Indhold

[Visionsdokument (Enok) 1](#_Toc420483453)

[Vision 1](#_Toc420483454)

[Interessent analyse 2](#_Toc420483455)

[Featureliste 4](#_Toc420483456)

[Supplerende Kravsspecifikationer 5](#_Toc420483457)

[Sekvensdiagram (Enok) 6](#_Toc420483458)

[Klassediagram (Enok) 6](#_Toc420483459)

[Samspil mellem Sekvensdiagram & Klassediagram 7](#_Toc420483460)

[Databaser (Enok) 7](#_Toc420483461)

[Databasemodel: 8](#_Toc420483462)

[Normal former: 9](#_Toc420483463)

[1. Normalform: 9](#_Toc420483464)

[2. Normalform: 9](#_Toc420483465)

[3. Normalform: 10](#_Toc420483466)

[IT Sikkerhed 10](#_Toc420483467)

[GitHub (Enok) 11](#_Toc420483468)

[Alternative versioneringssystemer 11](#_Toc420483469)

# Visionsdokument (Enok)

Visionsdokumentet er et indledende dokumentet bestående af:

* Vision: En kort tekst som beskriver idéen og visionen for det software vi overvejer at udvikle.
* Interessent Analyse: En liste over interessenter og hvad de hver især er interesseret i, i forhold til vort kommende program.
* Featureliste: En liste over de features som vort program skal have.

## Vision

En kortfattet tekst der beskriver fremtidsvisionen for det nye stykke software, på en måde så vi ikke sammenligner det nye system med det gamle og ikke ligger os fast på nogen bestemt form for teknologi.  
Desuden skal visionen meget gerne afbillede de interesser som vore interessenter har.

I vores vision har vi forsøgt både at lave en inspirerende tekst, som vil kunne motivere alle i projektet til at lave et super godt program, men også afspejle det firma som køber programmet, nemlig Ferrari som netop har et helt bestemt brand, som vi gerne så afspejlet i programmet.

## Interessent analyse

Liste over de interessenter der har interesse i det kommende stykke software.   
Eksempler kan fx være Direktør/Direktion som køber softwaren, Brugere af programmer, Skat eller andre myndigheder, der fx kan stille krav til databeskyttelse eller har interesse i at moms og skat m.m. bliver beregnet korrekt.

Grunden til at man analysere disse interessenter er netop for at være sikker på at man tager hensyn til så mange af deres interesser så muligt. Således at man ikke pludselig opdager at programmet mangler en funktionalitet eller egenskab som en central interessent forespørger.

I forhold til vort kommende program har vi lavet analysen ovenfor.

I analysen indgår både interne interessenter, så som brugeren og de der skal betale for programmet (ofte en direktør eller direktion), samt eksterne interessenter, både kunden, men også de firmaer hvor vi benytter en service de har stillet til rådighed. I vort tilfælde er det Banken og RKI.

Endelig kan der også være lovgivere eller andre myndigheder som vi bør tage hensyn til i udformningen af vort program. I vores tilfælde Datatilsynet.

|  |
| --- |
| ”Vi forestiller os et FerrariFinanceSystem, som afspejler det brand som Ferrari er. Selve programmet skal understøtte det faktum, at Ferrari er et High-end produkt. Systemet skal være i stand til korrekt og sikkert at behandle oplysninger. Systemet skal være med til at skabe en unik følelse omkring købet af kundens nye Ferrari, der yderligere forstærkes i og med, at vi effektivt gennemfører låneforløbet, således at kunden er i stand til at kører væk i sin nye bil, den samme dag. Systemet skal være intuitivt, og derved være med til at give sælgeren mere frihed og overskud til at give en professionel betjening af kunden.” |
| Direktør:   * At salgsteamet for solgt mest muligt. * At låneanmodning bliver korrekt registreret med det samme. * At renten er korrekt beregnet. * At data ikke går tabt i processen. * At kontakt med bank/RKI forløber hurtigt og uden tab af data. * At låneanmodning fra samme kunde ikke bliver behandlet flere gange.   Sælger:   * At jeg sælger mest! * At låneanmodning kan godkendes hurtigst muligt. * At låneanmodning fra samme kunde ikke bliver behandlet flere gange. * At låneanmodning bliver korrekt registreret med det samme. * Lettest mulig arbejdsgang   Kunden:   * At få svar på låneanmodning hurtigst muligt. * At renten er korrekt beregnet. * At oplysningerne bliver opbevaret sikkert.   Bank:   * Korrekt brug af deres service.   RKI:   * Korrekt brug af deres service.   Datatilsynet:   * At data opbevares i overensstemmelse med persondataloven. |

