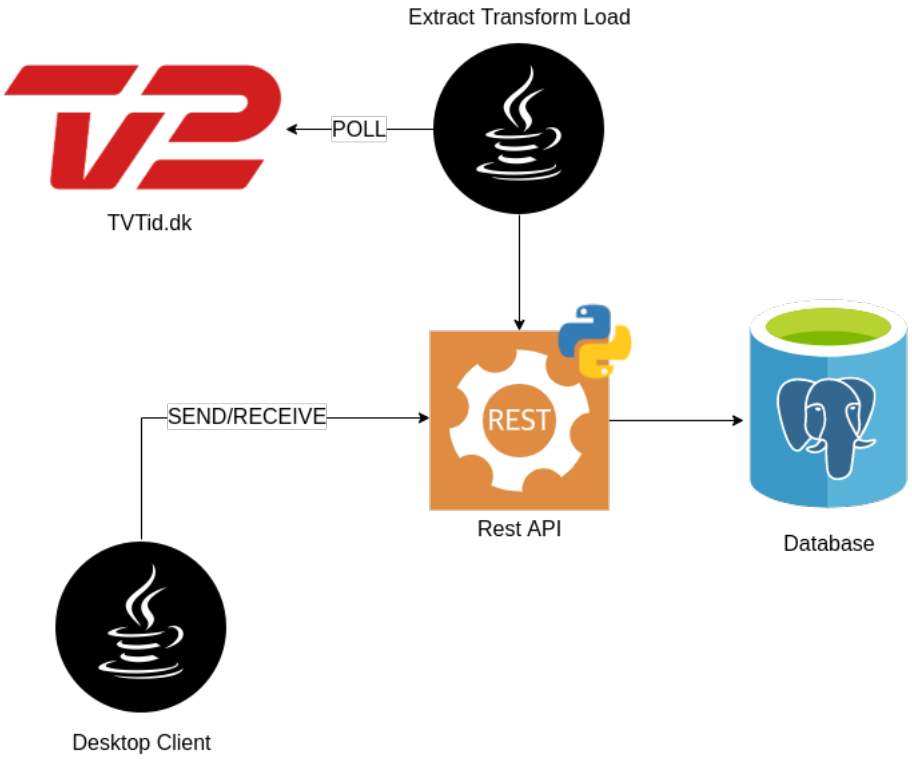
Creditoro

Et krediteringssystem



Softwareteknologi

Semesterprojekt 2. semester, ST2-PRO

Projektperiode: 01.01.2020 - 29.05.2020

Afleveringsdato: 29.05.2020

Projektgruppe 06:

Jakob Rasmussen, jakra19@student.sdu.dk

Kenneth M. Christiansen kechr19@student.sdu.dk

Kevin K. M. Petersen, kepet19@student.sdu.dk

Kristian N. Jakobsen, kjako19@student.sdu.dk

Mathias N. Rasmussen, mara816@student.sdu.dk

Simon Jørgensen, sijo819@student.sdu.dk

Vejleder: Henrik Lykkegaard Larsen, hlla@mmmi.sdu.dk

Syddansk Universitet
Det Tekniske Fakultet
Mærks Mc-Kinney Møller Instituttet
Campusvej 55, 5230 Odense M

Title:	Creditoro
Institution:	Syddansk Universitet Det Tekniske Fakultet, Mærsk Mc-Kinney Møller Instituttet Campusvej 55, 5230 Odense M
Uddannelse:	Softwareteknologi
Semester:	2. Semester
Semestertema:	Udvikling af cyber-physical softwaresystemer
Kursuskode:	ST2-PRO
Projektperiode:	01.01.2020 - 29.05.2020
ECTS:	10 ECTS
Vejleder:	Henrik Lykkegaard Larsen
Projektgruppe:	06

Jakob Rasmussen, jakra19@student.sdu.dk

Kenneth M. Christiansen, kechr19@student.sdu.dk

Kevin K. M. Petersen, kepet19@student.sdu.dk

Kristian N. Jakobsen, kjako19@student.sdu.dk

Mathias N. Rasmussen, mara816@student.sdu.dk

Simon Jørgensen, sijo819@student.sdu.dk

Antal sider: sider
Bilag: 1 bilag

Ved at underskrive dette dokument bekræfter hvert enkelt gruppemedlem, at alle har deltaget lige i projektarbejdet, og at alle således hæfter kollektivt for rapportens indhold.

