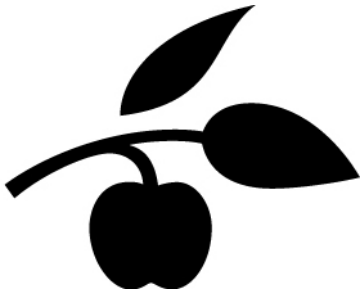


Syddansk Universitet

Teknisk Fakultet

Mærsk Mc-Kinney Møller Instituttet

Udvikling af Softwareprogrammer

Krediteringssoftware til TV2

Præsenteret af:		**Eksamens nr.**
Jonas Beltoft	jobel20@student.sdu.dk	493537
Hans Pedersen	haped20@student.sdu.dk	494042
Victor Bruun	vbruu20@student.sdu.dk	177620413
Jesper Diederichsen	jedie20@student.sdu.dk	496428
Casper M. Stillinge	casti19@student.sdu.dk	480325
Peter Ratgen	perat17@student.sdu.dk	454308

Vejleder:

Henrik Lykkegaard Larsen hlla@mmmi.sdu.dk

Semester: F21
Fagkode: T510043101
Gruppe: SE04
Aflevering: 31.08.2021

I Titelblad

Titel:	Udvikling af Softwaresystemer
Institution:	Syddansk Universitet, Det tekniske Fakultet Mærsk Mc-Kinney Møller Instituttet Campusvej 55, 5230 Odense M
Uddannelse:	Software Engineering
Semester:	2. Semester, F21
Kursuskode:	T510043101
Projektperiode:	F21
Omfang:	10 ECTS
Vejleder:	Henrik Lykkegaard Larsen
Projektgruppe:	SE04

I.1 Bidragserklæring

Ved at underskrive dette dokument bekræfter hvert enkelt gruppemedlem, at alle hæfter kollektivt for dokumentets indhold, samt at alle har bridraget til projektets udførelse.

Jonas Beltoft	`jobel20@student.sdu.dk`
Hans Pedersen	`haped20@student.sdu.dk`
Victor Bruun	`vbruu20@student.sdu.dk`
Jesper Diederichsen	`jedie20@student.sdu.dk`
Casper M. Stillinge	`casti19@student.sdu.dk`
Peter Ratgen	`perat17@student.sdu.dk`

II Resumé

III Forord

Denne rapport er udarbejdet af gruppe SE04 på software-engineering's bachelor studie gennem andet semester. Rapporten er skrevet med henblik på at oplyse læseren om, hvordan semesterprojektet er blevet udarbejdet, såvel som at vise hvordan det endelige slutprodukt fremstår. I denne rapport lægges der vægt på, at beskrive gruppens fremgangsmåde, tankeproces og arbejdsprocesser, for at kunne give et komplet billede af resultatet og vejen dertil. Gruppens arbejde med at programmere et krediteringssystem til TV2, som skal kunne erstatte de rulletekster, der normalt ville være efter endt program, har til formål, at kunne frigive mere tid mellem udsendelserne, som TV2 i stedet vil kunne bruge på ekstra programmer eller reklamer.

Dette semesters projekt omhandler brugen af databaser og optimeret objektorienteret programmering. Projektet er i forbindelse med vores to fag også bygget op omkring en udleveret case fra TV2. I denne case findes der en beskrivelse af hvilket system der skal udvikles, hvortil der er en række krav for, hvad systemet skal kunne. Løsningen af dette projekt er yderligere udvidet med flere implementeringer, som vi i gruppen vurderede nødvendige, for at slutproduktet er kategoriseret som færdig og optimalt.

Vores rapport er henvendt til enhver læser, som har en interesse inden for udvikling af et softwaresystem, samt processen som ligger bag udviklingen. Rapporten henvender sig også til de medarbejdere fra TV2 som har med vores case at gøre, for at give dem et indblik i udarbejdelsen af et softwaresystem, som kan supplere deres programmer i fremtiden.

Indholdsfortegnelse

IV Læsevejledning

Vi har som gruppe udarbejdet denne rapport over vores semesterprojekt, med hensigt på at skabe en overskuelig læseoplevelse. En læseoplevelse der skal beskrive vores tanker, beslutninger og de processer der danner rammer for vores projekt. Dertil har denne rapport til formål at belyse, diskutere og reflektere hvorvidt problemet for projektets case er løst og om vores produkt følger de givne krav.

Rapporten er bygget enkelt op, og for at skabe en let læseoplevelse anbefales det, at den læses fra toppen og ned. Derudover besidder rapporten et resumé som kan læses, hvis man ikke føler for at pløje igennem 40+ sider.

V Redaktionelt

Ansvarsområder			
Afsnit	**Ansvarlig**	**Bidrag fra**	**Kontrolleret af**
Forside	Victor Bruun		
Titleblad	Victor Bruun		
Resumé			
Forord	Victor Bruun		
Indholdsfortegnelse	Victor Bruun		
Læsevejledning	Victor Bruun		
Redaktionelt	Jesper Bork	Victor Bruun	
Indledning	Victor Bruun		
Fagligt vidensgrundlag	Alle	-	
Metode og planlægning	Jesper Bork		
Hovedtekst	Alle	-	
Krav	Jesper Bork		
Analyse			
Design			
Database Design	Jonas		
Implementering	Peter Ratgen Hans Petersen Jonas Beltoft		
Test	Peter Ratgen		
Diskussion	Jesper Bork		
Konklusion			
Perspektivering			
Procesevaluering			
Referenceliste			
Bilag			

1 Indledning

Dette semesters projekt tager udgangspunkt i TV2 og deres måde at håndtere krediteringer på. TV2 er en dansk tv-station, der blev grundlagt i 1988. TV2's forretning og den måde de tjener deres penge på, fungerer ved at lave kvalitetsbevidst tv til de danske stuer. TV2 gør deres tv-kanaler tilgængelige hos forskellige tv-udbydere, oveni at de viser reklamer mellem deres programmer, som skaber deres hovedsagelige indkomst.

TV2 er, ligesom alle andre tv-produktioner, en del af et stort marked, hvor tv-skærmen er deres vindue ud til forbrugeren. På sådanne et marked handler det om at få seeren til at bruge mere tid på sine egne kanaler, frem for konkurrentens. Hvis man vil holde styr på konkurrencen mellem kanalerne, vil det nemmeste nok være at kigge på, hvor mange seere der er på en given kanal, og hvor mange minutter seeren bruger på kanalen. Ud fra Kantars seereundersøgelser kan alle og enhver se seertallene på de forskellige tv-kanaler. Kantar indsamler nemlig seeretal hvert sekund, for at kunne levere de mest præcise seeretal [1] , som programlæggere i tv-branchen bruger til at forbedre tv-oplevelsen for seerene, og fastlægge hvilke tider i sendefladen, der er mest værdifulde. I følgende figur, som både er tilgængelig på Kantars hjemmeside, men også i vores case fra TV2 af, kan man se hvor mange minutter TV2s kanaler bliver set i forhold til de andre tv-kanaler.

Broadcaster	Antal min. dagligt	Antal min. ugentligt	Seerandel %
TV 2	74	519	51,5
DR	45	314	31,1
NENT	12	83	8,2
Discovery	6	43	4,2
Fox	3	19	1,9
Viacom	1	9	0,9
Disney	1	6	0,6
Turner	0	1	0,1
Andre kanaler	3	18	1,8

Figur 1: Seertid i uge 3 2020

Det første tal der ses i figur 1 viser, hvor mange minutter den gennemsnitlige dansker bruger på en given broadcasters tv-kanaler. Det næste tal viser hvor mange minutter det er om ugen, og det sidste tal viser, hvor stor en seerandel den givne broadcaster besidder på markedet for perioden. Som det ses har TV2 en stor seerandel, og af den grund består en vigtig del af deres indkomst af de reklamer de sender mellem deres programmer. Men den indkomst kan blive bedre endnu, hvis de bliver bedre til at optimere tiden mellem deres programmer, vurderer TV2. Og det er netop dette problem som danner grundlag for vores case.

1.1 Redegørelse for den udleverede case

Når et program slutter på en af TV2s kanaler, bliver der vist rulletekster for at kreditere de medvirkende i programmet. Disse rulletekster tager dog i nogle tilfælde op til 30 sekunder, hvilket er 30 sekunder for meget. TV2 har vurderet, at hvis man kan frigøre disse 30 sekunder fra rulletekster, og bruge dem på reklamer i stedet, vil de kunne tjene op imod 60 millioner DKK om året. [2]

Netop dette vil TV2 gøre ved at digitalisere krediteringerne, så de kan ses på nettet eller en app. For Udover at kunne sende flere reklamer mellem deres programmer, vil TV2 også have muligheden for at kreditere alle, og ikke bare de vigtigste efter endt program. I de fleste tilfælde var 30 sekunders rulletekster nemlig ikke nok til at kreditere alle medvirkende, så derfor har TV2 været nødt til at prioritere, hvilket selvfølgelig fører til at nogle krediteringer bliver glemt. Digitaliseres krediteringerne, vil alle kunne blive krediteret ligeligt, og der vil derved ikke være nogen som bliver glemt, fordi deres rolle er mindre vigtig.

Vores opgave i denne case er at skabe netop denne digitalisering af krediteringer. Det er vores opgave at udvikle et software system som kan tilføje, fjerne og redigere krediteringer i en database, hvortil vi har formuleret en problemformulering, som skal hjælpe os med at få udarbejdet en løsning på TV2s problematik med kreditering.

1.2 Formålet med opgaven

Formålet med det system vi har udviklet, er at gøre det lettere for producere, administratorer og TV2 som virksomhed, at kunne redigere og holde styr på krediteringer. Derudover er formålet med projektopgaven også at vi som gruppe skal forbedre vores evner inden for programmering af softwaresystemer og håndtering af databaser. Samtidigt med at vi videre udvikler vores kompetencer inden for gruppearbejde, og opsætning af et struktureret arbejdsmiljø.

Vores semesterprojekt er inddelt i to; først og fremmest har vi vores "inception", altså første halvdel. Denne del gik ud på at få udarbejdet et inceptionsdokument, som skulle lægge et fundament for vores videre arbejde med projektet. Formålet med dette inceptionsdokument var at vi som gruppe skulle ende med et veldefineret projektgrundlag, som indebar at der var styr på krav, scope og problemformuleringen. Udover dette skulle vi også klargøre, hvilke metoder vi ville benytte os af, for at kunne løse den problemformulering vi havde sat os for.

Som anden del af vores projekt, har vi den faktiske løsning af projektets problemformulering. Her har vi færdigudviklet koden, samt benyttet os af vores forarbejde i første del af projektet til at kunne besvare vores problem formulering så præcist som muligt.
Fordelen med at have delt vores projekt op i to halvdele, har været at vi som gruppe i første halvdel har kunne skabe os en forståelse for hvad problemet er, hvordan vi vil gribe casen an og hvordan vi som gruppe vil løse problemet på den bedst mulige måde. Dertil har vi så ef-

terfølgende, i anden del af projektet, kunne fokusere på at følge vores planer og benytte vores forarbejde, til at udvikle det bedst mulige softwaresystem, og besvare problem formuleringen uden problemer.

