: none"> - Lokale information blev hentet fra database (persistence used) - Ændrede regler for lokalet blev gemt (persistence used)

Low Level Design

Design Class Diagram

DCD-modellen kan findes i bilag 1, følgende er beskrivelser af elementer fra modellen:

AdminBookingView-, AdminRoomView- og CalendarView er brugergrænsefladerne. De giver en visuel grænseflade for brugeren til at interagere med.

UserViewModel- BookingViewModel- og RoomViewmodel-klasserne er de klasser der udvælger og 'oversætter' data mellem Models- og View-laget om hhv. brugere, bookinger og lokaler. De øvrige ViewModel-klasser håndterer logikken og dataene for de respektive views.

User-klassen repræsenterer en bruger i systemet. En bruger kan være en beboer, der ønsker at lave en booking, eller en administrator, der håndterer bookinger og lokaler. Brugeren har ansvaret for at interagere med systemet, som at lave nye bookinger, se eksisterende bookinger og annullere bookinger.

Booking-klassen repræsenterer en reservation af et lokale i systemet. En booking inkluderer oplysninger om hvilket lokale der er booket, hvem der har booket det, og hvornår bookingen finder sted. Booking-klassen holder altså styr på alle nødvendige oplysninger for hver reservation, så systemet kan administrere og vise disse informationer korrekt. Det gør det muligt for brugeren at se sine reserverede tider og detaljer, samt for administratorer at administrere disse bookinger.

Room-klassen repræsenterer et lokale, der kan bookes i systemet. Hvert lokale har specifikke egenskaber, der gør det unikt, såsom id, navn, kapacitet og beskrivelse/faciliterer. Klassen gør det muligt at holde styr på hvilke lokaler der er tilgængelige og deres kapacitet. Dette hjælper brugeren med at vælge det rigtige lokale til deres behov.

BookingRules-klassen definerer reglerne og begrænsningerne for bookinger, på et bestemt lokale, i systemet. Dette inkluderer tidsrummet hvori lokalet kan bookes og maksimum antal bookinger per beboer. Klassen sikrer, at alle bookinger overholder de fastsatte regler og begrænsninger.

RepositorySelector-klassen fungerer som hjælpeklasse til at vælge og returnere de rette repository-instanser.

RepositoryTextFile-klasserne implementerer interfacet IRepository og håndterer alle CRUD-operationer for de respektive domæneklasser.

Persistence-klassen håndterer læsning og skrivning af data til og fra tekstfiler. Den fungerer som en hjælpeklasse for at gemme og hente data fra databasen.

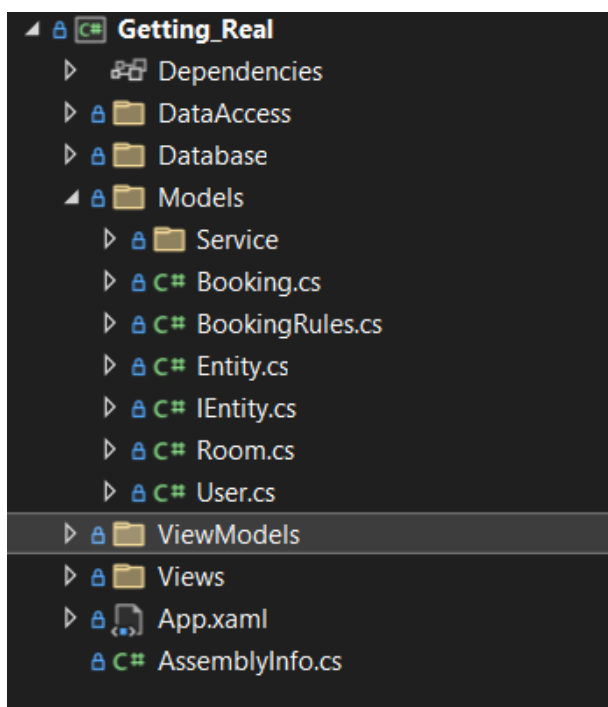
Nederst i strukturen findes vores database, som består af tekstfiler, hvori informationen om vores Model-objekter gemmes til og indlæses fra.

