# GRASP (Jacob)

## Controller (Jacob)

I GRASP design bruges controller til at håndtere den data der kommer fra UI. Det er den første klasse data møder, som er ansvarlig for at håndtere og videresende information. Et helt konkret eksempel vil være: Hvis man trykker på login, hvor bliver password og login modtaget I en teoretisk applikationen?

I en Spring applikation er det rimelig let at identificere hvilke java klasser som er controllere, fordi Spring kræver at bruger en @Controller annotation ved Controller klasser. De ligger alle sammen I samme mappe og har formodentlig controller som en del af deres navn. Det gøres så spring ved, at her er en klasse som modtager html requests. Springs UI er naturligvis alle html filerne med tilhørende css filer, og det er her controllererne får deres information fra. Informationen kan være <form> submit data eller et link til en ny html adresse.

I vores applikation har vi 5 controller klasser, hvor 1 af dem kun håndterer /error, nemlig HomeController. De andre controller klasser er CustomerController, MotorhomeController, RentMotorhomeController og ActiveMotorhomeController. De fleste af controllerklassernes funktion giver sig selv, CustomerController håndterer alle html sider hvor Customer er I fokus, fx at oprette en ny kunde I systemet. MotorhomeController håndterer siderne hvor modellen Motorhome er I fokus, fx at hvis NMH har købt et nyt motorhome. RentMotorHome beskæftiger sig med at oprette nye leje aftaler, og modtager altså information såsom dato fra starten af lejeaftalen. ActiveMotorhomeController er måske den, hvor navnet giver mindst mening, hvis man ikke ved hvad der menes med active. I den her applikation skal active forstås som leje aftaler, som er oprettet og ligger I systemet. Derfor håndterer ActiveMotorhomeController ting som fx at aflyse en allerede eksisterende aftale eller afslutte en aftale når kunden returnerer et motorhome.   
De modtager alle sammen information fra html siderne, som de sender videre til de relevante repositories. På nedenstående figur kan man se et diagram over hvordan controller modtager information og sender den videre.

## Creator (Jacob)

Creator GRASP design princippet beskæftiger sig med, hvem opretter hver enkelt klasse. Hvem har ansvaret for at skrive new. Det er naturligvis vigtigt, at det er ikke er tilfældigt hvem der gør det, både for læsbarheden af koden, men også så man har en klasse der er information expert, så den klasse der opretter et objekt også har den information den skal bruge for at oprette klassen.

I vores application kan dette bedst ses ved brug af Controller klasserne, som har ansvaret for at instantiere de respektive repositories. Det giver bedst mening, at netop disse klasser instantiere repositories, fordi det er her klasserne skal bruges.   
Det bliver også brugt når de forskellige klasser, både controllerer og repositories, opretter modeller motorhome. De bliver oprettet I klassen selv, da det er her de skal bruges og det er her de har informationen. Det er altså ikke en handler klasse eller lignende, som står for at lave nye Motorhome modeller.

Hvis man skal se på både creator og controller sammen, vil det være controller klassen som får data, og her bruger den informationen til enten at oprette modeller som den selv bruger, eller sender den relevante information videre til et repository som vil overtage ansvaret.

## Low Coupling (not done, måske ikke med I endelig rapport)

Low coupling GRASP princippet forsøger at opnå, at ændringer I koden påvirker applikationen så lidt som muligt. Det er ønskværdigt, så små opdateringer ikke ændrer programmets funktion eller introducere bugs. Det betyder at klasser og metoder skal være så uafhængige I deres virke fra andre metoder og funktioner, at det ikke ændre noget andre steder hvis den enkelte metode bliver opdateret.

I vores Spring application kan dette princip ses meget tydeligt I vores Thymeleaf fragments. Hvis man ønsker at opdatere menuen, som ligger på alle html sider, skal det kun ske I en html fil og vil være gældende for alle html sider. Det giver utrolig god coupling I den forstand, at uanset hvad, så kan menuen kun ændres ét sted, og hvis der er fejl I menuen kommer det ét sted fra. Det ses på samme måde I footer.