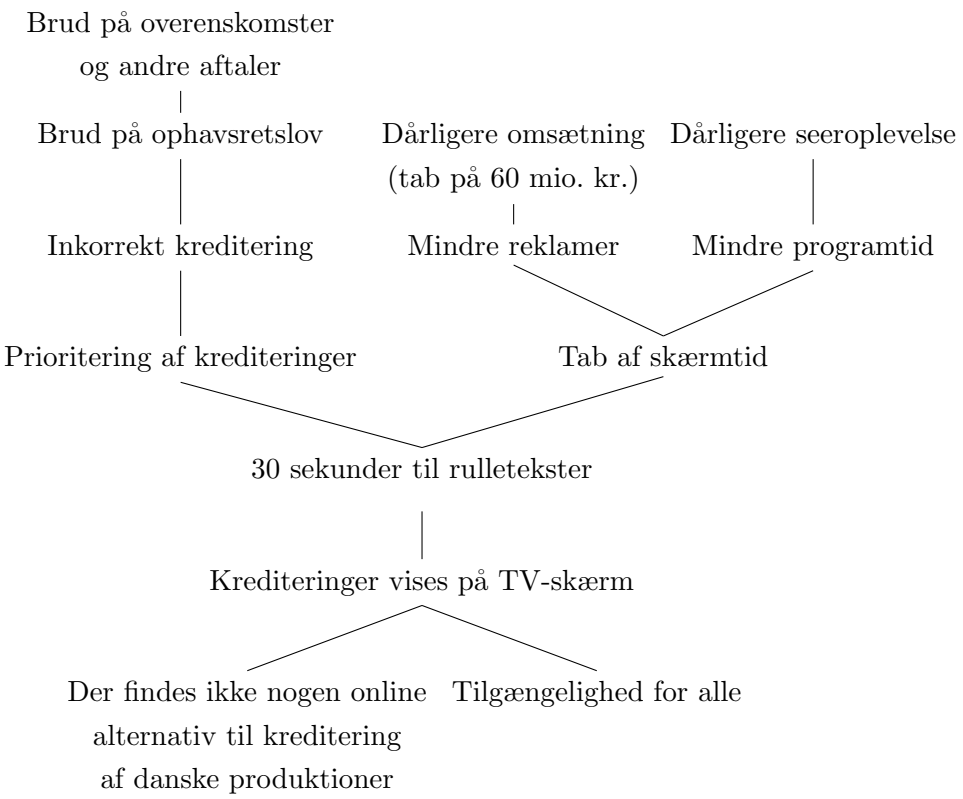
1.3 Problemanalyse

For at gå i dybden med den stillede case fra TV2, og dermed vurdere hvad det essentielle problem er, samt skabe os en problemformulering at arbejde ud fra, har vi først analyseret casen, med fokus på at identificere, hvilke mulige problemer der er. Den case som TV2 har lavet, er relativt udpenslet, og der er ikke meget tvivl om, at det essentielle problem for dem er, at det er forbesværligt at håndtere krediteringer til produktioner på den måde, som de gør i øjeblikket, nemlig at synliggøre krediteringer i 30 sekunder efter endt program i form af rulletekster. Denne nuværende løsning er et problem, da 30 sekunder ikke er nok til at kunne vise alle medvirkende i nogle udsendelser. [2] Der er også flere regler, som skal overholdes når der vises rulletekster, som f.eks:

- Ophavsretsloven
- TV 2s overenskomster
- Kontrakter for ophavsmænd og udøvende kunstnere
- Almindelig god skik

Ud fra de informationer vi har erhvervet os fra casen omkring TV2s regler for hvad og hvordan der skal angives krediteringer, har vi skabt et problem træ, der skal visualisere hvordan vi ser dette problem. Dette problemtræ ses i figur 2.

Kigger man på problemtræet, vil man kunne se, at de 30 sekunder, som der maksimalt må vises rulletekster i, skaber nogle problemer for TV2. TV2 mister nemlig skærmtid, af at skulle vise rulletekster efter hvert program, men udover det, har de, som tidligere nævnt, svært ved at få plads til samtlige krediteringer på 30 sekunder. Vi vurderer derfor at årsagen, bag disse problemer er, at TV2 er begrænset til at vise krediteringer i tv'et, og dette problem kan løses ved at flytte krediteringer over på et andet medium end tv-skærmen.



Figur 2: Problemtræ omkring de informationer, der er givet i casen.

1.4 Problemformulering

Ud fra vores analyse af TV2s case, er vi kommet frem til en problemformulering og dertilhørende underspørgsmål. Vores problemformulering har vi formuleret således:

Hvordan kan vi udvikle en prototype til et krediteringssystem, der vil kunne erstatte de klassiske rulletekster efter et afsluttet program?

Og vores underspørgsmål, til at supplere problemformuleringen lyder således:

- Hvordan er reglerne for krediteringer for danskproducerede programmer?
- Hvilke informationer skal og må inkluderes i krediteringer?
- Hvordan kan sådan et program gøres mest mulig brugervenligt(tilgængeligt) for både seer og administrator?
- Hvordan kan vi sikre et velstruktureret program, der mindsker fejl og ventetid samt strukturere databasen således, at der undgås fejl, duplikering af værdier og null-værdier?

1.5 Afgrænsninger

Projektet kan blive stort, og der er et utal af muligheder for, hvordan man kan skabe en overskuelig softwareløsnign på TV2s problem. Derfor har det også været vigtigt for os, at begrænse os. Vi vil gerne undgå at bruge for meget tid på, funktionaliteter, som ikke er vigtige, og lægge vores kræfter i de ting, som vi ved, vi kan overskue at gennemføre, og som der er tid til at få udviklet. Gruppen holder sig derfor til, udelukkende at arbejde med danske krediteringer, og undgår derved bevidst krediteringer til udenlandske shows og programmer. Gruppens produkt udarbejdes også kun som en prototype, og det betyder, at produktet ikke vil blive udviklet til et færdigt produkt, som vil opfylde alle givne krav fra TV2s side af. Krediteringerne vil dog stadig overholde de givne regler for krediteringer, men kommer til slut ikke til at være et produkt TV2 kan adaptere direkte over til.

2 Fagligt vidensgrundlag

Vidensgrundlaget for udarbejdelsen af denne rapport gennemgås i dette afsnit, med redegørelse for relevante begreber, teori samt det faglige vidensgrundlag. Dette afsnit tager udgangspunkt i det tilsvarende afsnit fra inceptionsdokumentet med tilføjelser samt udbedringer. Gruppens viden og færdigheder omkring udviklingen af software programmet, samt metode og planlægning, bygger på det lærte indhold fra de øvrige fag på 2. semester.

2.1 Kravudvikling

Når kravene til et system skal findes vil man som regel følge et workflow, der sikrer at kravene bliver så præcise, komplette og konsistente som muligt.

- Præcise krav, da tvetydige krav kan fortolkes på forskellige måder.
- Komplette krav, så de indeholder beskrivelser af alle aspekter.
- Konsistente krav der er konsekvente. Kravene må altså ikke modsige hinanden eller være i konflikt med hinanden.

Kravene til et softwaresystem er derfor vigtige at have styr på, da de fastsætter hvad systemet skal gøre, og definerer begrænsninger for dets udvikling og drift.

For at få et optimalt workflow til at få skabt nogle gode krav, har vi i gruppen kigget på at lave en brugsmønstermodel, også kaldet use-case. Det indebærer, at vi først har fundet de aktører, der forbindes med systemet. For at identificere en aktør kan man spørge: "Hvem eller hvad bruger systemet?Til hvad?"og "Hvorfor?". Man kan kigge på hvilken rolle de spiller i interaktionen med systemet. Man kan også spørge, hvilke andre systemer bruger eller bidrager med til systemet. Når man er afklaret med hvilke aktører der er, skriver man dem op i listeformat, hvor man inkluderer aktørernes navne, deres mål (hvad og hvorfor) og deres bidrag.

Næste punkt i en brugsmønster-model er at finde og identificere brugmønstre. Når man identificerer brugsmønstre vil man gå ud fra listen over aktører, og stille sig selv en række spørgsmål. Hvilke mål har en aktør? Det er skrevet ned i listen, og man kan derfor lave et brugsmønster til hvert mål aktøren har. Hvilke funktioner vil en specifik aktør have af systemet? Til det kan man lave et brugsmønster til en funktion, der giver målbare resultater til aktøren. Gemmer systemet information, som den henter frem? I så fald, for hvilken aktør gør den det? Dette er også værd at overveje, når vi skal lave brugsmønstrene. Ligeledes som med aktørerne, laver vi en liste over brugsmønstre, der indeholder deres navne og deres formål. Til slut skal der identificeres systemgrænser. Til det skal der først tegnes et brugsmønster diagram med aktører, de fundne brugmønstre, og linjer herimellem, der forbinder aktøren med de relevante brugsmønstre. Nu er det vigtigt, at man evaluerer, sine brugsmønstre, aktører og

emnet. Ud fra sine evalueringer kan man nu danne sig et billede af, hvilke krav, der stilles til systemet, og man ender derfor med en skitsering af en kravspecifikation til systemet.

2.2 Analyse af brugsmønstre

Brugsmønstrene kan vi derefter analysere og udarbejde sekvensdiagrammer og operationskontrakter, som vi kan overføre til et klassediagram og derved opbygge et overblik over systemet.

Dette foregår ved at der bliver udvalgt en række brugsmønstre til analysen. Til hvert af disse udformes et sekvens diagram, som viser det specifikke hændelsesforløb indefor et brugsmønster og de objekter og den adfærd der er beskrevet i brugsmønsteret.

Ud fra skevens diagrammet laves en operationskontrakt, som beskriver de forpligtigelser operationen har, samt hvilke ændringer der forekommer i systemet efter operationen er kaldt.

Dernæst kan et systemsekvens diagram udarbejdes. Et systemsekvens diagram er et udvidet sekvensdiagram, som viser den komplette række af begivenheder der udføres når operationen udføres. Til sidst kan det udvidet sekvens diagram overføres til et klassediagram.

2.3 Arkitektonisk design

Arkitekturen for systemet fastlægges tidligt i elaborationsfasen, når det er Unified Process der bliver benyttet. Det er vigtigt for ethvert software projekt at systemets arkitektur er velovervejet og godt udarbejdet, da det vil få stor betydning for det endelige produkt kvalitet. Designet fastlægges hovedsaligt ud fra de ikke-funktionelle krav, som f.eks. at vi i dette projekt har et krav om en tre-lags arkitektur, der allerede har sat bestemte rammer for det arkitektoniske design.

2.4 Brug af SCRUM i projektet

SCRUM er et agilt projektudviklings værktøj der primært er udviklet til udviklingen af software. SCRUM bygger overordnet på 3 artefakter. Product backlog, sprint og inkrementer.

Product backlog Product backlog er en liste af opgaver/krav der skal implementeres på et produkt for at nå et fælles produktmål. Emnerne i product backlog er prioriteret således at det mest nødvendige implementeres først. Disse emner er hvad der indgår i det næste SCRUM artefakt, sprintet.