Igennem ovenstående tabel forsøger vi at vise det samspil der er mellem visionen og interessentanalyse. Når vi står med det endelige program, ved afslutningen af projektet, bør vi være i stand til at holde det op imod hele visionsdokumentet, og se det afspejlet både i visionen, interessentanalysen og featurelisten.

## Featureliste

Kort tekst som beskriver de funktioner som det kommende program skal have, men på en sådan måde at vi heller ikke her ligger os fast på bestemte metoder eller teknologi, med mindre at det direkte er beskrevet i kravene til vort kommende program.

I forhold til vores feature liste havde vi en bestemt ting som var blevet pålagt af krav til programmet, nemlig at det skulle kunne eksportere lånetilbud m.m. til en CSV fil.

Så ud over de centrale ting in programmet måtte dette krav også med i feature listen.

# Supplerende Kravsspecifikationer

Når man har lavet visionsdokumentet, og især interessent analysen, kan det være særdels oplagt at man også kigger på Supplerende Kravsspecifikationer.

Supplerende Kravsspecifikationer er de ikke funktionelle krav som der kan stilles til programmet. Disse kategoriseres ofte i 4 kategorier:

* **Reliability**: Hvor pålidelig er systemet. Det kan fx være krav til hvor meget man forventer systemet skal være til rådighed. Et krav her kunne fx være 99,9% op-tid på systemet.
* **Availability**: Hvor let systemet er at bruge, fx for 1. gangs brugere. Et krav her kunne være at en første gangs bruger skal kunne foretage et salg på max. 5 minutter.
* **Maintainability**: Hvor let er vedligeholde. Der kan fx være krav til tredeling i software arkitekturen, eller til at nye moduller let kan installeres ind i det nye program.
* **Performance**: Hvad programmet skal kunne klare. Et typisk krav her kan være er mængden af beregninger pr sekund, eller hvor langtid en backup session fx højest må vare.

Grunden til at det er logisk at lave denne liste efter visionsdokumentet, er fordi at de krav man opstiller her ofte er udledt af de interesser som en interessent har.

Hvad er er fælles for alle de krav der opstilles under Supplerende Kravsspecifikationer, er at de er målbare. Dvs. at når vi er færdige med vort software, kan vi teste det og sammenligne resultatet med vore krav her og sikre os at alle de ikke-funktionelle krav er opfyldt.

# Sekvensdiagram (Enok)

Sekvensdiagrammet er et værktøj til software design. Her kigger man især på kommunikationen mellem softwareobjekter, samt på programmets adfærd under udførelse. Desuden er et sekvensdiagram med til at identificerer hvilke objekter der har ansvaret for konkrete opgaver.

Sekvensdiagrammet bør være opbygget på en sådan måde at den læses fra venstre til højre, og oppe fra og ned.

I her er et eksempel på et af vores Sekvensdiagrammer, som illustrer hvad der sker når vi beder systemet om at vise os alle låneanmodninger(Use Case FFS-02):



Ved at følge metode kaldene ned efter, kan vi følge den kommunikation der sker mellem programmets objekter og se hvem der laver hvilke objekter hvornår og i hvilken rækkefølge. Dette gør at når vi endelig sidder med programmeringen så vil vi let kunne se hvilke metoder objekternes klasse skal have. Samtidig kan vi også kontrollere at vi holder vores tredeling, hvis vi samtidig er lidt opmærksom på hvor hvilke klasser er placeret i vores software arkitektur. Hvis vi fx ser at DataAccess objekter kalder metoder på logik, så bør vi måske et eller andet i vort design.

Men tilbage til sekvensdiagrammet, for at lette overblikket er det vi ser på det foregående diagram, kun oprettelsen af en facade og der efter et nyt vindue, resten af diagrammet har vi valgt at placere i et andet diagram:



Dette diagram viser kommunikationen under udførelsen af selve hentning af data(hvilket vores FetchLoanRequest Command står for.

