# 5. MVC & MVP Pattern

1. **Redegør for hvad software design pattern er**
2. **Redegør for i MVC Pattern og dets variationer**
   1. Forsøger at adskille GUI logik fra Forretningslogik
      1. Opdeler kode med forskelligt ansvar
   2. Model:
      1. Forretningslogikken
      2. Afspejler databasen
      3. Benytter dataannotationer som kan valideres i controller
      4. Fjerner unødvendig forretningslogik fra controllere
      5. Ændringer i Model notificeres til modellens View
      6. Frakoblet fra GUI, fremstiller kun data
         1. kan derfor bruges flere steder
   3. Controllers
      1. Beslutter hvilke ændringer der skal ske i Modellen
      2. Ved et tryk på en knap, er det controlleren der fortæller View hvad der skal vises
         1. fortæller også Modellen hvad der skal ske
      3. bruger sender et request fra view, controller tager inputtet og sender det til Modellen
         1. hvis modellen godkender input, sendes det til controller og videre til view
   4. View
      1. Præsentere Modellen for en bruger
      2. Kan være implementeret som f.eks. et Strategy Pattern
         1. Controlleren giver viewet Strategien.
      3. Kan være implementeret som et Composite Pattern
         1. kan bestå af flere elementer, ”Composites” indeholder ”leafs” eller andre ”Composites”
   5. Fordele/ulemper ved MVC
      1. + God til at vise en Model på flere måder
      2. + Ingen Business Logic i UI
      3. + Nem at Unit Teste
      4. – Ikke skalerbart, separere UI men ikke Model
      5. – Controllere bliver ofte store (uoverskuelige)
      6. – Overholder ikke Single Responsibility og interface Segregation principperne
3. Redegør for Model-View-Presenter og dets variationer
   1. MVP er en udvikling af MVC
   2. View får et interface som presenter benytter sig
      1. Øger testbarheden af presenter
   3. Model og View fungerer som i MVC
   4. Controller bliver en Presenter (her ligger forskellen fra MVC)
      1. Indeholder krævende business logic
   5. View indeholder kun lige præcis nok logik til at kalde funktionaliteten fra presenteren
   6. Sørger for at modtage og videgiver data korrekt fra model til view
   7. To typer:
   8. Passiv view
      1. View kender ikke til model
      2. View har ingen logik
      3. Ingen **code behind**
      4. Al data binding skal skrives selv
   9. Supervising Presenter
      1. Viewet kender til modellen (modellen kan ikke opdatere view)
      2. Presenteren interagere med Viewet med den opdaterede Model
   10. Se Eksempler i disposition
   11. For passivt view gælder at:
       1. View
          1. Består kun af en række kontroller der viser modellen, brugerinteraktion går fra view til presenter
       2. Model
          1. Er repræsentation af data som i MVC
       3. Presenter
          1. Manipulere modellen ud fra modtagne events fra viewet, opdaterer view gennem iView, interfacet står for formatering. GUI logik
   12. For Supervising Presenter gælder
       1. View
          1. Databinding er tilladt, ellers ingen ændringer
       2. Presenter
          1. presenter kommunikerer med Modellen om hvad der skal ske i View
       3. Der er ikke meget kode i presenter, men det er sværere at teste
   13. Fordele/ulemper:
       1. + komplekse opgaver deles ud i mindre opgaver
       2. + Mindre objekter -> Færre fejl -> Nemmere at debugge
       3. + Testbart
       4. – Mange lag
       5. – ingen genbrug af model
       6. – view og presenter er bundet til en specifik Model (use case)
4. Redegør for MVVM og dets variationer
   1. Specialisering af MVP, bruges med WPF Frameworket
   2. Næsten samme opbygning som MVVM
   3. Består af følgende:
      1. View
         1. Kan skrives kun i XAML for at eliminere behovet for code behind
         2. Kender sin ViewModel gennem DataContext
      2. ViewModel
         1. Udstiller attributter i view’et
         2. INotifyPropertyChanged
         3. udstiller properties og commands
      3. Model
         1. Domæneklasser lavet ud fra kravsspecifikationen
         2. Forretninglogik
         3. INotifyPropertyChanged
   4. Se eksempel i disposition
   5. Connecting viewmodel and view
      1. To fremgangsmåder View First eller ViewModel First
      2. View First creating ViewModel in CodeBehind
         1. Fordele/Ulemper
            1. + Parametre kan sendes med I ViewModel Ctor
            2. + Logik kan laves til vælge View
            3. – Ingen dummydata i design mode
            4. – Svært at teste Code Behind
      3. View First creating ViewModel Declaratively
         1. Fordele/Ulemper
            1. + Dummy data kan benyttes i design mode
            2. + Eventhandler kan referere VievModel gennem dets navn i XAML
            3. – Ingen parametre i ViewModel Ctor
      4. View First Using a ViewModel Locator
         1. Fordele/Ulemper
            1. + Parametre I ViewModel Ctor
            2. + Logik der kan bestemme hvilken ViewModel der skal bruges
            3. + Ingen Code Behind
            4. + Dummy data
            5. – Det kan være svært at åbne et nyt view fra viewmodel og sætte den nye viewmodel til at referere data i forretnings logikken
      5. ViewModel First Datatemplate as View
         1. Fordele/ulemper
            1. + ændringer i ViewModel binder automatisk til Viewet
            2. + Parametre i ViewModel Ctor
            3. + Mulighed for logik der vælger ViewModel
            4. + Ingen Code Behind i datatemplates
            5. – ingen dummy data
            6. – ingen design for datatemplates i Visual Studio
      6. ViewModel First UserControl as View
         1. Fordele/ulemper
            1. Samme som ovenfor
            2. Undtagen at Visual Studio kan designe UserControls