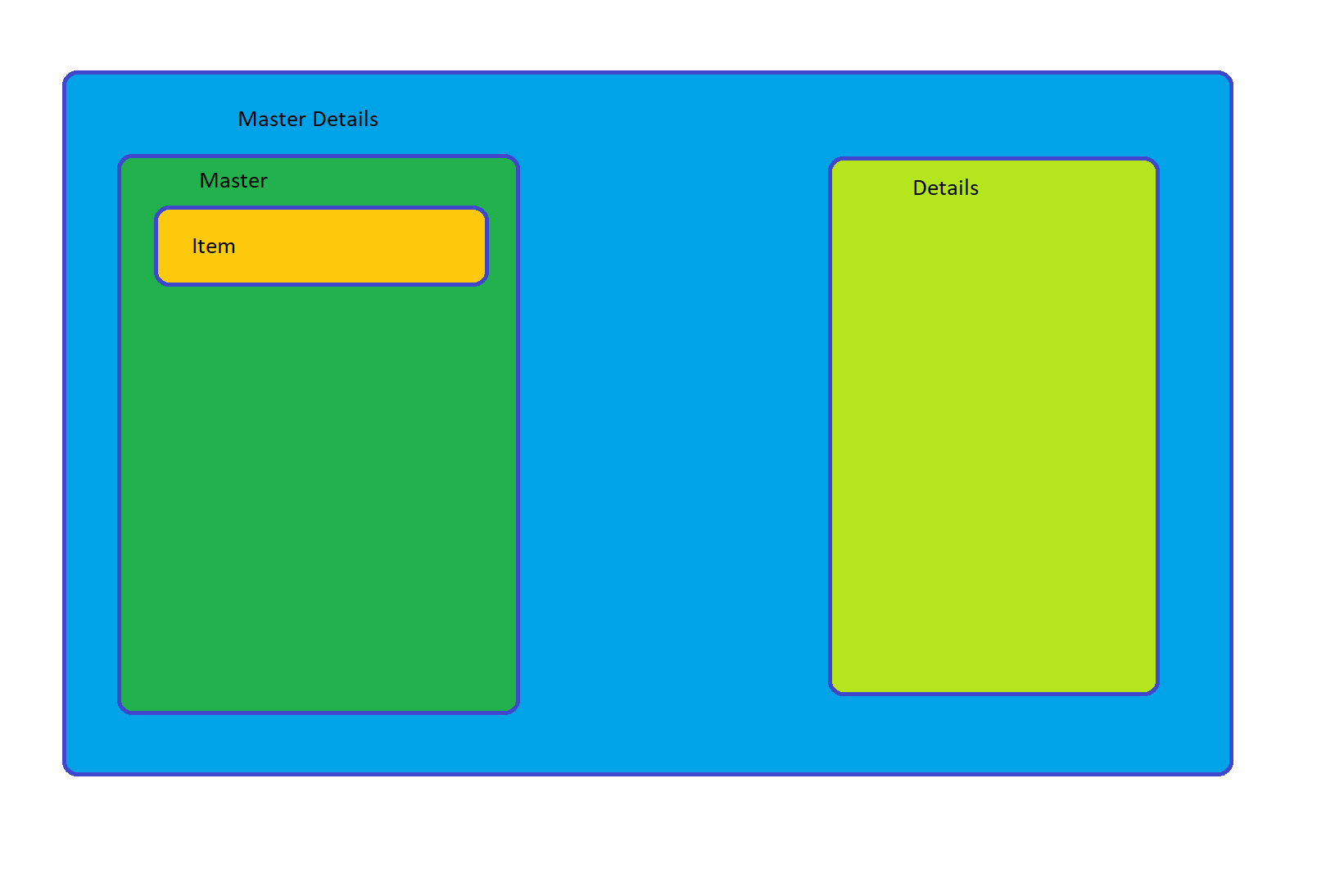
Vores udfordring med koden har ikke været domæne koden, men viewmodel koden. Da vores domæne logik er relativ simpel har vores udfordring været at få det vist i vores view på den måde vi ønsker.