Sprint Et sprint er en plan for at implementere en mængde af krav, for at opnå et mål. Man deler sprintplanlægningen op ved at besvare følgende spørgsmål om det. hvorfor, hvad og hvordan. Ved at finde ud af hvad målet for sprintet er, har man hvorfor sprintet er vigtigt. Derefter vælges de emner i product backlog, der, når implementeret, opnår dette mål. Disse emner sættes ind i et såkaldt sprint backlog der fungerer på samme måde som product backlog,

blot kun for et sprint. Dette sprint backlog er også prioritet er derved har man en plan for sprintet, og derved besvaret hvordan.

Inkrement Et inkrement er et betydeligt skridt på vej hen mod det endelige produkt. Et inkrement skal præsentere en virkende del af programmet. Det vil sige der hele vejen igennem er et kørende program, der udbygges med features inkrement efter inkrement.

SCRUM benytter sig også af nogle roller, der er essentielle for succesfuld brug af værktøjet.

SCRUM master En SCRUM master står får at få etableret SCRUM-flowet. Dette gøres ved at de skaber en fælles forståelse for hvordan arbejdsprocessen ser ud, og hvilket værktøjer der anvendes og hvordan. Det er SCRUM masterens opgave at sørge for hele processen bliver overhold, for hvis ikke dette formås, forsvinder alle fordelene ved SCRUM. I Gruppen har Hans Pedersen ageret som SCRUM master

Product owner En product owner er ansvarlig for product backlog. Det er vigtigt at de emner der skal op på product backlog, går gennem product owner, så de har det fulde overblik over hvad produktet indeholder. Derfor er der også kun én product owner, så man sikrer sig at der altid er en der har 100% styr på hvordan produktet ser ud, og hvordan det skal se ud. Idet gruppen er lille nok til kun at indeholde et SCRUM team, har gruppen udnævnt at product master er hovedansvarlig for produktet. I gruppen har Jonas Beltoft ageret som product owner, da han er iderig, og har gode evner indenfor programmering, og visualisere sig programmer.

Developers Developers, eller udviklere, er dem der sørger for at et sprint rent faktisk også arbejder fremad mod et inkrement. De står for at implementere de emner der er tilføjet til sprint backlog I gruppen er alle medlemmer indgået som udviklere.

2.5 Objekt-orienteret programmering

Hvad er OOP? Objekt-orienteret programmering (eller OOP), er et programmeringsparadigme baseret på et koncept af "objekter". Disse objekter kan indeholde en række data, generelt kaldet attributter. Kode kan være skrevet i form af en procedure, der skal lave noget bestemt, dette bliver kaldt en metode. Et eksempel kunne være at kode en person, denne person ville være et objekt. Personen har flere forskellige metoder til at kunne tale, bevæge sig, trække vejret OSV. Person objektet skal også have nogle attributter så som en højde, deres køn, deres alder osv.

Ved brug af Java-kodesprog i dette projekt kommer vi til at gøre stor brug af klasser. Når man definerer en klasse, opretter man egentlig et blueprint til et objekt. Heri defineres ikke direkte data, men der defineres hvad der skal gøres med noget data der er givet til objektet.

2.5.1 OOP's fire basale koncepter

Indkapsling Når man snakker om indkapsling inden for OOP, så betyder det at nogle data eller noget funktionalitet bliver indkapslet. Dette gøres ved hjælp af Access modifiers. Access modifiers hjælper til at bestemme, hvem der har adgang til data og funktioner, og hvor meget der er adgang til. Der findes en række forskellige access modifiers som kan ses på tabellen 1 på side 9. Indkapsling er en god måde at lave dataskjulning for andre klasser. Indkapsling gør det også nemmere at teste produktets sikkerhed.

Access Modifier	**Inden for klassen**	**Inden for pakken**	**Uden for pakken kun af subklasser**	**Uden for pakken**
Private	X			
Default	X	X		
Protected	X	X	X	
Public	X	X	X	X

Tabel 1: Liste over tilgange ved de forskellige access modifiers

Abstraktion Abstraktion er en måde på at gemme specifikke værdier og metoder. Abstraktion kan ses som en udvidelse på indkapsling, siden det kan gemme information eller metoder for ekstern kode. Dette gør interfacet af objekter meget simpelt. F.eks. kan abstraktion forstås ved at kigge på kroppen, vores skind fungere som an abstraktion til at gemme hvad der foregår inde i kroppen.

Nedarvning Nedarvning kommer fra at være en subklasse. Her nedarver subklassen attributter og metoder fra dens superklasse. Dette gøre det muligt at nemmere kunne genbruge kodestykker. Derudover bruges det også til metodeoverskrivning så man kan opnå runtime polymorfi. Imodsætning til nedarvning, hvilket er it såkaldt "IS-A"forhold, findes der også et "HAS-A"forhold, et aggregat. Et HAS-A forhold beskriver de gange hvor et objekt gøre brug af et andet objekt til at gemme data. F.eks. hvis man har en kunde med nogle informationer inde i et objekt, så kan der herunder oprettes et adresse objekt hvilket indeholder information som by, kommune, post nummer osv. Et aggregat kan være god at bruge i situationer hvor en stor klasses information kan deles ud i mindre bidder. Det hjælper også meget på kode genanvendelse. I den virkelige verden kan vi forestille os nedarvning ved at kigge på insekter. Alle insekter har lignende egenskaber så som at have seks ben eller et exoskelet. Altså ville en grasshoppe eller myre have nedarvet disse egenskaber fra superklassen af insekter.

Polymorfi Polymorfi er muligheden for at et objekt kan tage flere former. Dette gør at vi kan implementere en metode fra en superklasse i en subklasse på forskellige måder. Det ville

altså sige at hvilket som helst subklasse objekt i Java kan hvilken som helst form som er i superklassens hierarki, inkluderende sig selv. Generelt er der to typer af polymorfi:

- Dynamisk polymorfi

  Dynamisk polymorfi er også kendt som run-time polymorfi. Dette kan beskrives ved at tænke på superklasse-subklasse forhold. De begge har den samme metode, og subklassen har overskrevet superklassens metode (metode overskrivelse). Når et objekt bliver tildelt en klasse reference, og en metode fra objektet bliver kaldt, så er det metoden indeni objektets klasse, der bliver udført. Der bliver altså ikke kaldt metoden fra reference klassen, hvis altså referencen er en superklasse.

- Statisk polymorfi

  Statisk polymorfi er også kendt som compile-time polymorfi eller metode overbelastning. Dette betyder, at man har angivet flere af de samme metodenavne, men med forskellige argumenter. Det betyder altså, når man kalder sådan en metode, så ville kompilatoren beslutte hvilken metode, der bliver kørt i forhold til hvilke parametre, der er angivet.

2.6 Trelagsarkitektur

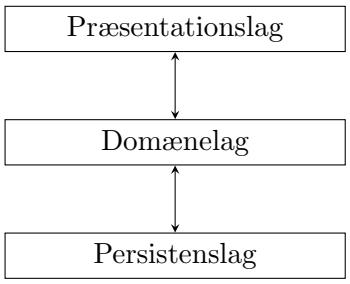
Trelagsarkitekturen er en softwarearkitektur, der deler programmet op i tre lag. Et præsentationslag, et logiklag og et datalag:

- Et præsentationslag som udelukkende står for at forsyne brugeren med en brugerflade og hente information fra brugeren og med den information kommunikere med det næste lag, logiklaget.

- Logiklaget (også kaldet domænelaget) står for alle de logiske operationer som er associeret med de gældende elementer i præsentationslaget på baggrund af den information, som brugeren har indtastet. Logiklaget kommunikerer også med datalaget, og kan her sammenholde information fra brugeren med det i databasen for at tage en beslutning om, hvad programmet skal gøre.

- Datalaget står for at lagre systemets data som logiklaget kan anvende.

Figur 3 beskriver denne lagdeling.



Figur 3: Beskrivelse af en trelagsarkitektur

For en succesfuld anvendelse af trelags arkitekturen vil brugeren udelukkende blive præsenteret for præsentationslaget og behøver ingen viden om de underliggende lag, da al kommunikation mellem bruger og program sker i dette lag. Brugen af trelags arkitekturen giver nogle fordele.

- Først og fremmest kan udarbejdelsen af de forskellige lag uddelegeres således at de udvikles samtidig. Dette gør udviklingen af programmet hurtigere
- Da al logikken sker et sted, bliver programmet mere skalérbart idet at koden/programmet ikke er kludret sammen og har for mange indgangspunkter.
- Hvis et lag bryder sammen eller ikke virker, påvirkes de andre ikke, da de er opdelt.
- Grundet at trelags arkitekturen opdeler præsentationslaget og datalaget med et logiklag, vil man med korrekt implementering kunne sikre, at en bruger ikke får tilladelse til at tilgå data, som ikke er tiltænkt dem.
- Lagdelingen bevirker at man kan ændre på implementeringen af et givent lag uden at de andre bemærker det.

2.7 Unified Process

Unified Process (UP) er en iterativ og trinvis udviklingsproces. UP er delt op i 4 trin, eller faser, disse og deres definitioner er:

- Inceptionsfasen

  I inceptionsfasen skal man finde ud af projektets gennemførlighed, finde frem til vigtige krav og identificere potentielle risici.

- Elaborationsfasen

  I elaborationsfasen skal der laves en arkitekturprototype, oveni at risikovurderingen forbedres. Hertil skal hovedparten af brugsmønstrene findes, samt at der skal lægges en plan for konstruktionsfasens forløb.

- Konstruktionsfasen

  I konstruktionsfasen færdiggøres den endelige identifikation af brugsmønstre, samt disses beskrivelse og realisering. Analyse, design, implementation og test færdiggøres også i denne fase. Undervejs redigeres projektets risikovurdering.

- Transitionsfasen

  I denne fase rettes fejl, og brugerne forberedes på at anvende softwaren. Her laves også manualerne og anden dokumentation, til sidst gennemføres et review af projektet.

En vigtig pointe omkring UP er at det er en proces drevet af brugsmønstre (use cases), samt at der fokuseres på risikostyring og arkitektur. Og som tidligere nævnt er der fokus på iterationer og en trinvis udviklingsproces. Dette betyder at systemet udvikler sig trinvist for hver iteration.

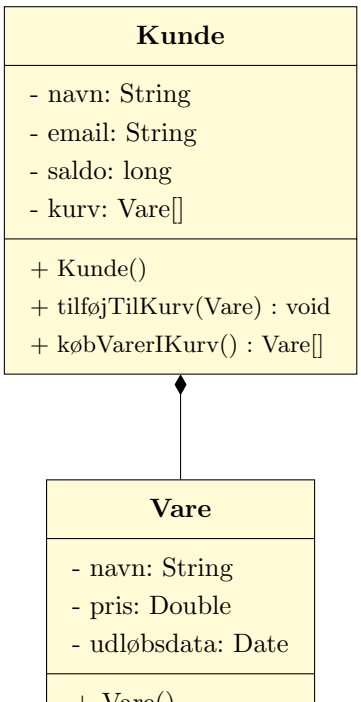
2.8 UML

UML eller Unified Modeling Language, tilbyder en måde at visualisere et systems arkitekturelle blueprints i et diagram.

