# Beskriv kort systemets funktionalitet:

Det skal kunne indsætte debtors til en liste.

# Arkitektur overvejelser:

Funktionelle og ikke-funktionelle krav der supporterer dine designvalg

Overall- systemarkitektur og hvilke patterns der er brugt (vis diagrammer)

# Vis design og layout af GUI

Hvordan den bruges, og hvordan en bruger skal benytte systemet

# Tanker/problemer/erfaringer fra implementering

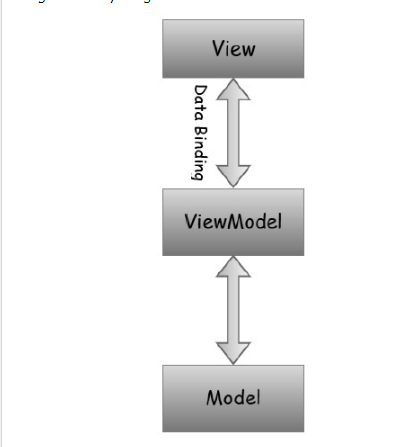
Problem med at opdatere korrekt i de forskellige views. Det er løst med at…

# 

# MVVM in WPF

De største fordele ved MVVM er separation af applikation logic og presentation layer, altså separation mellem Model og Views. Dette medfører, at vi har mulighed for at ændre funktioner i model, uden der skal ændres i views, og vice-versa. Derudover bliver de forskellige komponenter også genbrugelige, fordi at de ikke afhænger af andre. Dvs. at en model eksempelvis kan bruges af flere forskellige viewmodels.

Hvis man har en mindre applikation kan det dog nemt være overarbejde at skulle lave en MVVM arkitektur i sit projekt, fordi at det er omfattende. Hvis funktionaliteten er simpel vil det nødvendigvis ikke være en fordel at bruge MVVM.



Viewmodel bliver brugt som linket mellem model og view.

Det giver mening for vores system at bruge MVVM arkitekturen, da vi har flere views som skal bruge den samme model. Eksempelvis kan man i mainview tilføje en ny ”debtor” eller man kan se historikken for en eksisterende ”debtor” og tilføje debts. De 2 beskrevne views har forskellige viewmodeller, men begge viewmodeller benytter den samme model til at opdatere og indsætte data.

I applikationen er der lavet et nyt view til hver vindue, og hvert view har en tilhørende viewmodel.

Alle viewmodeller i applikationen benytter de samme models, hvilket er en fordel ved MVVM arkitekturen.

Man kunne fordelsvist have lavet en ViewModelLocator klasse, til at styre hvilke views, der skal benytte hvilke viewmodeller. Dog sættes vores view modeller i applikationen enten i XAML eller i code behind.