Vi har hovedsageligt gjort brug af en master details model, til visning af vores data, som illusteret på figur 1.

Figur 1

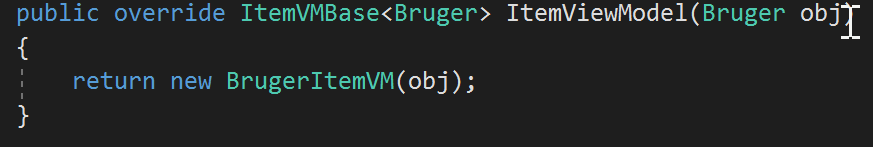
Dette bliver gjort ved hjælp af 5 klasser: Details-Klasse, Master-Klasse, Item-Klasse, Factory-Klasse og en Master-Details klasse.

Vi har forsøgt at lave vores program så generalt som muligt, inden for vores evner. Det gør sig gældende i at vi har lavet de 5 før nævnte klasser generic, dem kalder vi baseklasser, så alle klasser som skal repræsenteres som et master details view kan parametiseres i baseklasserne.

Master details forholdet bliver brugt i næsten hvert af vores views og derfor er der generet en del kode med de 5 base klasser så vi vil gennemgå dem kort herunder:

**FactoryVmBase**

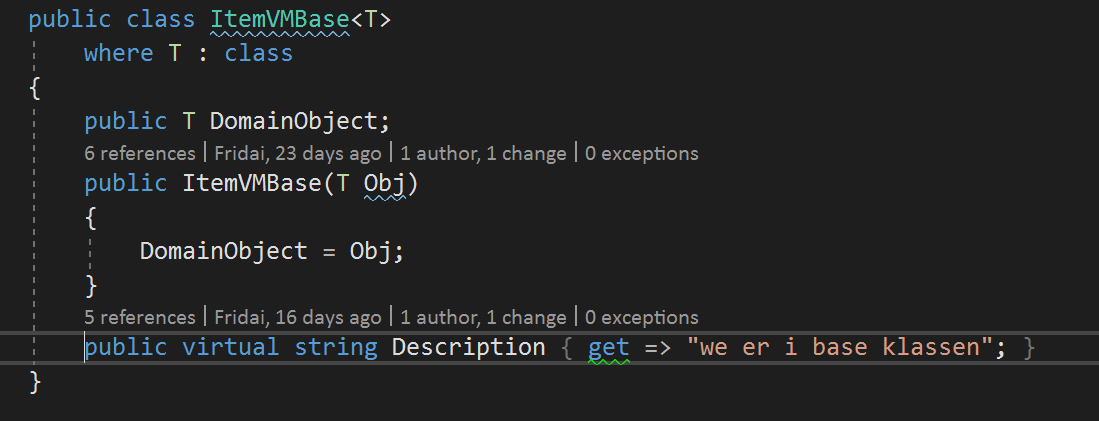
Factory klassen er har til ansvar at tage et model objekt, så som bruger og laver det til et viewmodel objekt, så som BrugerItemVm.



Figur 2

**ItemViewModelBase**

ItemViewModelBase klassen har til ansvar at skulle være logik til de enkelte objekter der bliver vist i master delen af master details, eller sagt på en anden måde er de individuelle objekter du kan se og trykke på i listviewet.