2.8.1 Klassediagrammer

Det er bl.a. brugt til at lave Class Diagrams, da det giver en oversigt over hvilke klasser der er i et program, hvilke attributter og metoder de har, samt disses synlighed (Access Modifiers). UML anfører også sammenhængen mellem klasserne, om de fx nedarver fra klassen eller andet, hvilket er en stor del af grunden til at man bruger UML. Ud fra det kan man aflæse hvordan forskellige klasser bliver brugt andre steder i programmet, og derved få en bedre forståelse for hvordan programmet virker, uden at man har set programmet køre. Der er særlig notation til at fremhæve alle OOP's fire basale koncepter,

hvilket gør det muligt at udarbejde og/eller visualisere præcist det program, der er ønsket. Øverst ses navnet på den klasse, der beskrives. I sektionen under, er alle klassens attributter, og under dem er klassens metoder. Hvis klassen har en eller flere constructors, kan disse også ses, da de har det samme navn som klassen. På grund af alt dette, kan UML være et meget kraftfuldt værktøj under udviklingen af et objektorienteret program, da man kan visualisere alle de objekter og klasser, der bliver lavet og instantieret. Samt sammenhængen mellem disse objekter og klasser. UML bruges også til andre ting, såsom brugsmønster-diagrammer, som giver et overblik over aktører og de brugsmønstrer de interagerer med. Dette er en anderledes måde at modellere på, men det er alt sammen under



UML, da det er en universel måde at forbinde elementer på, i en visuel kontekst.

2.8.2 Beskrivelse af database med UML

UML kan også bruges til at vise en database. Det ses nemt hvilken udgave, der er brugt, da man her vil se notationer så som {PK} som beskriver at denne attribut er en primær nøgle i denne tabel. Hver kasse er her ikke en klasse, men en tabel. Den har kun attributter, da en tabel ikke har funktioner. Et UML diagram over en database vil også have forskellige måder at visualisere sammenhæng mellem tabellerne. Klassediagrammet som beskrevet har kun én standardiseret måde at forbinde klasserne på, hvilket gør det nemmere at lære og hurtigere at finde ud af, hvad der menes med de forskellige symboler man støder på, hvorimod et UML diagram over en database har flere forskellige måder at vise sammenhæng på, selvom de viser det samme. I klassediagrammet vil man se symboler som f.eks. betyder at denne klasse nedarver fra en anden klasse, eller bl.a. viser at den ene klasse "bruger"eller "afhænger"af en anden klasse. Dette finder man ikke i et database UML diagram. Her viser forbindelserne, hvor mange elementer, der kan være, f.eks. kan en kunde købe mange varer, men en type vare kan købes af mange forskellige kunder, så det vil kaldes en mange til mange forbindelse. Dette kan noteres på forskellige måder, f.eks. med crows foot, Chen eller andre. Vi som gruppe vil bruge Chen, som er en meget simpel måde at vise, om det er 0, 1 eller flere. Disse notationsformer er også brugt i ER og EER modeller, men det betyder ikke at et UML diagram er det samme som et ER eller EER diagram, da de ikke går direkte ned i databasen, men er mere en overordnet beskrivelse og kan vise andre ting end UML kan.

2.9 MoSCoW

MoSCoW er en model der bruges når en liste af krav skal prioriteres. Kravene indeles i 4 grupper, efter hvor vigtige de er for projektets succes. Her startes der med "Must Have"som er de vigtigste krav og som er nødvendige for at projekt bliver en succes. Dernæst kommer "Should Have"som er krav, der ikke er nødvendige, men det vurderes at de giver en tilstrækkelig værdi til projektet, i forhold til arbejdsindsatsen. "Could Have"er krav som ikke er vigtige for projekts gennemførsel, de kan tages med i tilfælde af overskydende arbejdsressourcer, men der planlægges ikke efter, at de skal medtages. Til sidst findes "Would Have"som er krav, der giver mening for projektet, men er valgt fra i den pågældende udgave af projektet.

2.10 FURPS+

- Usability

  "Usability"handler om de menneskelige interaktioner med programmet. Det gælder om at se på, hvor effektivt programmet er. Er dokumentationen for programmet i orden og uddybende nok, til at forklare hvad programmet står for.

- Reliability

  "Reliability"omhandler det, der vedrører oppetid, præcision i systemets beregninger.

- Performance

  "Performance"drejer sig om, hvordan systemet skal reagere på større mængder af information.

- Supportability

  "Supportability"omhandler krav, som vedrører testbarhed, kompatibilitet, konfigurerbarhed.

- Plus

  "Plus"er den del af FURPS+ der indeholder begrænsninger og specifikke krav for systemet.

  - Design constraints

    Man kigger på designmæssige begrænsninger, der kunne være for projektet. Ændrer I/O enheder eller den valgte DBMS, hvordan softwaren skal bygges?

  - Implementation requirements

    Man kigger også på implementationen og dets krav, og beskriver, hvordan programmørene skal forholde sig. Skal de bare forholde sig til standarderne?

  - Interface requirements

    Interface krav kigger på om, der er nogle interfaces som programmet skal fungere på eller med.

  - Hardware requirements

    Til sidst er der hardware krav, hvor der kigges på, om der er nogle krav til hvilken hardware systemet skal interagere med eller kører på, og om det skaber nogle forudsætninger for produktionen.

3 Metoder og planlægning

I løbet af projektet har gruppen benyttet både Unified Process (UP) og Scrum. Overordnet set har gruppen fulgt UPs faser og iterationer fra inceptionsfasen til transitionsfasen, dette har

givet struktur og målrettet det forberedende arbejde op til elaborationsfasen, hvor det faktiske udviklingsarbejde begyndte. Det var i denne elaborationsfase vi gjorde brug af Scrums agile projektudviklings værktøj. Der vil ikke blive yderligere redegjort for arbejdet i inceptionsfasen, da det alt sammen er dokumenteret i inceptionsdokumentet i bilag G.
Brugen af Scrum i elaborationsfasen og tilmed konstruktionsfasen, har fungeret godt for gruppen. Dette skyldes at hvert Scrum sprint har givet et "releaseable"produkt, lige vel som hver iteration i UP kræver. Ydermere har Scrums sprint metode været, en god hjælp til at fastlæge deadlines, for enkelte dele af udviklingsarbejdet.

3.1 Plan i elaborationsfasen

Gruppen fik i inceptionsfasen udarbejdet en række krav og brugsmønstre, som kan ses i afsnit 4.1 tabel 2, disse lå til grund for arbejdet med Scrum. Til første sprint startede gruppen ud med at oprette en product backlog, ud fra de højest prioriteret krav og brugsmøsntre. Til strukturering af produkt backloggen har gruppen benyttet sig af GitHub projects. Backloggen har gruppen benyttet ved at der i fællesskab, blev tilføjet en række opgaver til en To Do liste, i starten af hvert sprint, hvorpå medlemmerne kan påtage sig en eller flere opgave. Når en opgave er tildelt, markeres det at den er under udvikling, hvorefter den ved færdiggørelse rykkes til gennemsyn og til sidst står som afsluttet.
Til hver iteration har gruppen gennemgået 2 sprints, som i tilsammen løber op i 4 sprints.

- **Sprint 1**
  I første sprint delte gruppen sig op i 3 par, hvor hvert par fik tildelt en række opgaver. Første iteration bestod hovedsageligt i at få opbygget skelettet af programmet og et udkast til GUI'en, hvilket er målet med sprintet.

- **Sprint 2**
  I andet sprint delte gruppen sig op i 2, hvorpå en gruppe stod for implementering af at skrive til og læse fra tekst filer. Den anden gruppe stod for at tilføje funktionalitet til at modtage input og vise output i til brugeren fra GUI'en. Implementering af dette gav os et kørende program, som kan skrive og læse fra tekst filer, som opfylder målet der var sat for Sprint 2 og 1. iteration.

- **Sprint 3**
  I sprint 3 gik projektet fra 1. iteration til 2. iteration, hvor det endelige mål er at få skiftet persistenslaget ud fra tekst filer til en relationel SQL database. Målet med dette sprint var at få designet og opsat en online database, og omstruktureret dele af koden, med implemetereing af facader og interfaces.

- **Sprint 4**
  Til sidst har vi sprint 4, vis mål er at stå med et endeligt produkt, som kan skrive og

læse fra databasen, og opfylder en tilfredsstillende mængde af de fastsatte krav. Sprint 4 gik derfor primært på at få systemet til at skrive til og læse fra databasen.

4 Hovedtekst

I dette afsnit dokumenteres det faktiske arbejde, samt de opnåede resultater fra både 1. og 2. iteration.

4.1 Overordnet krav

De overordnede krav er udformet på baggrund af den stillede case fra TV 2, samt et opfølgende interview med Morten Lehm, som er udvikler hos TV 2. De overordnede krav blev sat under arbejdet i inceptionsfasen, men er sidenhen tilpasset og revideret, derfor indholder dette afsnit en opdaterede udformning af de overordnede krav, samt prioritering. Disse funktionelle krav afspejler brugmønstrende, hvilket de derfor skal forstås som brugmønstre lige vel som krav.

ID	**Funktionelle Krav**	**Beskrivelse**	**MoSCoW**
F/B 01	Registrering af kreditering	Det skal være muligt for producere og firmaer at tilføje programmer og kreditere medvirkende hertil.	Must
F/B 02	Bruger system	Prototypen af forbrugersystemet skal gøre det muligt for forbrugeren at få informationer om krediteringer og produktioner.	Must
F/B 03	Søgning på kreditering	Det skal være muligt at slå en person op og derved se samtlige programmer en person har været med i.	Must
F/B 04	Oprettelse af personer og programmer	Der skal implementeres en skabelon, som skal udfyldes, når der skal oprettes en person eller et program.	Must
F/B 05	Rediger krediteringer	Det skal være muligt at redigere i eksisterende krediteringer.	Must
F/B 06	Anvendelse af data	Det bør være muligt for TV 2 at eksportere en specifik mængde af data i forskellige formater som XML og CSV, så TV 2 nemt kan anvende dette data på anden vis.	Should
F/B 07	Nem tilgang til kreditering	Efter endt show vises en kode på skærmen, som kan slås op i programmet og krediteringene til det pågældende program vises	Should

