**Indledning til uc1 (begrundelse af valg af beskrivelse, hvorfor er det den vigtigste UC)**

Grunden til vi har valgt at gå mest i dybden med usecase 1 i forhold til produktbeskrivelsen er at den indeholder den største og vigtigste del af hele vores endelige program, altså Bestil Kørsel. Den har indeholder tråde under udregning af pris, den bruges til at oplyse alle informationer for kørsel til programmet når de videresendes og er den hele programmet mere eller mindre omhandler ved brug. Den indeholder både prisen, adressen, tidspunktet, dato, antal personer, antal hjælpemidler, antal bagage og en kommentar til MidtTrafik. Det er alle vigtige informationer for vores program, da vores andre usecases anvender disse information. Det er den vigtigste grund til at vælge denne usecase, udover det faktum at den har masser af indhold at forklare om.

**Formel usecase beskrivelse(variationer, hvert punkt beskrevet osv.)**

Først og fremmest er succesgarantien i vores formelle usecase at brugeren får registreret sin kørsel bliver sat klar til godkendelse i systemet. Desuden skal brugeren have besked om den succesfulde bestilling.

For hovedscenariet er punkt 1 bare interessepunkt, hvor brugeren gerne vil anvende bestillingssystemet. Punkt 2 er så hvor brugeren anvender programmet til at indtaste de nødvendige oplysninger for at udregne en pris til sin kørsel med flextrafik. Punkt 3 anmoder brugeren om at få udregnet en pris for sin kørsel, og i punkt 4 reagerer systemet på anmodningen og validerer oplysningerne, og hvis oplysningerne er indeholder fejl i forhold til udregning af prisen giver punkt 4a en fejlmeddelelse og går tilbage til punkt 2 for at genindtaste oplysninger. Hvis oplysningerne ikke er forkerte udregner systemet pris for brugerens kørsel i punkt 5. Da dette kører som en tråd så brugeren kan arbejde videre på sin bestilling samtidig med at programmet udregner giver programmet en besked om at den er i gang med at udregne prisen for brugeren. Når systemet har færdigudregnet prisen for kørslen viser den prisen for brugeren i systemet ved punkt 7. I punkt 8 indtaster brugeren så de manglende oplysninger såfremt han ikke har nået det under udregningen af pris, og i punkt 9 accepterer han prisen ved at anmode om at få registreret sin bestilling til systemet. Systemet validerer så igen informationerne, dog for hele kørslen denne gang, og sikrer sig at brugerens informationer er korrekte. Hvis der mangler informationer eller de er ugyldige giver systemet en fejlbesked i punkt 10a og specificerer hvad den ugyldige/manglende information er, og fortsætter så hovedscenariet fra punkt 8 igen. Hvis brugeren har indtastet informationerne korrekt registrerer systemet den bestilte kørsel i sit system i punkt 11, og i punkt 12 giver systemet en besked til brugeren om at registreringen af kørslen er blevet gennemført.

**Beskrivelse af brugergrænsefladen (ide(formel usecase) til mockup til scenebuilder, gestaltlove) ->javafx ->fxml filer del op fra kode -> haandter metoder fxml**

Vores brugergrænseflade for usecasen Bestil Kørsel er sat op efter at være så overskuelig som mulig på trods af at rigtigt meget information skal udfyldes. Vi startede med en mockup hvor vi nogenlunde gik igennem idéer til hvordan opsætningen kunne være i selve programmet, og kom frem til at det ville være en god idé at vælge at sætte startadressen og alle informationer på en enkelt linje, og så slutdestinationen sat op på samme måde nedenunder, så man kategoriserer dem sammen. Da alt der ligger over prisen har relevans for selve kørslen med tid, dato og destination valgte vi også at sætte det op så tiden på startdestination bliver specificeret lige under startdestinationens linje, og dato og antal kilometer der bliver kørt bliver smidt ind nedenunder slutdestinationen. Under dette valgte vi at sætte prisen, for at dele siden i to så der er nemt at overskue hvilke informationer der var nødvendige at udfylde for at kunne udregne prisen. Det der ligger under prisfeltet er så resterende information om selve bilen, altså hvor mange personer der er med på turen og hvor mange pladser der skal fyldes op med bagage og hjælpeudstyr. Aller nederst giver vi en sidste mulighed for at man kan skrive en kommentar til admin så de kan se hvis der er nogle specielle behov brugeren har for kørslen der skal tages hensyn til. Under kommentarfeltet dukker det endelige bestillingsfelt så frem når man har fået udregnet prisen, så man ikke er i tvivl om man ikke kan bestille uden først at have fået en pris fastslået.