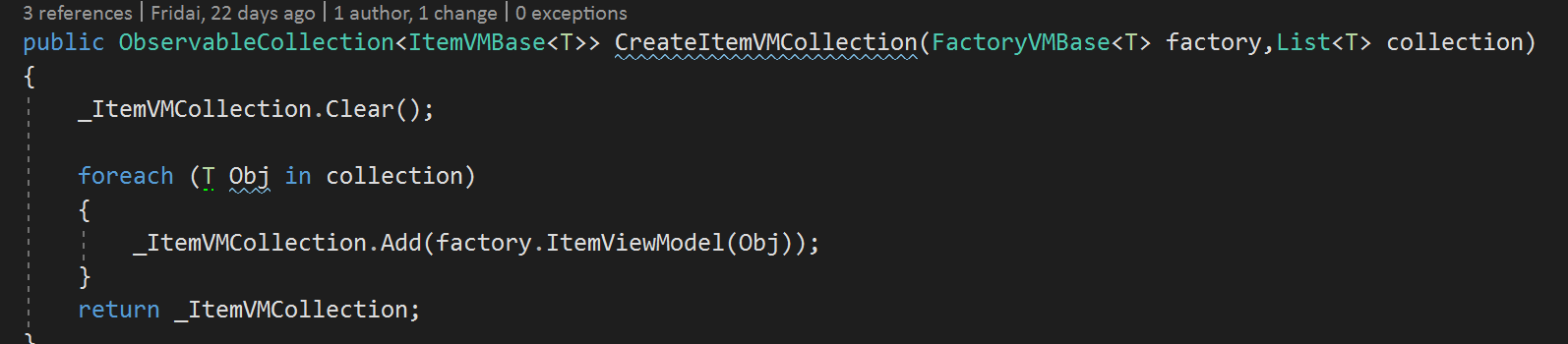
Figur 3

Klassen har en enkelt propertie, som er gjort virtuel så i den paramatiseret klasse, kan vi overwrite den med den aktuelle beskrivelse.

**MasterVmBase**

Masterviewmodelbasen har til ansvar at tage en normal liste af objekter og lave den om til et observeable liste af viewmodel objekter, vi er nød til at bruge observeablelist da en normal liste ikke kan bruges til visning i et listview, i XAML koden.

Det gøres ved at give metoden en factoryVmBase med af typen som er relevant, samt den list, man ønsker lavet om til en observeablelist.



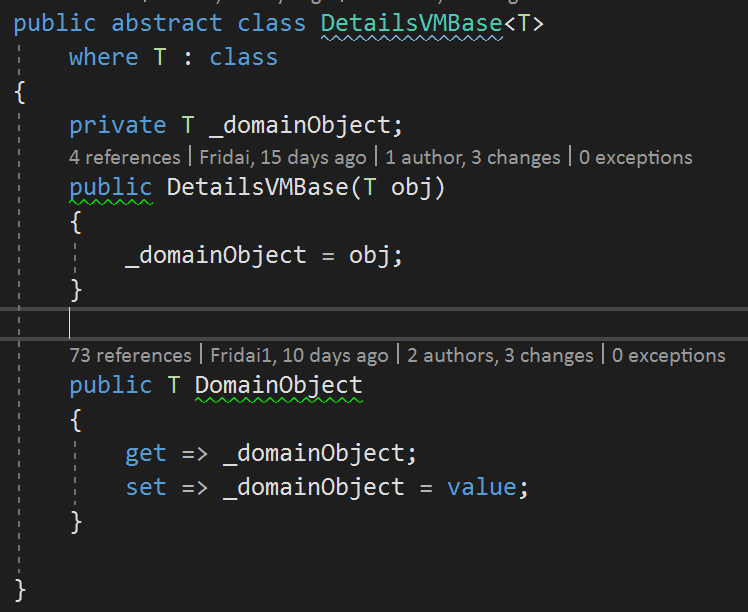
Figur 4

Vi starter med at .Clear, kollektionen da vi godt kan kalde på den samme metode flere gange og ikke ønsker kopier af det samme objekt i vores liste mere en 1 gang.

Derefter laver vi et simpelt foreeach løkke som tilføjer alle objekter fra den medgivet liste til den nye observeablekolletion, med metoden fra den medgivet factory, med det medgivet objekt som parameter.