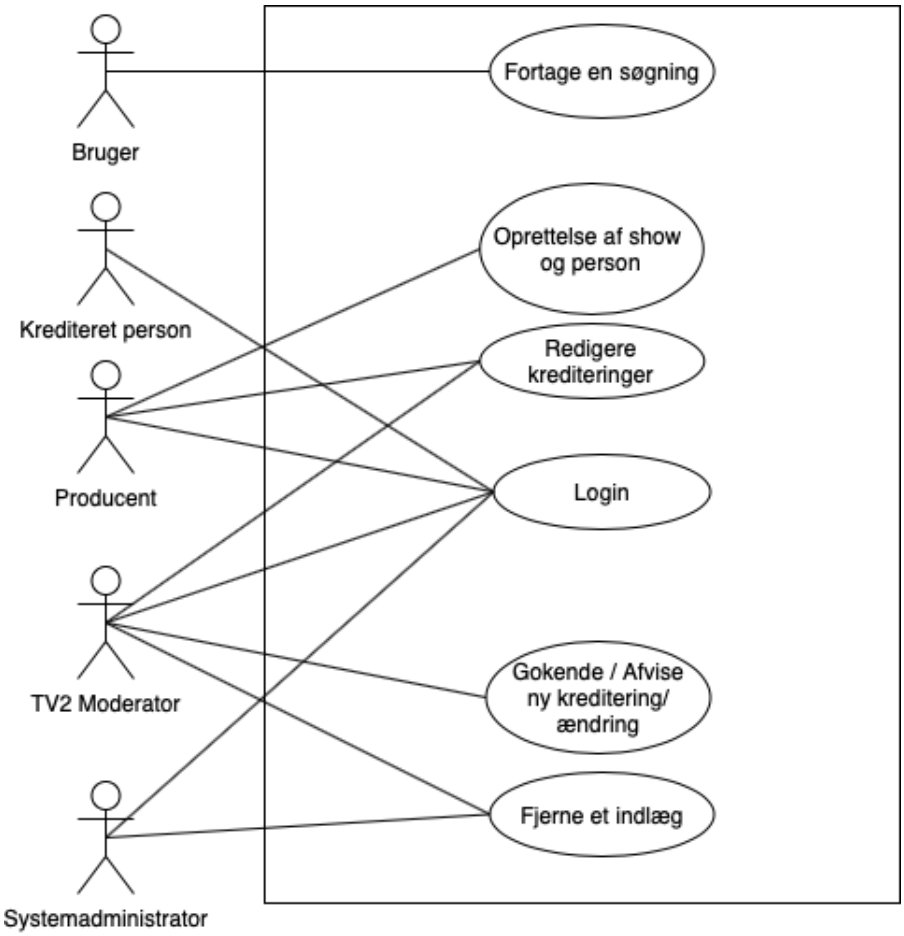
F/B 08	Mulighed for at skelne mellem personer og firmaer	Hver person og firma har sit eget unikke ID, hvis det er for svært at skelne på ID, kan der forsøges at hente yderligere oplysninger som f.eks. telefonnummer, kaldenavn, alder.	Should
F/B 09	Fleksibel søgning	Der behøves ikke vælges, hvad en bruger søger efter, men programmet kan matche inputtet med alle mulige entries (uanset om det er program eller person). Brugeren skal også have mulighed for eksplicit at vælge hvilken type (person, program osv.)	Should
F/B 10	Adgangskontrol	Der skal implementeres en form for adgangskontrol, for hvem der skal have adgang til redigering, oprette og slette data.	Should
F/B 11	Rapportering	Det bør være muligt for TV 2 at eksportere en specifik mængde af data i forskellige formater som XML og CSV, så TV 2 nemt kan anvende dette data på anden vis.	Could
F/B 12	Valg af sprog	Det bør være muligt at skifte mellem sprog, som minimum dansk og engelsk.	Could
F/B 13	Notifikationer	Hver gang der sker noget i databasen skal TV 2's medarbejder modtage en notifikation med ændringer.	Would

Tabel 2: Funktionelle krav - Revideret prioriteringer i MoSCoW

Ovenfor i tabel 2 ses den endelige prioritering af krav. Siden inceptionsfasen er "Adgangskontrol"blevet nedprioriteret, da arbejdet med andre funktioner synes mere vigtig og giver større værdi for produktet, dog er en alternativ login funktion implementeret for at skelne mellem aktørenes rettigheder. Derudover er kravet om redigering tilføjet som et "Must"krav.

4.1.1 Brugsmønster diagram

Ud fra kravene/brugsmønstrende opstilles et brugsmønster diagram, som viser interaktionenerne mellem aktører og brugsmønstre. Da det ikke er alle krav/brugsmønstre, som er implementeret i produktet, ses nedenfor et brugsmønster diagram (Figur 5) over de opnåede funktioner.



Figur 5: Opdateret brugsmønster diagram

Ovenfor ses et opdateret brugsmønster diagram i Figur 5, med en række aktører. Aktørene er her opstillet i hierarkisk rækkefølge fra den almene bruger til systemadministratoren, dvs. at alle aktører har de samme rettigheder som den overstående. I dette diagram findes enkelte "overflødige"aktører, dette skyldes at der er tiltænkt bestemte rettigheder til disse aktører, som er blevet nedprioriteret og ikke er implementeret i denne udgave. Disse overflødige aktører er listet op i diagrammet da de stadig er en del af de alternativ login system, og derfor stadig eksistere i programmet.

4.1.2 Supplerende krav

Ud over de funktionelle krav og brugsmøsntrende har gruppen udarbejdet end række supplerende krav, som er med til at udforme de ikke-funktionelle krav. Disse krav er udformet på baggrund af casen stillet af TV 2, og enkelte krav til design, som er stillet af SDU.

Opnået	
3-lags arkitektur	Det ønskes at programmet følger en 3-lags arkitektur.
Oppetid	TV 2 har et krav om at systemet har en oppetid på 99,7 - 99,8% .
Svartid	TV 2 Ønsker at svartiden på systemet skal være så lav som muligt.
Delvist opnået	
Sikkerhed	TV 2 stiller et meget stærkt krav om, at der er styr på, at al data i systemet er korrekt, da data skal anvendes for at sikre, at krediterede personer får den betaling, de har ret til.
Unikke personer	Der er et krav om, at man skal kunne differentiere personer med f.eks. identiske navne.
Ikke opnået	
Integration med andre systemer	TV 2 har et ønske om at programmet skal kunne integreres med en række af de systemer, de anvender i forvejen. Eksempler kunne være EPG og login-systemet til TV 2 Play.

Tabel 3: Supplerende krav

I Tabel 3 overfor er det supplerende krav, som blev udformet tilbage i inceptionsfasen. Dog er ikke alle krav er opnået i den forventede udstrækning. Kravet om sikkerhed er placeret under delvist opnået, da funktionen om at krediteringer indtastet af producenter, skal godkendes af en TV 2 moderater er implementeret. Dog er et aktiv login system ikke implementeret, og derfor vurderes det at kravet ikke er opfyldt fuldt ud, øvrige supplerende krav er uddybet i afsnit 4.2.1.

4.2 Detaljeret krav

Ud fra de overordnede krav er enkelte krav/brugsmøsntre udvalgt til en mere detaljeret beskrivelse. Brugsmønstrene er udvalgt på baggrund af vigtigheden for projektet, derfor er det brugsmøsntrene "registrering af kreditering"og "søgning på kreditering", der er medtaget i rapporten.

Brugsmønster: Registrering af kreditering.
ID: B01
Primære aktører: Producenter og firmaer (Opretter)
Sekundære aktører:
Kort beskrivelse: Producent for showet eller en SoMe-ansvarlig fra et firma kan logge ind på systemet og oprette showet, ved at udfylde en skabelon.
Præconditioner Man skal være logget ind som enten producent eller firma

Hovedhændelsesforløb

1. Producent eller firma logger ind på systemet.
2. Opretteren navigerer til siden med oprettelse af show.
3. Opretteren udfylder skabelon til oprettelse af show som indbærer:
    - Navn på show og evt. serie ID.
    - Dato for første udsendelse.
    - Navn på producent.
    - kategori.
    - mm.
4. Krediteret personer tilføjes til show.
    - Opretteren skal kunne hente personer, som eksisterer i databasen og tilføje dem til showet.
    - Ved tilføjelse af ikke eksisterende person: se alternativt hændelsesforløb.
5. Den udfyldte skabelon registreres og sendes til databasen som "ikke godkendt", hvorefter en TV 2 moderator kan tilgå krediteringen og godkende den.

Postkonditioner: Showet står som "ikke godkendt"indtil en ansat fra TV 2 har kvalitetstjekket indholdet og godkendt det.
Alternative hændelsesforløb: Skridt 4. Ved tilføjelse af personer, der ikke allerede eksisterer i databasen, får opretteren mulighed for tilføjelse af den nye person ved et pop-up vindue, for yderligere specificering se B02 - Oprettelse af person.

Tabel 4: Detaljeret brugsmønster over B03

Brugsmønster: Søgning på kreditering
ID: B03
Primære aktører: Bruger
Sekundære aktører:
Kort beskrivelse: Brugeren bliver mødt af et søgefelt, hvor de indtaster deres ønskede søgekriterier og vælger herefter hvilken kreditering, de ønsker at få vist
Præconditioner: Brugeren har et søgekriterie i form af navn på show/person eller ID

Hovedhændelsesforløb: 1. Brugeren bliver mødt af et tomt søgefelt 2. Brugeren indtaster et søgekriterie i form af navn på show/person eller ID 3. Brugeren søger med de givne søgekriterier 4. Brugeren får vist en liste af krediteringer, der matcher søgekriterierne 5. Brugeren vælger her den ønskede kreditering 6. Brugeren får nu vist den specifikke kreditering som de har valgt
Postkonditioner: Logiklaget har leveret informationer til brugeren, og brugeren har adgang til de krediteringsinformationer de ønsker
Alternative hændelsesforløb: • Skridt 5: Hvis brugeren ikke kan finde den ønskede kreditering kan de fortage endnu en søgning eller opgive

Tabel 5: Detaljeret brugsmønster over B01

4.2.1 Supplerende krav

En detaljeret beskrivelse af de supplerende krav kan med fordel stilles op ved hjælp Furps+ model. Med Furps+ kategoriseres kravene efter hvilke område de vedrører.

- **Reliability**
  Under dette punkt placeres de krav, som bliver sat til pålideligheden af programmet. I vores tilfælde har vi 2 krav, som omhandler oppetid og sikkerhed for programmet.

Oppetid	TV 2 har et krav om at systemet har en oppetid på 99,7 - 99,8% .
Sikkerhed	TV 2 stiller et meget stærkt krav om, at der er styr på, at al data i systemet er korrekt, da data skal anvendes for at sikre, at krediterede personer får den betaling, de har ret til.

Tabel 6: Supplerende krav - Reliability

- **Performance**

Til dette punkt har vi et enkelt krav, som ikke er yderligere specificeret end, at programmet skal have en svartid der er så lav som muligt.

Svartid	TV 2 Ønsker at svartiden på systemet skal være så lav som muligt.

Tabel 7: Supplerende krav - Performance

- **Plus**

  Design constraints: Her er det vi placere de specifikke begrænsning vores system er pålagt. Kravet om 3-lags arkitektur placeres her da det er et specifikt krav til systemet stillet af SDU.

3-lags arkitektur	Det ønskes at programmet følger en 3-lags arkitektur.

Tabel 8: Supplerende krav - Design constraints

Implementation requirements: Implementerings krav stiller retningslinjer for programmørenes implementering, og de blot skal forholde sig til standarderne eller noget specifikt. Her har vi et krav stillet TV 2, som omhandler måden hvorpå en person registreres i systemet. Det skal her være muligt at differentiere mellem personer med identiske navne. Dette vurdere vi til at være delvist opnået, da det er muligt at oprette personer med identiske navne, og de vil få tildelt unikke ID'er. Dette er dog ikke tydliggjordt for brugeren, hvorfor vi mener den er delvist opnået.

Unikke personer	Der er et krav om, at man skal kunne differentiere personer med f.eks. identiske navne.

Tabel 9: Supplerende krav - Implementation requirements

Interface requirements: Interface requrements kigger på om der nogle interfaces, som programmet skal kunne på eller med. Til dette projekt har TV 2 haft et ønske om at programmet skal interagere med deres øvrige programmer, dette har dog ikke været muligt at udfører i denne udgave af produktet.