Resultat og implementering

Kode-struktur

I og med at vores produkt er et bookingsystem, skal programmet både kunne tilgås af brugere og gemme bestemte informationer persistent. Derfor er det nødvendigt med kode, der håndterer en UI-del af programmet, ligesom det er nødvendigt med en del, der håndterer persistens. I vores prototype til programmet, besluttede vi at bruge WPF til UI og tekstfiler til persistens, da det er hvad vi har været omkring på uddannelsen (hvis man ser bort fra muligheden for et konsol-interface). Hvis applikationen nogensinde skal tages i brug i virkeligheden, vil både UI-formatet og persistenstypen dog efter alt at dømme skulle udskiftes, og vi har således prioriteret en modulær kode-struktur med så løs en kobling lagene imellem som muligt for at lette en evt. fremtidig udskiftning af UI/persistens-lag.

I vores WPF-implementering har vi benyttet os af MVVM-mønstret, hvor vi har et Views-lag, med XAML- og code-behind-filer, et Models-lag med vores domænerelaterede klasser og "business logic" og sidst et ViewModels-lag, der står for at strukturere vores Models-objekter til brug i UI-laget. Således bliver Models-laget ikke direkte påvirket af en evt. udskiftning af Views-laget.



Figur 9 - Oversigt over filstruktur Kilde: egen udarbejdelse

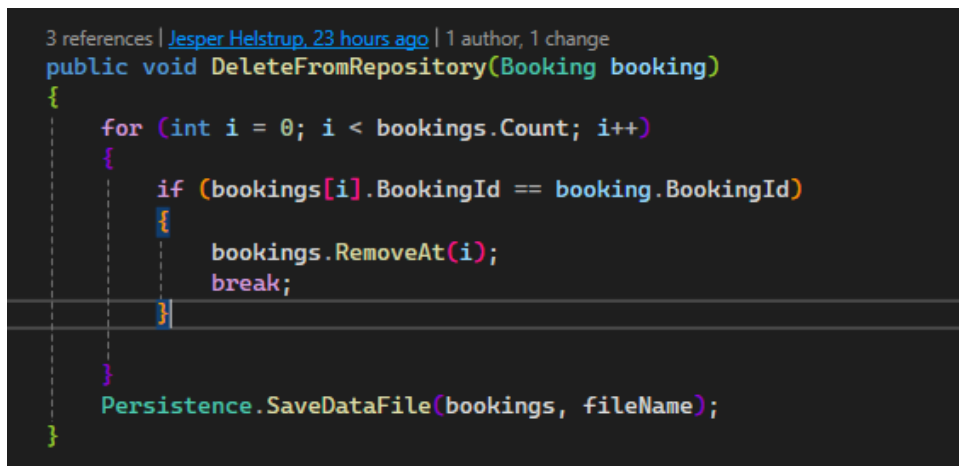
Ifm. persistens-implementeringen har vi hentet inspiration fra Repository-mønstret. Vi har således et lag mellem vores database og vores Models-lag, DataAccess-laget, som henter information fra tekstfilerne og på baggrund af disse data opretter Models-objekter. På denne måde bliver Models-laget "*persistent mechanism ignorant*" (Shukla, 2015): Models-laget kommunikerer aldrig direkte med databasen, men altid igennem repositories, hvilket sikrer, at vi kan udskifte database-typen uden at påvirke Models-lag-kode.

Denne struktur gør vores kode modulær og skaber løs kobling imellem lagene. Ifm. netop koblingen mellem lagene løb vi ind i et problem, da det er nødvendigt for vores ViewModels-lag at kommunikere med vores repositories, når der skal hentes eller gemmes data til og fra UI-laget. Dette løste vi med klassen 'RepositorySelector', som ligger i en Service-mappe/namespace indeni Models-laget. Når ViewModels-laget således vil kommunikere med et repository, går den igennem denne klasse, som så kommunikerer videre ned til DataAccess-laget, og der er således ikke en direkte kobling mellem ViewModels- og DataAccess-laget.

Kode-eksempler

Vi har udvalgt enkelte metoder fra vores kode for at illustrere vores brug af forskellige centrale metoder og redskaber fra undervisningen.