**DetailsVmBase**

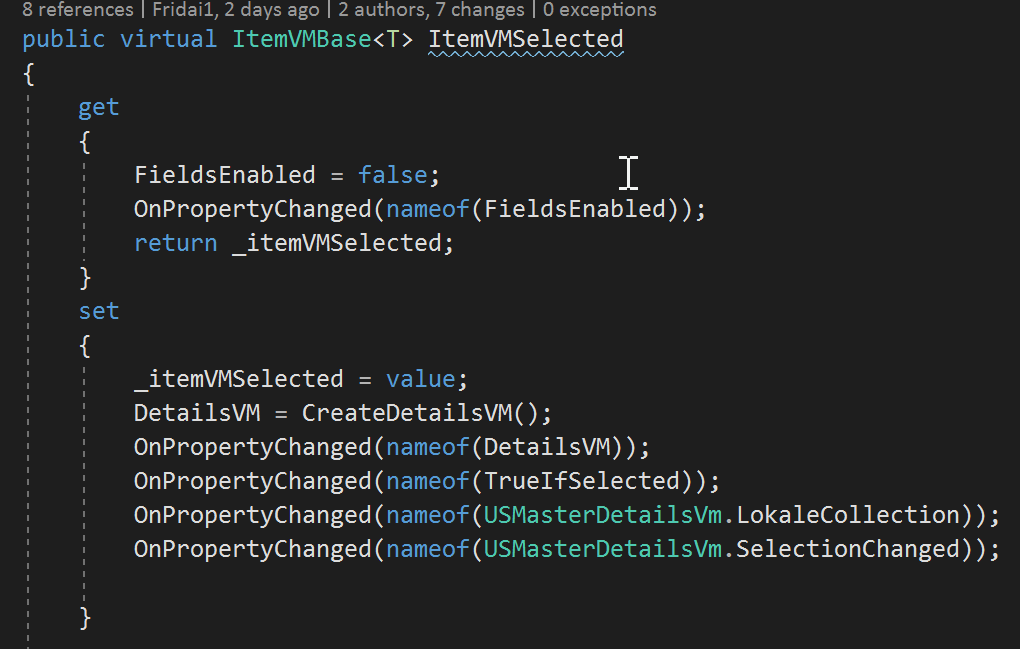
DetailsViewModelBase, har til ansvar at fungere som en mellemled mellem viewet og en model klasse, så der kan skrives og hentes fra det pågældende objekt.



Figur 5

**MasterDetailsVmBase**

Masterdetailsviewmodelbase klassen er hvor det hele bliver samlet og instanceret, samt generalt er klassen ansvarlig for alt hvad der har med master details forholdet at gøre, så som knapper og relevante metoder/properties.

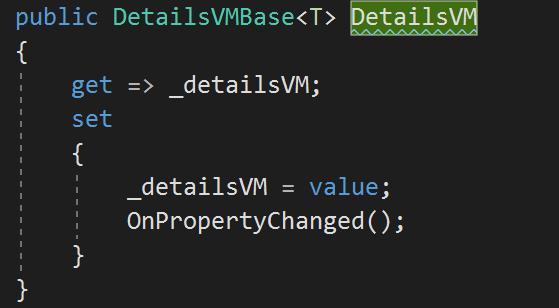
For at viewet kan fungere er vi nød til at havde nogle Properties som kan binde de før omtalte klasser sammen, vi starter med ItemVMSelected

Figur 6

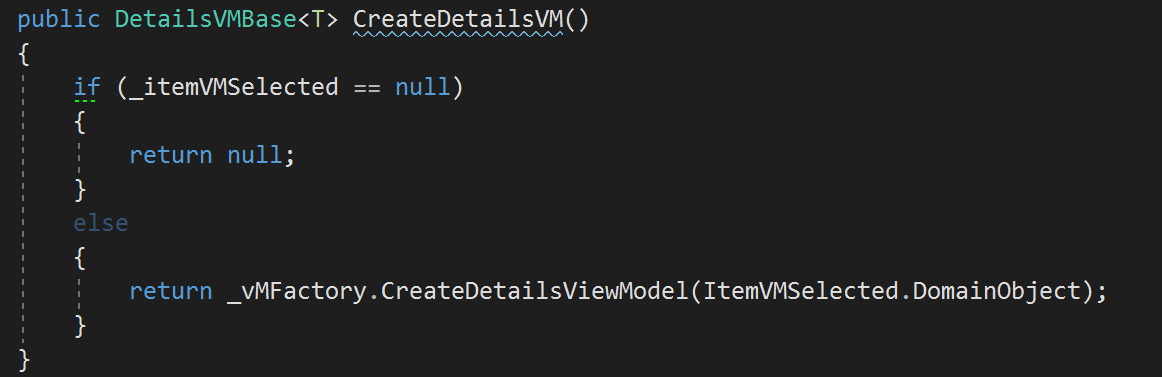
På figur 6 kan det ses at vores propertie har en Get og en Set. Når vi ber om data fra propertien, bliver en bool sat til false, denne bool er bindet til viewet og er ansvarlig for at gører felter enables eller disabled, vi ønsker ikke at brugeren kan redigere i objektet før de trykker på en rediger knap.