Integration med andre systemer	TV 2 har et ønske om at programmet skal kunne integreres med en række af de systemer, de anvender i forvejen. Eksempler kunne være EPG og login-systemet til TV 2 Play.

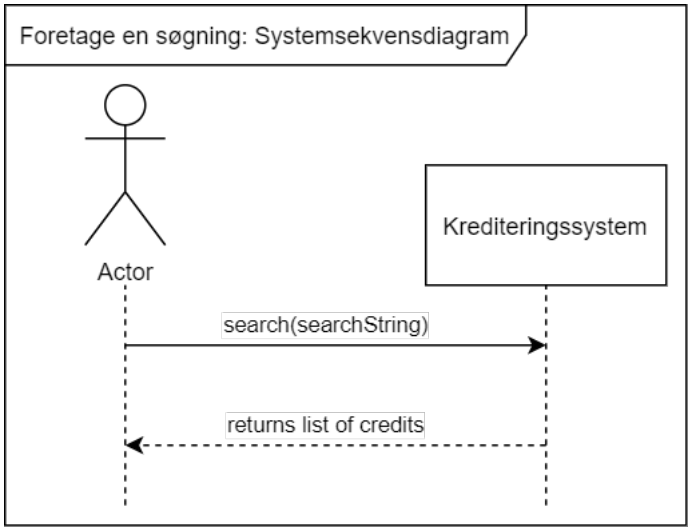
Tabel 10: Supplerende krav - Interface requirements

4.3 Analyse

Dette kapitel beskriver analyse, med tilhørende beslutninger og resultater.

4.3.1 Brugsmønsterrealisering

B03: Foretage en søgning Først foretages en brugsmønsterrealisering på brugsmønsteret B03 foretage en søgning. Først dannes der et systemsekvensdiagram. Dette giver et overblik over hvad målet er. Systemsekvensdiagrammet for B03 ses på firgur 6



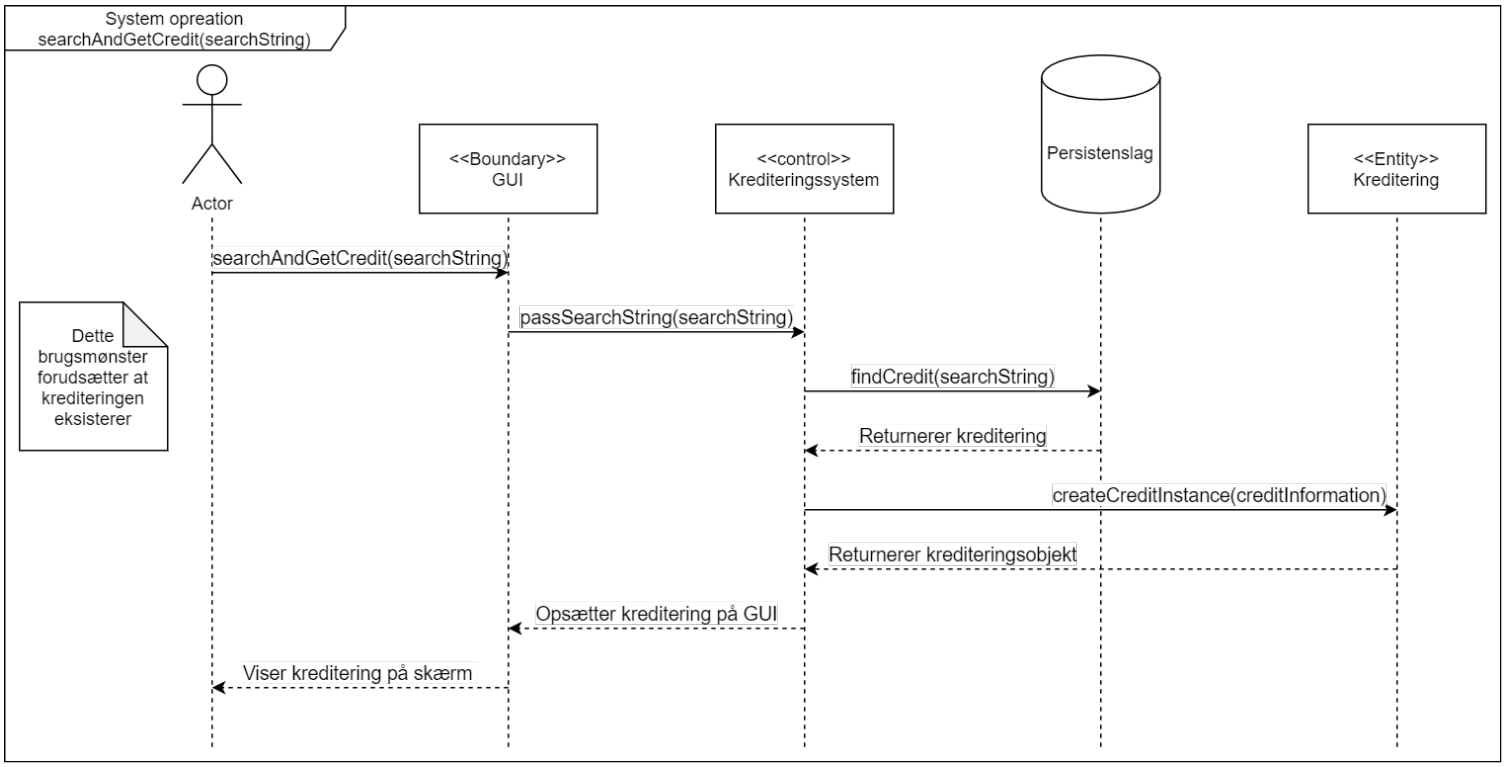
Figur 6: Systemsekvensdiagram for search

Firgur 6 viser at en aktør, her en bruger, bruger funktionen search på krediteringssystem som derefter returnerer en liste af krediteringer. Ud fra dette diagram blev der opsat en operationskontrakt over hvilket ansvar search operationen har. Denne operationskontrakt ses på tabel 11

Kontrakt	
Operation	search(searchString)
Refererer til	Brugsmønster: Foretag en søgning
Ansvar	Ansvaret for denne operation er at at modtage en forespørgsel fra en bruger og vise de krediteringer der svarer til forespørgslen, for brugeren hvis: • Der eksisterer krediteringer der matcher brugerens forespørgsel • Den ønskede kreditering er godkendt af TV 2 moderatorer Den liste som brugeren får vist, vil indeholde en mængde af krediteringer der passer til deres forespørgsel, og kunden kan her vælge en af disse krediteringer, og få yderligere information om den
Prækonditioner	Datebasen indeholder data
Postkonditioner	Brugeren får vist en liste af krediteringer som matcher forespørgslen.

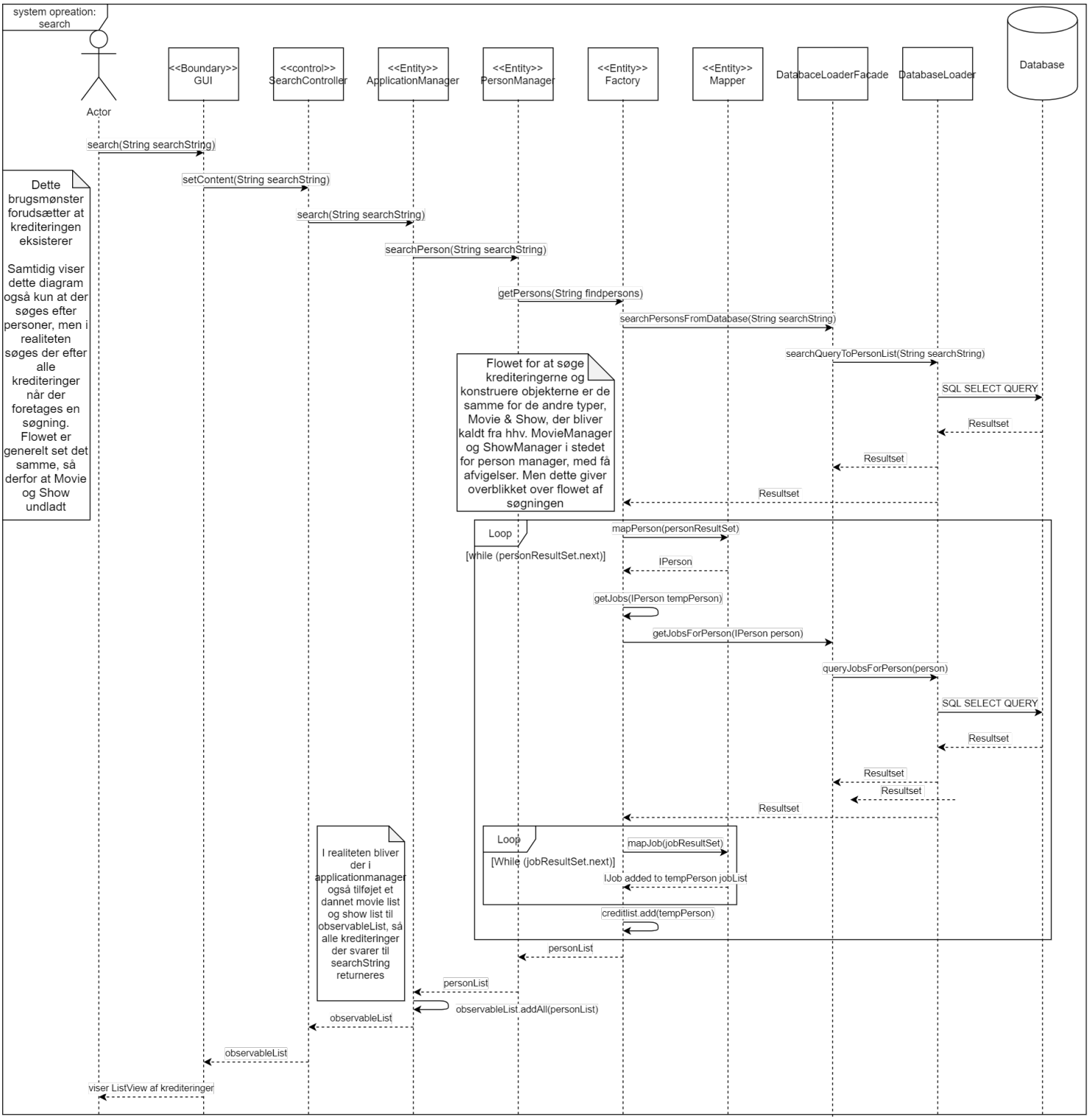
Tabel 11: Operationskontrakt for operationen search

Tabel 11 fortæller at ansvaret for search operationen er at finde de krediteringer der matcher brugerens forespørgsel, og returnere dem så de kan ses. Dette kræver at der er data det matcher i databasen som samtidig er godkendt af en TV 2 moderator. Herefter udarbejdede gruppen et sekvensdiagram for operationer inde i systemet. Først på figur 7 er resultatet for iteration 1, og derefter på figur 8 resultatet for iteration 2



Figur 7: Sekvensdiagram for operationer på ved searchAndGetCredit i iteration 1

Iteration 1 var en simpel udgave af programmet og havde ikke vanvittig meget funktionalitet. Figur 7 viser operations sekvensdiagrammet for søgefunktionen searchAndGetCredit der bliver kaldt på GUI delen. GUI'en kalder Krediteringssystem der var en kompleks og stor klasse med meget logik, og sender søge strengen videre. Den kalder så en funktion findCredit på en DatabaseLoader i peristenslag som henter data ind fra in txt fil og returnerer det til Krediteringssystemet. Krediteringssystemet danner så en instans af det ønskede objekt ved at bruge en af de klasser der ligger under Kreditering. "Kreditering"på diagrammet skal forstås som være en gruppe af specifikke krediteringer, som person, film, serie osv. Dette objekt returneres så til systemet, og herefter til GUI'en som viser det for aktøren. Dette var flowet i 1 iteration, som gruppen endte med at være utilfreds med, og gennemgik derfor en større refactoring process i iteration 2.