Igen diagrammet her er en del af en større sammenhæng, men for at fremme overblikket er det taget ud, for at diagrammet ikke skal blive FOR forvirende med alt for mange objekter og metode kald osv.

Og for at komme helt i mål har vi det sidste diagram i use casen:



Diagrammet her visser kommunikationen fra at swing command, som er vores tråd, får dataen og sender den retur til dialogframen.

Som før fortalt så giver sekvensdiagrammer fordele når vi endelig sider og skal kode vort program, man bør dog altid huske på at sekvensdiagrammet, lige som en del af de andre artefakter, er dynamiske, og vi kan ændre ting igennem hele projektet ind til vort software er endelig færdig. Så det er ikke sådan at designet er sat i sten så snart vi har lavet det første sekvensdiagram.

# Klassediagram (Enok)

Hvor SD diagrammet beskrev de dynamiske adfærd under kørsel af vort program, ud fra objekter, beskriver klassediagrammet den statiske struktur der findes i programmet ved at visualisere relationerne mellem softwareklasser.

Ofte vil et klassediagram tage et formål og beskrive det, for at fremme overblik ved kun at bekrive de detaljer der er relevant for netop det formål.

I vort diagram har vi dog forsøgt at få plads til de fleste klasser ved at bruge farve notation som gerne skulle fremme overblikket tilstrækkelig til at man vil kunne bevare overblikket.

**Farve kode:**   
Hvid(kun MainFrame): repræsenterer User Interface klasser  
Lys lilla: SessionFacade klasse  
Hudfarvet: Controller klasser  
lilla: Logic klasser  
Sandfarvet: DataAccess

Vi har brugt dette diagram til at forklare forholdet imellem vores UI og vores controllere. Hvor vi har en sessonFacade der fordeler alle metode kald fra UI til de rigtige klasser. Derudover har vi brugt klasse diagrammet til at sikre tredeling i software designet. Således at kommunikationen kun går en vej, nemlig fra toppen til bunden.

Normalt vil man i klasse diagrammet også have en liste over attributter og metoder i klasserne. Samt vise hvor vidt disse er private or public. Men vort diagram her er til for at visse relationer mellem de forskellige klasser.

## Samspil mellem Sekvensdiagram & Klassediagram

I sekvensdiagrammet arbejder vi med objekter, disse objekter er enten afledt af klasser eller Javas indbyggede klasser og metoder. Dvs. at når vi laver klassediagrammet kan vi bruge vort sekvensdiagram. De fleste objekter, som ikke kommer fra en javaklasse, skal have deres egen klasse i klassediagrammet.

# Databaser (Enok)

For at gemme data i et program bruger man databaser. Databasen er en struktureret samling af digital data gemt som bits og bytes.

## Databasemodel:

For at lette selve processen med at designe databaser, laver man en database model. Modellen består af de forskellige tabeller, samt de attributter som hvert tabel indeholder.   
Derudover viser modellem også relationen mellem de forskellige tabeller med multipliciteter samt foreign keys osv.

Normalt vil man sige at vi har for mange tabeller i vort diagram da vi mennesker kun kan overskue 8 +/-2, men ved hjælp af farve kodning har vi gjort det meget lettere at holde overblikket.



Alternativt vil man tage de enkelte områder og tage dem ud i diagrammer for dem selv. Som vi har gjort her med vort ”Bil” diagram.

En ting at notere her er især tabellen car\_config\_component, som kun består af foreign keys. Dette betyder at tabellen er sat ind imellem 2 andre tabeller for at forhindre en mange-til-mange relation.

## Normal former:

For at gøre strukturen i databaserne og derved også søgbarheden bedre, har man nogle normalformer, som man bruger til at Normalisere databaser. Disse normalformer sikre at vi ikke bruger for meget data i vores databaser, ved fx gengivelser, men sørger også for at vi let kan finde rundt imellem linkede tabeller, og at vores program hurtigt og effektivt vil kunne søge efter data i vores database tabeller.