OnpropertyChanged bliver kaldt for at gører FieldsEnabled properien opmærksom på at den er blevet ændret.

I SET, sætter vi Instancefieldet  lig med den valgte itemviewmodel i listviewet. Der efter sætter vi DetailsVM propertien lig med metoden CreateDetailsVM().



Figur 7



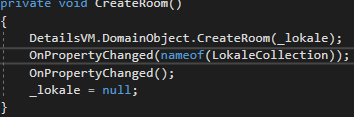
Figur 8

Så det valgte viewmodel objekt i listviewet bliver brugt til at oprette et details view ved hjælp af Factory, som set på figur 8. DetailsVM propertien bliver så bindet til UI, hvor man så er i stand til at gå ned og vælge den attribute på det valgte objekt. .

**Master detail parametiseret**

Det har været nødvendigt for os at tilføje yderligere kode til de instanceret masterdetail klasser, som er for specifict til at komme i base klassen, for eksempel har vi i undervisningsstedview, lavet et mini masterdetails, i form af at kunne tilføje lokaler til et undervisningssteds objekt. Dette kræver selfølgelig yderlige kode som er unikt til lige netop dette view.

At få det lavet var lidt af en udfordring, at finde ud af hvor koden skulle skrives henne til at starte med og hvordan det bedst kunne svarer sig. Vores tankegang var om det skulle i overview klassen eller om det hørte til i masterdetails klassen, men efter som at stadig var en del af et større master details forhold, bestluttede vi os for at skrive koden i den instanceret master details klasse.

Koden til at tilføje lokaler til en liste, viste sig at være meget lignende den måde vi havde valgt at gøre det på i baseklassen, med en metode der tilføjer den string som blev indtastet i lokale textboxen. 

For at få det vist i et listview i viewet, gjorde vi det på samme måde som i figur 4.

**OverView viewmodel**

Da der i et UI vindue også foregår mere end bare et master details forhold så som Login, har vi valgt at instancere alle relevante klasser til det enkelte view i en overview klasse, for eksempel BrugerOverview.

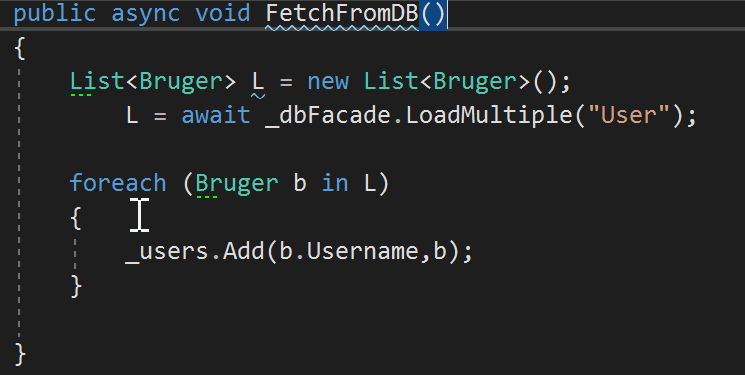
**GodBase**

Efter at havde lavet et bar OverviewVm gik det hurtigt op for os at vi gentog os selv en del med henhold til login releteret kode. For at gøre vores kode mere dry valgte vi at lave en Baseklasse til overviewet, som inholder alt Login logikken.

**LoginVM**

LoginVm klassen er ansvarlig for at styrer hvem der bliver lukket ind i systemet og gemme den bruger som bliver lukket ind i en static attribut. Vi bruger en static attribut så vi, i hver klasse kan sætte \_loggedInBruger  attributen lig den statiske attribute .