Figur 8: Sekvensdiagram for operationer på ved search

Operations sekvensdiagrammet på figur 8 viser hvilke klasser som systemet går igennem for at lave en søgning. Det starter med at aktøren indtaster sine søgekriterier og trykker søg på GUI'en. Dette kalder funktionen search som tager en string som parameter. Search kalder funktionen setContent på SearchController klassen, som har til ansvar at få en liste fra searchString og vise den liste på GUI. SearchController kalder så funktionen search på application manager, som tager searchString og søger efter de forskellige typer af krediteringer. I dette eksempel vises kun søgningen efter personer, men flowet er det samme for film og serier, og derfor er de ikke medtaget i diagrammet. Application manager kalder searchPerson på PersonManager. PersonManager står for mange af funktonerne på objekter af typen Person. Personmanager kalder derefter getPersons på Factory. Factory er den klasse der står for at afgøre objekter der skal konstrueres og hvor data skal hentes fra. Factory kalder så searchPersonsFromDatabase på DatabaseLoaderFacade og sender igen searchString videre. DatabaseLoaderFacade er den simple indgang til databseLoader, hvor Facaden kalder funktionen searchQueryToPersonList stadig med searchString som parameter. Denne funktion queryer så databasen ved at finde de tupler der minder som søgekriteriet searchString. Databasen returnerer derefter et ResultSet af tupler, som sendes tilbage til Factory gennem DatabaseLoader, og DatabaseLoaderFacade. Her begynder konstruktionen af objekterne. Factory starter et loop, der slutter når der ikke er flere tupler i ResultSettet. For hver tupel, kalder Factory mapPerson på Mapper og sender en tupel som parameter. Metoden mapPersons tager denne tupel, og instantierer et person-objekt som en IPerson. En produktion som denne person har været med på kaldes et job. Factory sender den nyligt instantierede person til databasen (ved samme flow som perons) og får returneret et ResultSet af Jobs der har personens personID. Factory starter igen et loop over alle tupler i ResultSettet og tilføjer et instantieret job til en Arrayliste som sættes til personens jobs variabel. Her er personen konstrueret færdig, og personen tilføjes til et liste af krediteringer, creditList. og loopet kører igen. Når der ikke er flere tupler, returneres creditList, først gennem PersonManager, til ApplicationManger der tilføjer listen med personer til en generel list af Krediteringer. Her vil ApplicationManager så søge efter de nadre typer af krediteringer, film og serier, og tilføje dem til listen også. Når dette er gjort, laves listen om til en ObservableList som returneres til SearchController som sætter et ListView på GUI til at vise denne Observable list, og her får aktøren så vist listen af krediteringer der svarer til deres søgning.

Yderligere brugsmønsterrealisering De yderligere brugsmønsterrealiseringer er lagt i bilag **(RET BILAG HER!!!!!!!!!)**

4.4 Design

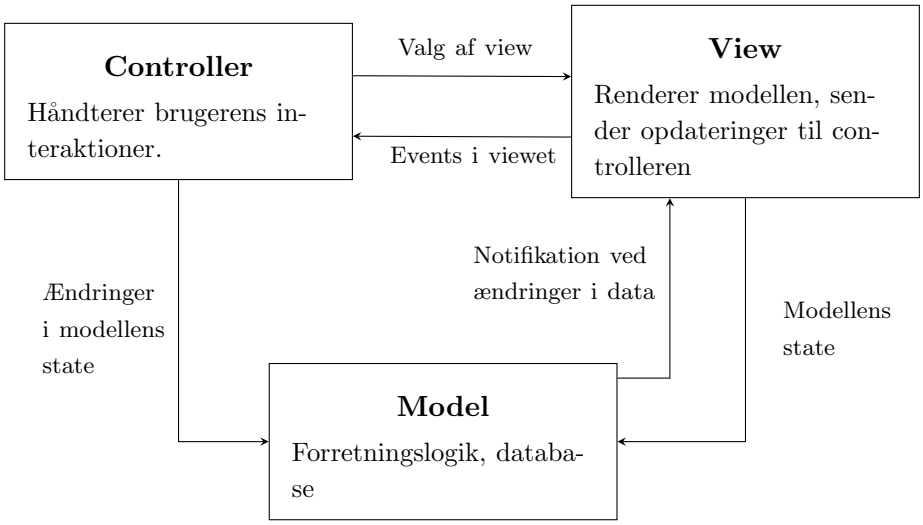
Formålet med dete kapitel er at planlægge løsningen, ud fra de krav der der er stillet, sammenholdt med erfaringerne fra analysen.

4.4.1 Trelagsarkitektur

Kravet om en trelagsarkitektur (IF-01) er markeret om et "must"i prioriteringen af krav. I FURPS+ analysen er kravet placeret som en "Design constraint", da dette stiller et krav til designet af systemets arktitektur.

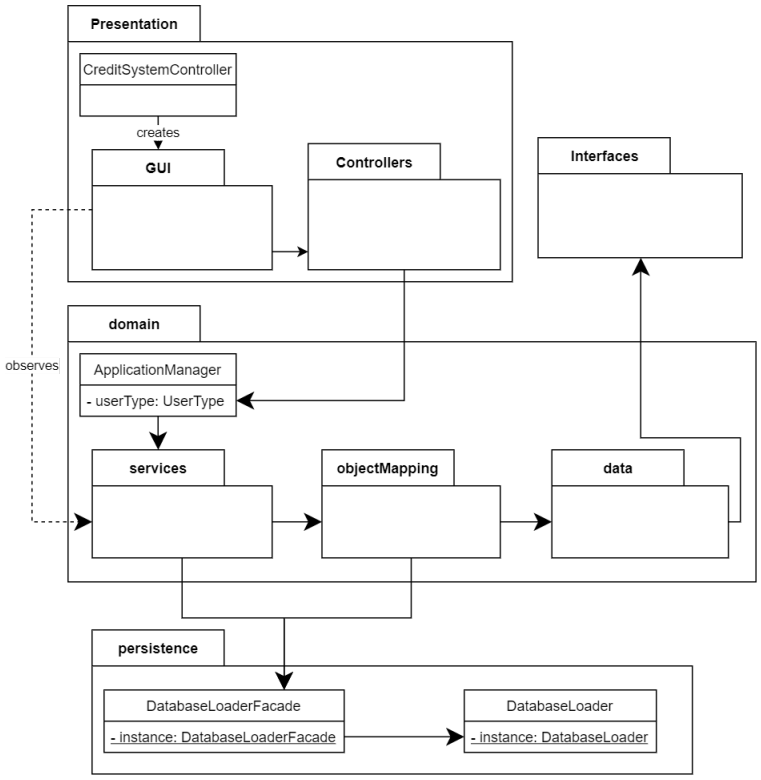
Interfaces Et oplagt værktøj til at opdele lagene i trelagsarkitekturen er anvendelse af interfaces. Med interfaces i mellem lagene skal hver enkelt lag kun afhænge af det specificerede interface. På denne måde er der ingået en *kontrakt* i mellem de to pågældende lag. Ved anvendelse af disse interfaces behøves det lag der anvender interfacet ikke at kende til selve implementatationen af interfacet.

Separation mellem presentationslaget og domænelaget Et designmønster der er en oplagt som en del af trelagsarkitekturen er designmønsteret 'Model-View-Controller'. Dette designmønster er struktureret således:



Dette designmønster separerer interaktion og præsentation fra systemets data. Det vil sige at præsentations- og domænelaget separeres. View *observerer* data der holdes i modellen. View præsenterer det data der holdes i modellen til brugeren. Dette gør at dataen kan ændre sig uafhængigt af præsentationen, siden præsentationslaget udlukkende observerer dataen. Controlleren står for at respondere på der sker i view, fx hvis en bruger klikker på et element der skal starte et nyt view, da står den for det sætte det nye view til at blive vist. Den står også for at håndtere hvad der gøres når der klikkes på et given element i viewet. Modellen repræsenterer de to andre lag, forretningslogiklaget og persistenslaget, og står derved for håndtere og manipulere med data.

Pakkediagram Systemet er delt op i pakker efter præsentations-, domæne- og persistenslag. Ved denne opdeling kan det nemmere sikres at trelagsarkitekturen overholdes. Et pakkediagram blev opsat og kan ses på figur 9



Figur 9: Pakkediagram for systemet

Figur 9 viser hvordan pakkerne kommunikerer med hinanden. Klassen CreditSystemController, viser en GUI for brugeren. Når brugeren foretager en handling, knytter GUI sig til nogle controllers i pakken Controllers som kalder metoder på ApplicationManager i domænelaget. ApplicationManager kalder den passende Manager i pakken Services som kan gøre en af to ting:

- Hvis brugeren har oprettet en ny kreditering i GUI er objektet allerede instantieret, og manageren kalder derfor en 'put' metode på DatabaseLoaderFacade i persistensLaget, som derefter kalder den passende metode i DatabaseLoader.
- Hvis brugeren har søgt efter en kreditering, kalder manageren først pakken objectMapping, som herefter kalder en
- Hvis brugeren har oprettet en ny kreditering i GUI er objektet allerede instantieret, og manageren kalder derfor en 'put' metode på DatabaseLoaderFacade i persistensLaget, som derefter kalder den passende metode i DatabaseLoader.

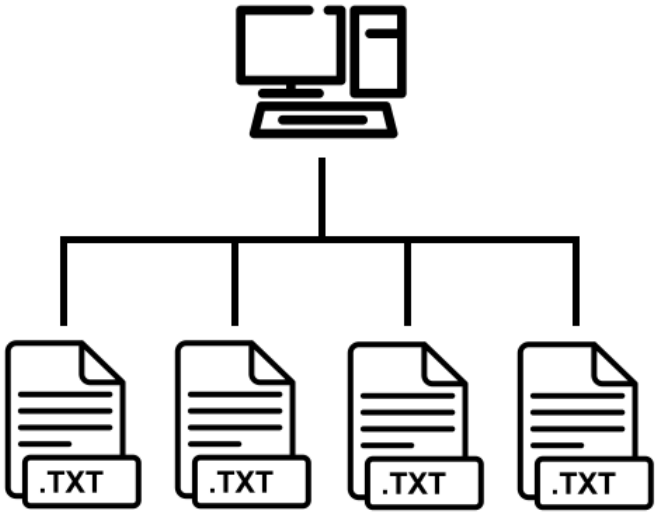
- Hvis brugeren har søgt efter en kreditering, kalder manageren først pakken objectMapping, som herefter kalder en 'get' funktion på DatabaseLoaderFacade som dernæst kalder den passende metode på DatabaseLoader der queryer databasen, og returnerer resultaterne. Disse resultater får objectMapping pakken, og instantierer objekter via data pakken i domain.