I DeleteFromRepository-metoden fra BookingRepositoryTextFile-klassen sletter vi en booking fra vores liste i repositoryet og opdaterer vores database (txt-fil).



```
3 references | Jesper Helstrup, 23 hours ago | 1 author, 1 change
public void DeleteFromRepository(Booking booking)
{
    for (int i = 0; i < bookings.Count; i++)
    {
        if (bookings[i].BookingId == booking.BookingId)
        {
            bookings.RemoveAt(i);
            break;
        }
    }
    Persistence.SaveDataFile(bookings, fileName);
}
```

Figur 10 – kode-snippet for DeleteFromRepository metoden i BookingRepositoryTextFile klassen, Kilde: egen udarbejdelse

Her ses et eksempel på hvordan vi indlæser de data, der ligger i vores database.



```
public static string[] LoadDataFile(string fileName)
{
    string pathName = Path.Combine(targetDirectory, fileName);
    if (!File.Exists(pathName))
    {
        using StreamWriter sw = new(pathName);
    }

    using StreamReader sr = new(pathName);
    string[] lines = sr.ReadToEnd().Split(new string[] { "\r\n", "\r", "\n" }, StringSplitOptions.None);

    return lines;
}
```

Figur 11 – kode-snippet for LoadDataFile metoden i Persistence klassen, Kilde: egen udarbejdelse

Denne metode fra CalendarView code-behind er et eksempel på hvordan vi skifter mellem vores views. Det gøres ved et tryk på en knap i GUI'en.

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    AdminBookingView adminBookingView = new AdminBookingView();
    adminBookingView.Show();
    this.Close();
}
```

Figur 12 – kode-snippet for Button_Click metoden i Caldendar klassen, Kilde: egen udarbejdelse

Konklusion, diskussion og reflektion

Når vi i et projektarbejde skal samarbejde som gruppe, vil det være hensigtsmæssigt at planlægge fra begyndelsen, hvilke redskaber og teknikker der skal anvendes. Dette kan være med til at skabe en mere ensartet og effektiv arbejdsproces. Efterhånden som vi lærer flere redskaber og teknikker vil løbende revurdering dog være nødvendig. Dette afsnit er derfor hovedsageligt en refleksion over, hvad vi ville gøre anderledes, hvis vi skulle starte på et lignende projekt i dag.

Et eksempel på dette er vores arbejde med forskellige UML-modeller. Her endte vi med at bruge længere tid end nødvendigt på at udvikle detaljerede modeller, hvilket kostede os tid, vi kom til at mangle senere i processen. Dette resulterede i, at vi ikke fik implementeret alle de funktioner, vi havde sat os for. I fremtiden vil vi lægge større vægt på modellernes formål og relevans, i stedet for at vi hænger os i alle notationer og konventioner.

Der har også været en del usikkerhed om, hvordan vi bedst strukturerede vores kode, og hvordan man præcist benytter forskellige design-mønstre. Dette har medført en masse refaktorering, men har også tvunget os til at gå i dybden med mønstrene og udvikle vores forståelse på området. Om den struktur, vi er endt på, er helt optimal, er nok tvivlsomt, men ift. vores ønske om en modulær kode, der forsøger at sikre en løs kobling af lagene, opfylder vores endelige kode-struktur sådan set i tilfredsstillende grad dette ønske: Både UI-laget og databasen har kun forbundne afhængigheder til et enkelt andet lag. Havde vi haft mere tid, ville vi dog ved brug af vores interfaces have arbejdet henimod endnu lavere kobling og højere kohæsion.

Havde vi haft mere tid, ville det give mening at refaktorere koden for at udnytte funktionaliteten af vores interfaces bedre, hvilket yderligere ville løsne koblingen mellem de forskellige lag og øge den interne kohæsion i klasserne.

Ligeledes har vi pga. tidsnød to forskellige implementeringer af hvordan vores model-objekter får deres ID-numre – dette ville også kunne refaktoreres for at simplificere koden.

Vi er opmærksomme på vigtigheden af Test Driven Development, men desværre fik vi ikke implementeret dette i vores projekt. Efter færdiggørelsen af vores kode opdagede vi, at vi ikke havde nået at implementere nogen form for testafvikling. Dette er således en læring for os til fremtiden, hvor vi har til hensigt at prioritere Test Driven Development.

User-klassen er på nuværende tidspunkt kun delvist implementeret. Der er ikke knyttet nogen bruger til bookinger i systemet. Vi har antaget, at boligforeningen har et allerede implementeret system for brugere, som vores program ville 'arve', hvis det rent faktisk skulle implementeres, hvilket er en af grundene til, at vi har prioriteret andet over dette. Booking er dog sat op med en UserId-property, så det kan sættes op i systemet i fremtiden.

Litteraturliste

- Larman, C. (2004) *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. Pearson Education.
- Shukla, A. (2015) *Repository pattern in c#*, *codecompiled.com*. Tilgængelig ved: <https://codecompiled.com/?s=Repository%2Bpattern%2Bin%2Bc%23%2B2015> (Besøgt: 24 Maj 2024).
- Den røde bog

Bilag 1: DCD