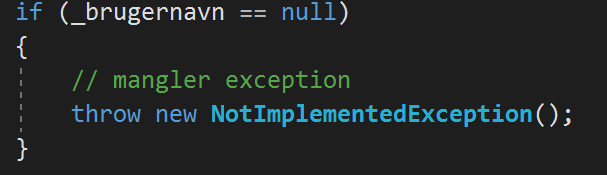
Når LoginVm bliver instanceret kører den en metode i constructoren FetchFromDB().



Som laver en ny liste og gør den lig med en liste af bruger objekter fra databasen, derefter bliver de bruger objekter tilføjet til en dictionary, med brugerens brugernavn som key og bruger objektet som value.

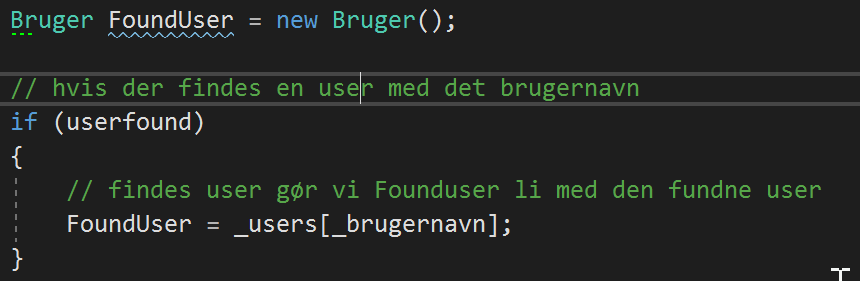
Grunde til at vi benytter en dictionary er primært pågrund af 2 årsager:

* Når der kommer mange brugere og der skal søges efter en bruger er en dictionary meget hurtigere.
* Ved at bruge en dictionary kan vi lettere udfører opgaven at søge efter de 2 atributes vi ønsker og sætte den endelige bruger lig med den statiske attribut givet oplysningerne er korrekte.

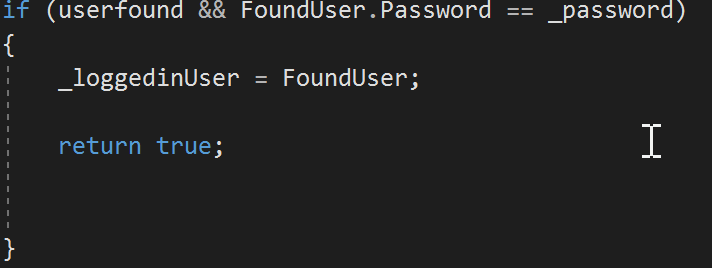
Selve login metoden er af typen bool. Den starter med et sanity check 

Hvorefter hvis brugernavn != null, 

Ser vi om det username som er indtastet også findes i vores dictionary, vi søger efter key.

Hvis brugeren findes gør vi den fundne brugere lig med en attribute af typen bruger 

Til sidst checker vi for om den fundne brugers password stemmer overens med, det password som er blevet indtastet, er det korrekt gør vi den statiske attribute lig med den fundne bruger, og returner true.

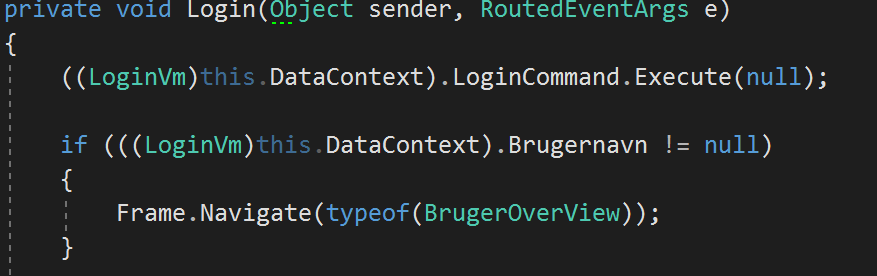


Hvis det er ikke er gældende, sætter vi brugernavn og password til null og returner false.

**Problemet:**

Problemet vi løb ind i med dette var at få det bindet til en knap og få den til at kører metoden før den sendte brugeren videre. I første omgang checkede den koden men om den så var true eller false sendte den alligevel brugeren videre, ved hjælp af Click eventen i XAML .