4.4.2 Klassediagram

4.4.3 Systemsekvensdiagrammer

4.5 Database Design

Databasen bestod i første iteration af individuelle TXT filer som blev brug som CSV filer. Dette lod os dele dataen op i de forskellige typer af objekter vi gerne ville have opbevaret, f.eks. Episode, Person og Movie, så hver type fik sin egen fil. På den måde undgik vi at indlæse alt vores data når vi skulle have fat i noget specifikt, og dermed spare ressourcer i form af tid og lagring.



I anden iteration er der gjort brug af en SQL relationel database, på baggrund af kravene til anden iteration.

4.6 Implementering

4.7 Test

5 Diskussion

Målet for projektet var at udvikle en prototype til et program, der skal erstatte de klasse rulletekster efter et tv-show, for TV 2. TV 2 udformet en case beskrivelse med en række af krav, som er blevet udvidet og prioriteret af gruppen i første fase af projektet. Kravene blev prioriteret efter MoSCoW modellen, hvorpå en række af kravene blev kvalificeret som et "Must"krav. "Must"kravene er de nødvendige krav, som skal opfyldes for at vi mener produktet har substans. Disse krav er alle blevet opfyldt og fungere efter hensigten. Dernæst har vi en rækken af krav under "Should", som ikke har været i højeste prioritet. En tilfredsstillende del af disse krav er opfyldt eller delvist opfyldt.
Med de opfyldt og delvist opfyldt krav, har gruppen formået at udvikle en kørende program, som tillader en producer at tilføje og redigere produktioner i form af film eller serier. Til

med kan en producer oprette og registrer personer, og tilføje dem til de produktioner de har medvirket til udviklingen af, og tildele dem roller og evt. karakternavn. Alt dette føjer programmet til en online database, hvorefter en medarbejder fra TV 2, har muligheden for at kontrollere producentens indtastninger og godkende, før det bliver tilgængeligt for den almindelige bruger. Den almindelige bruger kan søge på person og produktioner, og finde tilhørende oplysninger, på en nem og brugervenlig måde.
Gruppen havde forventet at opnå enkelte krav, som ikke er blevet en realitet. Her er der tale om kravet omkring adgangskontrol, som denne udgave af prototypen indholder et alternativ til. Hvilket også medfører at en systemadministrator ikke har nogle betydelige funktioner, denne var tiltænkt til håndtering af de øvrige brugers rettigheder/roller. Derudover var det ønsket at en krediteret person skulle have mulighed for at redigere sine egne oplysninger, hvilket heller ikke er muligt. Til slut havde gruppen et ønske om en funktion, der tillader at hente krediteringer i forskellige formatter, som f.eks. XML og CSV.
Selvom gruppen ikke har formået at implementere alle de ønskede funktioner, er det endelige resultat dog stadig tilfredsstillende. Kvaliteten af de implementerede funktioner, er gennem udviklingsarbejdet blevet prioriteret højere end at opfylde samtlige krav af laver kvalitet. Skulle der arbejdes videre på systemet er det bygget op på en sådan måde, at det ikke kræver nogen omstrukturering for at implementere et evt. loginsystem til adgangskontrollen.

6 Konklusion

7 Perspektivering

8 Procesevaluering

Referenceliste

[1] Ukendt-forfatter. *Kantar seer-undersøgelse*. http://tvm.tns-gallup.dk/tvm/pm/2020/pm2003_Consolidated.htm. Last accessed 26 May 2021.

[2] TV2 og SDU. *Credits Management, A case by TV 2 Denmark A/S*. https://docs.google.com/document/d/1p6lQjWV76TX9uTLst2OAdmV07XfMRn5fObelzBpd2EI/edit. Last accessed 26 May 2021.

A Oversigt over kildekode

B Brugervejledning

C Samarbejdsaftale

D Vejlederaftale

E Projektlog

F Udfyldt rapportkontrolskema

Kapitel	krav	Opfyldt +/-
Forside	Projekttitel, uddannelsesinstitution, fakultet, institut, uddannelse, semester, kursuskode, projektperiode, vejleder, projektgruppe og projektdeltagere (fornavn, efternavn, sdu-email). Må gerne have illustrationer.	
Titleblad	Samme oplysninger som på forsiden, samt afleveringsdato og projektdeltagernes underskrifter (Projektdeltagernes aktive deltagelse i projektforløbet anerkendes gensidigt ved projektdeltagernes underskrifter). Må ikke have illustrationer.	
Resumé	• En kort introduktion til projektet - hvad blev der arbejdet med og hvorfor. • Problemformuleringen og vigtige afgrænsninger. • Metode - hvordan angreb I problemet og hvordan realiserede I løsningen (hvem, hvad, hvornår og hvorfor) • Hovedresultater og konklusioner – hvad kom der ud af arbejdet. (max 1 side)	
Forord	Hensigten med rapporten, målgruppe, forhistorie, anerkendelser.	
Indholdsfortegnelse	Samlet indholdsfortegnelse for hele projektrapporten. Højst to eller tre niveauer i indholdsfortegnelse (der kan evt. være flere i selve rapporten). Afsnit på niveau 1 og 2 skal være nummererede.	
Læsevejledning	Vejledning i hvordan rapporten kan læses, eksempelvis i form af hvilken rækkefølge afsnittene kan læses i, og hvordan sammenhængen er mellem de forskellige dele af rapporten, herunder mellem hovedrapport og bilag. Rapportens målgruppe.	

<table>
<tr><td>Redaktionelt</td><td>Skriveprocessen og ansvarsområder i skriveprocessen.
Ansvarsområder kan fx beskrives på fx følgende form:
<table><tr><th>Afsnit</th><th>Ansvarlig</th><th>Bidrag fra</th><th>Kontrolleret af</th></tr><tr><td>Afsnit a</td><td>Person a</td><td>Person b</td><td>Person a, b, c</td></tr><tr><td>Afsnit b</td><td>Person b</td><td>Person a</td><td>Person a, b, c</td></tr><tr><td>Afsnit c</td><td>Person c</td><td>Person b</td><td>Person a, b, c</td></tr></table></td><td></td></tr>
<tr><td>Indledning</td><td>Projektets rammer og baggrunden for projektet. Resume af udleverede case. Problemformulering og afgrænsninger. Formål og mål med projektet. Problemformulering og afgrænsninger. (indledningen må gerne includere materiale direkte fra inceptionsdokumentet, det skal blot have en tydelig reference)</td><td></td></tr>
<tr><td>Faglig vidensgrundlag</td><td>Begrebsdefinitioner, teori og fagligt vidensgrundlag.</td><td></td></tr>
<tr><td>Metode og planlægning</td><td>Metode: Benyttede metoder i projekt (hele projektet). Kombination af UP og Scrum. Fordele og ulemper</td><td></td></tr>
<tr><td></td><td>Planlægning: Plan for elaborationsfasen og de enkelte iterationer. Prioritering af krav i planlægningen. Det faktiske udviklingsarbejde. Faserne, iterationerne og det faktiske arbejde i dem? Scrum: backlogs, roller, begivenheder, scrum-buts.</td><td></td></tr>
<tr><td>Hovedtekst
(skal indeholde resultater både fra iteration 1 og fra iteration 2)</td><td>Overordnede krav: Opdateret resume af overordnede krav fra inceptionsdokument inklusive overordnet brugsmønsterdiagram og oversigt over supplerende krav.</td><td></td></tr>
<tr><td></td><td>Detaljerede krav: Detaljeret brugsmønsterdiagram (hvis relevant). Detaljerede brugsmønsterbeskrivelser. Detaljerede beskrivelser af supplerende krav, fx organiseret efter FURPS+.</td><td></td></tr>
<tr><td></td><td>Analyse: Overvejelser, beslutninger og resultater vedr. analysemodellen, inklusive både den statiske og den dynamiske side af analysemodel.</td><td></td></tr>
</table>

	Design: Overvejelser, beslutninger og resultater vedr. Softwarearkitektur og detaljeret design, herunder design af persistens.	
	Databasedesign: Overvejelser, beslutninger og resultater vedr. tabeldesign og SQL-forespørgsler.	
	Implementering: Overvejelser, beslutninger og resultater vedr. konvertering fra design til kode illustreret gennem udvalgte centrale eksempler, samt andre vigtige implementeringsbeslutninger. Implementering af database.	
	Test: udførte test samt resultatet af dem.	
Diskussion	Hvad er der opnået og hvad er der ikke opnået i projektet i forhold til det forventede som beskrevet i indledningen. Hvad er styrkerne og svaghederne ved resultaterne. Kunne I have opnået bedre resultater?	
Konklusion	Opsummering af resultaterne og diskussionen af dem. Svar på problemformuleringen.	
Perspektivering	Er den fundne løsning brugbar i anden sammenhæng? Hvad bidrager løsningen og den opnåede viden til. Fremtidigt arbejde (næste skridt i projektet, hvis I havde mere tid).	
Procesevaluering	Processen og gruppens refleksion over processen: Læringsprocessen, teamroller, samarbejdet internt i gruppen og med vejleder, projektarbejdsformen, arbejdsformer, metoder, skriveprocessen, den tidsmæssige styring af projektet,ledelse af projektet, arbejdsfordeling i projektet m.m. Hvordan ville I gribe arbejdet an, hvis I skulle starte forfra?	
Referenceliste	Litteratur angivet på en anerkendt form. (Alle former for litteratur som bøger, artikler og hjemmesider) Kildehenvisninger i teksten. Materiale som gruppen ikke selv har fremstillet i dette projekt skal være angivet med kilde! Alle kildehenvisninger i teksten skal være anført på samme måde. Kildeangivelser på figurer, grafer etc. som projektgruppen ikke selv har frembragt.	

Bilag	A. Oversigt over kildekode B. Brugervejledning C. Samarbejdsaftale D. Vejlederaftale E. Projektlog F. Udfyldt rapportkontrolskema G. Inceptionsdokument I. (Andre Bilag)	

Rapporttekniske elementer		
Layout	Er der anvendt samme layout i alle kapitler. Er layout overskueligt/harmonisk.	
Sprog	Formidler rapporten projektet faglig og sagligt. Er sproget neutralt, aktivt, upersonligt, konkret, præcist, kortfattet og korrekt. (Procesevalueringen må benytte personligt sprog)	
Sidenummerering	Er der korrekt og konsistent sidenummerering i rapporten.	
Figurer/diagrammer	Er alle figurer konsekvent nummererede. Er der figurtitel og figurtekst til alle figurer. Er figurtitler og figurtekster dækkende og afklarende. Er figurerne tydelige og læsbare. Er figurerne informationsgivende og i den rette sammenhæng.	
Tabeller	Er alle tabeller konsekvent nummererede. Er der en forklarende tabeltekst til alle tabeller. Er alle søjler og rækker forsynet med parametre. Er der enheder på alle relevante rækker og søjler.	
Sporbarhed af begreber	Er der en konsekvent brug af samme betegnelse for et givet begreb igennem rapporten.	

G Inceptionsdokument