Vi har desuden valgt at lave videre til det aktuelle program fra mockup med JavaFX, eftersom FXML filer giver en god opdeling mellem selve designet fra vores GUI med dens controller, så det ikke ligger i samme klasse. Desuden gør det også at det er nemt at sætte metoder ind for de forskellige knapper og felter, hvor vi har valgt at bruge ”haandter” til de forskellige nødvendige felter. Dvs. hvis vi f.eks. skulle anvende en knap for at udregne prisen kalder vi den bare haandterUdregn, og kobler det på inde i JavaFX programmet så det også står i koden. Vores GUI controller håndterer derved det der sker ude i brugergrænsefladen og skaber en kobling imellem brugergrænsefladen og vores logik ifølge controller mønsteret i GRASP.

**Beskrivelse af domænemodel**

Vores domænemodel kigger på de koncepter og begreber der er i spil i hele vores system. Det viser hvordan de forskellige koncepter og begrebers sammenspil fungerer, og de forskellige funktioner der er imellem dem, altså alt hvad de hver især indeholder eller sender videre til andre koncepter. Vores domænemodel er sat op til at minde meget om et klassediagram, dog er det her mere en idé frem for det tekniske af programmet. Det er lavet alene for at skabe overblik over hele programmet, og hvordan de yderligere systemudviklingsartefakter skal laves, da domænemodellen viser sammenhængen mellem de forskellige koncepter. Ud fra domænemodellen udarbejdede vi vores prototype til databasen hvilket kun har haft meget få ændringer siden første forsøgte implementation. Farveopsætningen af vores domænemodel er lavet for nemmere at kunne se hvordan sammenspillet foregår imellem de forskellige koncepter og begreber, så de forskellige pile er sat op til at passe med at hovedkoncepterne sender dem videre eller får dem fra andre. Det vil sige alle pile der er sat til eller fra brugeren er farvekodet som rødt, alt der kommer til eller fra MidtTrafik er blåt, alt andet der kommer til og fra kørsel er gult, og resten er bare sat op som en standardfarve. Domænemodellen kan findes på INDSÆT BILAG HER.

**Aktivitetsdiagram**

Vores aktivitetsdiagram viser mere eller mindre hvordan vores formelle usecase er udarbejdet, da pointen i AD er at vise en diagramform for vores usecase, dog er dataformer imellem tilføjet så man har en bedre idé om hvad der foregår imellem anmodninger frem og tilbage mellem system og bruger. Altså, når brugeren har udfyldt prisoplysninger bliver der sendt Prisoplysninger videre som datatype til systemet, og når systemet oplyser pris for kørsel bliver datatypen Pris sendt tilbage til brugeren.

**SSD beskrivelse**

I vores systemsekvensdiagram bygger vi videre ud fra vores AD og formelle usecase. Her sender brugeren først et kald ”getPrisTilbud” over til FTP som FTP så validerer for at finde ud af om der er ugyldige oplysninger. Hvis den finder ugyldige oplysninger stopper programmet, ellers fortsætter den ned i diagrammet. Hvis den fortsætter i programmet sender FTP kaldet ”beregnPris” til flextur\_sats. Derefter bliver en besked til brugeren sendt om at FTP udregner pris med ”beregnerPrisBesked” imens at flextur\_sats arbejder. Når flextur\_sats har udregnet prisen, sender den et returkald til FTP ”kørselsPris” som FTP sender videre til brugeren med samme returkald bagefter. Når brugeren har fået sin pris sender han et kald ”accepterPris” med sine informationer som parametre til FTP, som igen validerer oplysningerne. Hvis oplysningerne er ugyldige sender den ”forkerteOplysningerException” som returkald til brugeren og stopper programmet, ellers kører programmet videre. Dernæst kalder FTP ”gemKørsel” på sig selv, og sender et returkald ”kørselsBekræftelse” tilbage til brugeren.