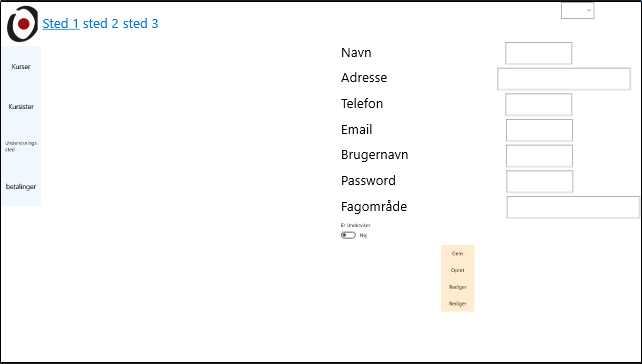
**Løsningen:**

Løsningen blev at kører commandoen fra codebehind, til click eventen.

**GUI**

Vi har i løbet af projektet haft kontakt med vores kunde (mohammed) når vi havde noget nyt at vise ham, så vi kunne se om det levet op til kundens forventninger, og hvad der skulle laves om. Vores første møde med kunde viste vi ham vores UI og login funktion samt oprettelse af en bruger. Dette var kunden ganske tilfreds med og vi fortsatte. Den næste demo gik ud på at vist kunden oprettelse af undervisningssted og tilhørende lokale, her var kunden også glad for funktionalitet men var lidt utilfreds over brugervenligheden og generalt vores GUI.

Vores første udkast af GUI:

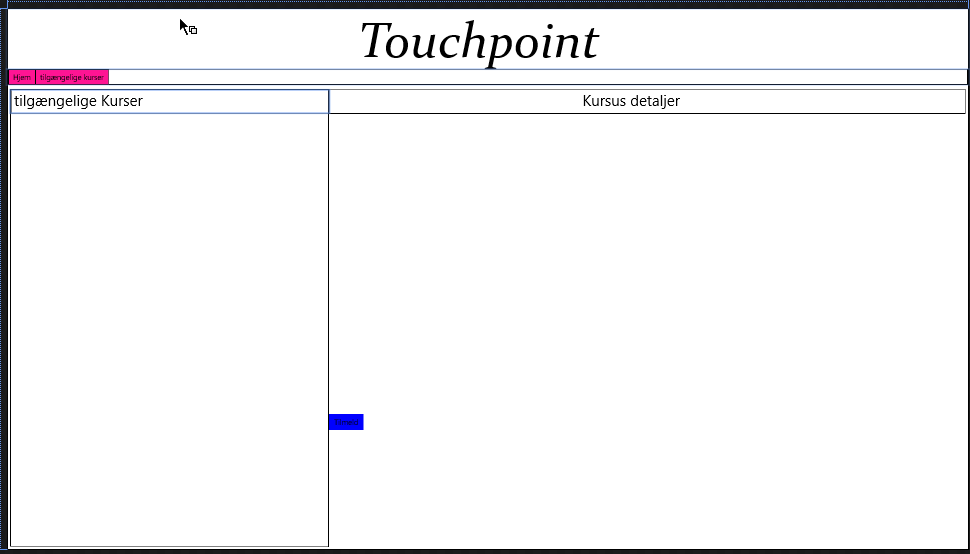


Tanken bag dette design var at havde venstre navigations menu og toppen af programmet altid være det samme, for let at kunne arbejdet ind i muskel hukommelsen hvordan man navigeret rundt som bruger.

Med et listview til venstre og details til højre.