I Resume

I dette projekt bliver der arbejdet ud fra TV2's problemstilling omhandlende muligheden for at flytte krediteringer fra TV til en anden platform. Denne plads kan så udfyldes med andet som fx. reklamer og promoveringer for egne programmer. Gruppen har valgt at bygge en fuldt funktionelt prototype, inklusive Rest API og EPG poller. Dertil er gruppen fundet frem til en problemformulering der omhandler udviklingen af et krediteringssystem, hvordan krediteringerne skal gøres tilgængelige samt håndteres. Derudover undersøges det hvordan systemet kan indeholde krediteringerne. Projektgruppen har valgt at afgrænse projektet ved at lave en prototype til et færdigt program. Ideelt ville systemet også have en webside, dog er dette vurderet til værende udenfor projektets scope.
Overordnet benyttes UP (Unified Process) i hele projektet. UP opdeler hele projektet i 4 faser: *Inceptionsfasen* hvor vigtige krav og kristiske risici identificeres. *Elaborationfasen* hvor en iterativ udvikling af krav, design, analyse og test konstrueres ud fra kravspecifikationer. *Konstruktionfasen* hvor systemet konstrueres - den faktiske kodeudformning. *Overgangsfasen* hvor det undersøges om systemet er færdigt og ikke har mangler.
I starten af projektets analysefase er et brugsmønsterdiagram, herunder aktørliste og brugsmønsterliste, udformet. Dette er med til at give et billede af hvordan systemet skal bygges op, samt hvilke aktører der skal kunne interagere med hvilke brugsmønstre. Til prioritering af brugsmønstrene er der foretaget en MoSCoW-analyse. Til identificering af ikke-funktionelle krav er der brugt metoden FURPS. Her er der sat krav op der omhandler Functionality, Usability, Reliability, Performance og Supportability. For yderligere specificering af ikke funktionelle krav er FURPS+ benyttet - FURPS med ekstra specificerende kategorier med til at opfylde kundens behov.
Til projektstyring benyttes Scrum til at fordele og få overblik over de forskellige opgaver, der opdeles i mindre issues. De forskellige arbejdsopgaver inddeles i sprints som forløber sig over en given periode - i dette projekts tilfælde to uger.
Hovedresultater og konklusioner - hvad kom der ud af arbejdet

II Forord

Denne rapport er skrevet i forbindelse med 2. semesterprojekt i Softwareteknologi på Syddansk Universitet i samarbejde med TV2, der ønsker at lave et krediteringssystem der skal flytte krediteringer fra TV til en anden platform. Denne plads kan så erstattes af reklamer og promoveringer for andre programmer.
Rapporten beskriver gruppens arbejdsform og fremgangsmåde for projektet, samt de konklusioner gruppen er kommet frem til. Hensigten med rapporten er at dokumentere arbejdsprocessen henimod udviklingen af et system til håndtering af krediteringer.
En særlig tak skal lyde til vejleder Henrik Lykkegaard Larsen samt Troels Mortensen for hans videoserie på YouTube omhandlende MVVM (Model - View - View Model).

God fornøjelse med læsningen.

Jakob Rasmussen
Kenneth M. Christiansen
Kevin K. M. Petersen
Kristian N. Jakobsen
Mathias N. Rasmussen
Simon Jørgensen