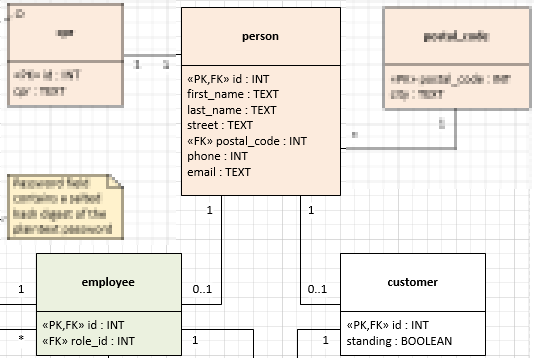
### 1. Normalform:

* Ingen felter med mere end 1 værdi.  
  Dvs. man må ikke gemme både postnummer og by i samme felt, men skal splittes ud i 2.
* Fordel data i særskilte tabeller med en slags information i hver.
* Ingen gentagelser. Så frem der er gentagelser bør den gentagne data tages ud og sættes i en selvstændig tabel, som så forbindes med en fremmednøgle/foreign key.

### 2. Normalform:

* Så frem data kan bruges i flere tabeller bør de placeres i sin egen tabel.

I vort system har det betydet vi har lavet en Person tabel, og knytter så hver person til enten en Custommer eller Emplyee tabel.



### 3. Normalform:

* Intet felt må afhænge af andre felter end primær nøglen.   
  For at overholde denne normal form, bliver vi fx, nød til at tage by ud af Person tabellen, og placere den i en tabel for sig selv. For by navnet afhænger nemlig ikke af primærnøglen i Person tabellen, men af postnummeret.



Desuden skal det bemærkes at for at 2. og 3. normalform kan være opfyldt skal de(n) foregående normalform(er) være opfyldt.

## IT Sikkerhed

Når man taler om IT sikkerhed er der en bestemt ting man altid bør tage højde for når man laver et stykke software: Hvor sikker er den information du gemmer?

I forhold til vort projekt er der en central information som er særdels følsom for både kunder og ansatte: CPR-nummeret. Dette kan give en evt. informations tyv mulighed for at foretage identitets tyveri og derved skade vore kunder og ansatte. Det er derfor meget vigtig at vi tager højde for dette denne risiko når vi laver vort system.

Måden vi har valgt at sikre CPR dataen på er ved at placere den i en tabel for sig selv. Dette gør at man kan begrænse adgangen til netop denne tabel, så det kun er særdels betroede medarbejdere der kan få adgang til den. Derved begrænser vi adgangen til den følsomme data.

Alternativt bør man, som minimum, sørge for at CPR-nummeret ikke bruges som primær nøgle i en person tabel. Således at det ikke er CPR nummeret der bruges til at hægte personer i Person tabellen sammen med andre elementer i andre tabeller.

Ud over CPR nummeret kan der også være andre data man bør behandle på en bestemt måde. Fx passwords. Et password kan give direkte adgang til vort system så frem det bliver stjålet fra databasen, så kan være en god idé at beskytte det.

Vort eksempel på databeskyttelse i vores database model:

Til slut skal det bemærkes at vi i Danmark har et Datatilsyn som opstiller regler for hvordan personfølsom data skal behandles og opbevares, og disse regler skal vort program og software selvfølgelig leve op til.

# GitHub (Enok)

Github er et det et webbaseret Git repository håndterings værktøj, som man bruger til at håndtere kildekode og evt. dokumenter til sit projekt. Det giver en mulighed for let at udgive forskellige versioner af sit program, efterhånden som projektet skrider frem. Desuden hjælper det en til at lade flere personer arbejde på den samme kode, og lade dem ændre i koden så at sige live.

Der arbejdes i brances, som der comittes til, og pulles fra. Og når man ønsker at lave en ny version, så oprettes en ny brance, som man så kan arbejde videre i. Men den gamle beholdes ofte, så man kan se hvor meget reelt blev lavet osv.

I vort projekt har vi brugt en brance strategi hvor vi har kørt vores brances i forhold til implementering, men set i bagklogskabens klare lys, vil vi måske have fået mere ud af at have en brance for hver eneste iteration. Dette ville have gjort at vi virkelig kunne se udviklingen i vore artefakter. Hvilket, ud fra et undervisningssynspunkt, ville have været bedre.

## Alternative versioneringssystemer

Af alternative værkstøjer kan nævnes: CodePlane, BitBucket, Source Forge og GitLab.

Grunden til at vi har valgt GitHub er mest at det er det vi har brugt i undervisningen.