# **Frederiks refleksioner over Wordleopgaven:**

## **Projektoversigt:**

WordleConsole:   
Den origninale konsol-app. Har 1 GameMaster objekt, med en række statiske hjælpeklasser der bruger statiske metoder for at fokusere på Single responsibility.

### WordleWPF:

**View:** Mainwindow er hvor fokusset ligger. WordleUserControl var WIP til refaktorering, og har ingen reel funktion nu. I sin nuværende funktion anvender alle knapper ”click” eventet. Hvis jeg havde tid, ønskede jeg at refactorere til at bruge Commands istedet.

**Views-code-behind:** Meget kodetungt. Dette lag ejer min View Model, der igen ejer min Model.

**ViewModel:** WordleViewModel i sin nuværende form tjener primært til at lade mit view-code.behind operere min model.

Som refactorering, var jeg igang med at skrive en LetterPressCommand, samt MyGuess og \_displayedGuesses2. Dette var for at gøre klassen klar til at understøtte databinding og Commands, for at følge MVVM arkitekturens principper. Disse har fungere dog ikke endnu, og har ingen funktion .

**Model:** Baseret på et Copy paste af WordleConsole klasserne, med refactorering til at understøtte den ændrede brugergrænseflade.

**Commands:** Til Refactoreringen. Her kan ses hvordan jeg var ved at uddelegere opgaver fra mit views code-behind, til Commands.

**UnitTest:** Ikke implementeret endnu.

## Processen:

Mit fokus i dette projekt var hurtigst muligt at have et funktionelt produkt klar til kunden, hvorfor vi aftalte at en konsol-applikation var startpunktet. Dette blev fokus efter det gik op for mig at mit umiddelbare valg af en MVC-Webapp med Razor var langt mere komplekst end jeg huskede, og jeg frygtede at jeg ikke havde nok tid til at sætte mig ordenlig ind i dokumentationen til arkitekturen.

Denne tilgangresulterede i en solid konsol-app der kunne genbruges i store dele, som model-laget til min fremtidige GUI

Efter konsolapplikationen var klar, gik jeg over til at lede efter en anden brugergrænseflade, der var simplere end mit første valg. Valget faldt på WPF med MVVM arkitektur, da det var hurtigt at sætte sig basalt ind i.

Igen lå fokus på brugerens oplevelse af appen, frem for at sætte sig ind i og følge arkitekturens designkonventioner.

Dette førte til en meget koblingstung arkitektur, hvor mit View ejer ViewModellen, som igen ejer Modellens primære klasse.

Testing af min MVVM blev udelukkende manuel, da jeg havde problemer med at den .NET core 6.0 version testprojektet kørte ikke ville understøtte referencer til et .NET core 6.0-windows. Anders fandt en løsning på afleveringsdagen, ved at gå ind i UnitTestens projekt-fil og manuelt omskrive <TargetFramework> til ”net6.0-windows”.

Min sidste prioritet i arbejdet blev derfor at forsøge at sætte mig ind i arkitekturen og refaktorere koden med den resterende tid i projektet.

**Erfaringer til fremtiden:**

For at undgå massive refactoreringsproblemer, ville det kunne betale sig at sætte en basal fungerende MVVM arkitektur op ved f.eks. slavisk at følge en tutorial og derefter begynde at implementere mine klasser. At gøre det den anden vej rundt efterlader for mange fejlkilder og gør refactoreringen meget mindre overskueling.

Derudover vil et tidligt tilføjet fungerende Unit-test projekt tillade, løbende at implentere tests for at undgå svært sporbare logiske fejl, som nogle af de der opstår i den nuværende implementering af mit View og dets code-behind.