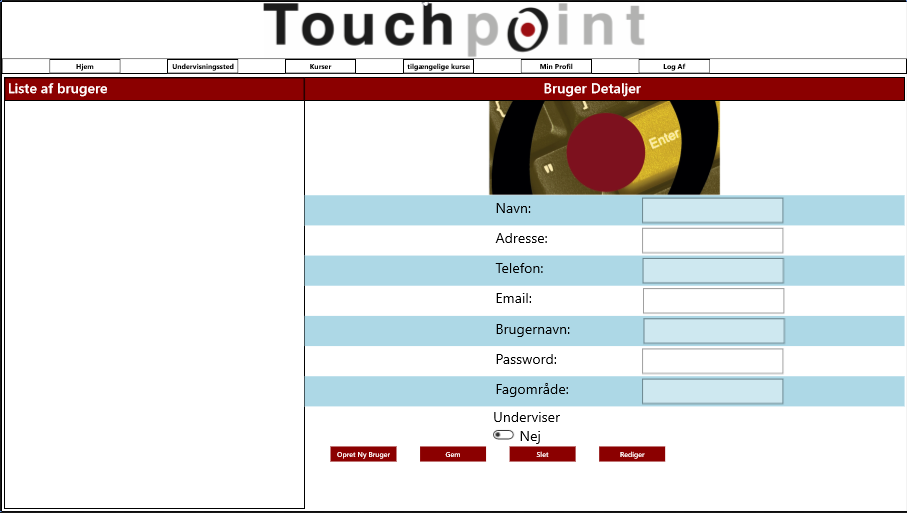
Ude i venstre side havde vi, som sagt en navigations menu, så vi kunne komme rundt til de andre views. I toppen havde vi planlagt en form for stifinder, så jo længere ind i programmet man kom, ville de opdatere sig. Så for eksempel, klikkede man på kurser, så ville sted 1 vise kurser, derefter trykkede man på opret kursus og blev taget til opret kursus view, og sted 2 vil nu skrive opret kursus, så var tanken det skulle kunne fungere som knapper så man kunne gå tilbage til et tidligere skridt nemt, som var det en stifinder i Windows.

Andet udkast:



Her er et første udkast af det nye GUI. Her har vi ændret GUI helt og starter med en stor titel af firmaets navn, herunder en menu bar for navigation rundt i programmet, vi bibeholder vores tidliger setup med et listview til venstre og detaljer til højre. Dette GUI har mere detaljeret beskrivelse af hvad du kigger på, og klar afgrænsning af hvor grænserne af hvert GUI element starter og stopper i form af sorte linjer.

endelige udkast:



Her har vi igen taget udgangspunkt i Touchpoints originale farver, deres mørke røde og brugt det farve tema, til at bringe mere liv i vores GUI. Vi har tilføjet en baggrunds farve til de stackpanels, som holder detaljerne, for lettere at kunne adskille dem fra hinanden. Vi er meget tilfredse med hvordan vore endelig GUI kom til at se ud.

**Del konklusion og reflektion:**

Vi synes at model lags koden var relativ simpel at skrive, det var nødvendigt at lave lidt ændringer i designet, som det skred fremad, men det er jo pointen med Unified process. Nævn ligt at vi havde glemt at tage højde for nogle metoder i kursus og undervisningssteds klasserne. Udover det var der ikke andre vanskeligheder med model laget. Vores problemer kom først rigtigt på spil da vi skulle skrive viewmodellerne, at få det hele til at snakke sammen, som vi havde forstillet os under designet viste sig at tage meget længere tid end forventet, og er desværre årsagen til at vi har måtte unscope lidt i projektet. Dvs. at vi har valgt ikke at lave iteration 3, for at få mere tid til at fikse de problemer vi havde med viewmodellen.

Da vi startede projektet var vi alle enige om at vi ville prøve at skrive det så smart, som vi var i stand til med vores nuværende erfaring, det betød blandt andet at lave database klasserne generic for at gøre programmet mere dry, dette havde vi regnet ville tage 1 uges tid, men ente med at tage 2-3 uger. Hvilke betød, at når viewmodellerne skulle kodes, blev der taget udgangspunkt i lokale kollektionerne og ikke database forbindelsen. Tanken bag de lokale kollektioner var dog at de skulle kundes sættes lig med en liste fra databasen med objekter der fra. Dette program er ikke et stort program, men er trods alt det største program vi har lavet til dato. C# og Xaml, virker som et godt værktøj til at skrive programmer i denne størrelse, med mere erfaring og bedre koordination er vi sikker på at vi ville kunne havde færdig gjort alle projektets iterationer.