**DCD beskrivelse ->model view control her, -> grasp**

Vi har valgt at dele vores DCD op i to diagrammer, et for at vise forholdene imellem klasserne, og et andet for at vise indholdet af de klasser der indgår i DCD’et. Disse to kan findes i INDSÆT BILAG HER og INDSÆT BILAG HER. I selve indholdet af klasserne har vi valgt kun at vise hvad der er i vores interfaces, da vi vil have åbne muligheder for at redigere i datakernerne uden at vi løbende skulle opdatere vores DCD hele tiden. Klassediagrammet giver dog stadig et godt overblik over vores program, specielt da den er delt op i de to forskellige diagrammer. Klassediagrammet viser hvordan vi bl.a. har anvendt trelagsmodellen, da vi har et logiklag der kalder ned på vores database og de er separeret fra hinanden, vores præsentationslag er dog ikke vist i klassediagrammerne da vi stadig vil have åbne muligheder for at redigere i præsentationslaget uden løbende at skulle konstant opdatere. Vores præsentation har heller ikke interfaces, så det ville ikke passe ind at sætte dem i vores DCD da der kun vises interfaces. Vores farvekodning i diagrammet med forholdet mellem klasserne er sat op så man kan se alle pile der kører i og fra logiklaget, da de alle er lavet røde, og alle interne pile i persistence-laget er sat til at være orange. Alle pile der kører fra persistence-laget til domænelaget er farvekodet til at være lyseblå. På den måde kan man nemt overskue hvilke piler der hører til hvilke domæner originalt, og kan nemt se når pilene i få tilfælde går over hinanden. Diagrammet er også en god måde at vise hvordan alt starter fra vores FTPController som sætter gang i hele programmet. Denne controller er også vores primære kobling til præsentationslaget, hvilket igen er et godt eksempel på trelagsmodellen. Vores diagram opsat efter interfaces viser bare alle de metoder der skulle implementeres, så man let kan genimplementere vores kode i form af et API. Grunden til at vores kørselshistorik-klasse er så stor er ikke fordi vi ikke har anvendt high-cohesion fra GRASP mønstret, men fordi at vores TableView i vores præsentation bruger en domæneklasse til at bygge dens felter op, så resultatet er at vi har brugt den store domæneklasse til at bygge vores TableView op, som gør den bliver lidt uoverskuelig. Et eksempel på hvordan vi har anvendt high-cohesion er hvordan vi bl.a. har delt vores StartDestination og SlutDestination fra Koersel-klassen ud i deres egen domæneklasse for at skabe mere overskuelighed.

**SD beskrivelse**

getPrisTilbud

Denne SD er lavet direkte ud fra OC1 og andre informationer fra de tidligere artefakter. Først kommer der et metodekald for getPrisTilbud på FTPController. Dette gør at FTPController skaber Validator, som laver en alt frame hvor den tjekker validerInformationer = true. Hvis det er true fortsætter programmet, ellers kommer en InvalidInformationException som et lost kald og slutter sekvensdiagrammet. Hvis programmet har fortsat forbi validationen skaber FTPController PrisBeregner, og så kalder FTPController en run metode på PrisBeregner. Her går de begge i tilstand for berening, hvor PrisBeregner kører beregnPris på sig selv og notifyObservers på sig selv. FTPController kalder så getPris på PrisBeregner.

accepterPris

accepterPris er også lavet ud fra OC1 og de tidligere usecase 1 artefakter. Den laver et metodekald med accepterPris på FTPController, som start en try/catch frame, hvor den prøver at kaste validerInformation på Validator, og catch fanger InvalidInformationException og sender som lost kald og slutter. Hvis der ingen exception blivet fanget i catch kører FTPController gemKørsel på KoerselsKartotek.