Indhold

III Læsevejledning

Rapporten er disponeret sådan, at *afsnit 1 - 3* beskriver projektets udgangspunkt. Herunder findes det essentielle problem rapporten er udformet ud fra, de benyttede metoder og værktøjer igennem hele projektet, planlægningen og strukturering samt det faglige vidensgrundlag.
Afsnit 5-6 indeholder kravspecifikationer herunder udarbejdelsen af de overordnede krav. Derudover kan de detaljerede krav findes her.
Analysen af projektet findes i *afsnit 7 - Analyse*, hvor der bliver udformet en brugsmønsteranalyse og brugsmønsterrealisering samt et resumé af konstrueringen af EAST-ADL modeller ud fra funktionelle og ikke funktionelle krav.
I *Afsnit 8-9* kan der læses om projekts designfase. Herunder findes systemets design samt systemets subsystemer. Det er her også muligt at se designafsnittet for databasen.
I *afsnit 10 - Implementering* kan der læses om konverteringen fra design til kode, samt vigtige implementeringsbeslutninger. Dette afsnit dækker hele systemet inklusiv subsystemer.
Afsnit 11 - Test indeholder tests af en række udvalgte metoder og klasser fra systemet i henholdsvis 1. og 2. iteration.
Rapporten rundes af med *afsnit 12 - 15*, hvor diskussion, konklusion, perspektivering samt procesevaluering er at finde. I disse afsnit vil der blive fokuseret på, hvad der opnået i projektet, hvad der kan gøres bedre, opsummering af resultater, samt en evaluering over hele projektets proces.
Vejleder- og samarbejdsaftale vil være at finde i *bilag* **??** og **??**.
Kun udvalgte modeller, tabeller og diagrammer er indsat i rapporten, resten vil være at finde i *bilag*.
Ønskes det at læses om systemets subsystemer - herunder Rest API og EPG Poller - vil dette være at finde i *bilag*. Her findes analyse, design, implementering og tests af de to subsystemer.

IV Redaktionelt

I tabel **??** ses de ansvarsområder gruppen har opdelt skriveprocessen i. Af tabellen fremgår hvilke gruppemedlemmer der har haft ansvaret for enkelte afsnit i rapporten, hvem der har bidraget til skriveprocessen af afsnittet og hvem der har kontrolleret afsnittet.

Afsnit	**Ansvarlig**	**Bidrag af**	**Kontrolleret af**
Forside	Kevin	Kenneth	
Titelblad	Kevin	Kenneth	
Resumé	Jakob & Mathias		
Forord	Mathias & Jakob		
Læsevejledning	Mathias	Jakob	
Indledning	Kenneth		
Metoder & Værktøjer	Kenneth		
Planlægning	Jakob		
Faglige Vidensgrundlag	Kenneth		
Overordnede Krav	Mathias		
Detaljerede Krav	Mathias		
Analyse	Jakob		
Design	Kevin & Simon		
Databasedesign	Kenneth	Kristian	
Implementering	Kristian & Simon		
Test	Kristian & Simon		
Diskussion	Mathias & Jakob		
Konklusion	Mathias & Jakob		
Perspektivering	Mathias & Jakob		
Processevaluering	Kenneth & Simon	Alle	Alle
Bilag A - Oversigt Over Kildekode	Kevin		
Bilag B - Brugervejledning			
Bilag C - Samarbejdsaftale	Alle		Alle
Bilag D - Vejlederaftale	Alle		Alle
Bilag E - Projektlog	Kevin & Kenneth		
Bilag F - Rapportkontrolskema			
Bilag G - Inceptionsdokument	Alle		Alle
Bilag H - Andre Bilag			

Tabel 1: Ansvarsområder i skriveprocessen

1 Indledning

Når et program bliver broadcasted på en TV station skal krediteringer vises. Dette gøres i slutningen af programmet, i maksimalt 30 sekunder. Det betyder, at der ikke altid er tid til at vise alle krediteringer, og derfor prioriteres de før de vises.
Hvis de 30 sekunder for hvert program kunne frigøres, kan danske TV stationer bruge tiden på at vise noget andet, som f.eks. reklamer og promovering af eget indhold. Derved har TV 2 mulighed for at øge deres årlige indtægter med op til 60 millioner kroner.
TV 2 har brug for et system, der kan administrere krediteringer for programmer produceret i Danmark. Hertil skal der kunne tilføjes nye krediteringer i systemet for nye produktioner, samt det skal være muligt at kunne søge efter eksisterende krediteringer. Det skal være muligt at kunne se hvilken rolle en given person har haft i en produktion, da denne person kan have haft flere forskellige roller på flere forskellige produktioner.

1.1 Projektrammer

Denne sektion har til formål at opridse rammerne for projektet, samt hvilket område projektgruppen arbejder indenfor.

1.1.1 Krav til Projektet

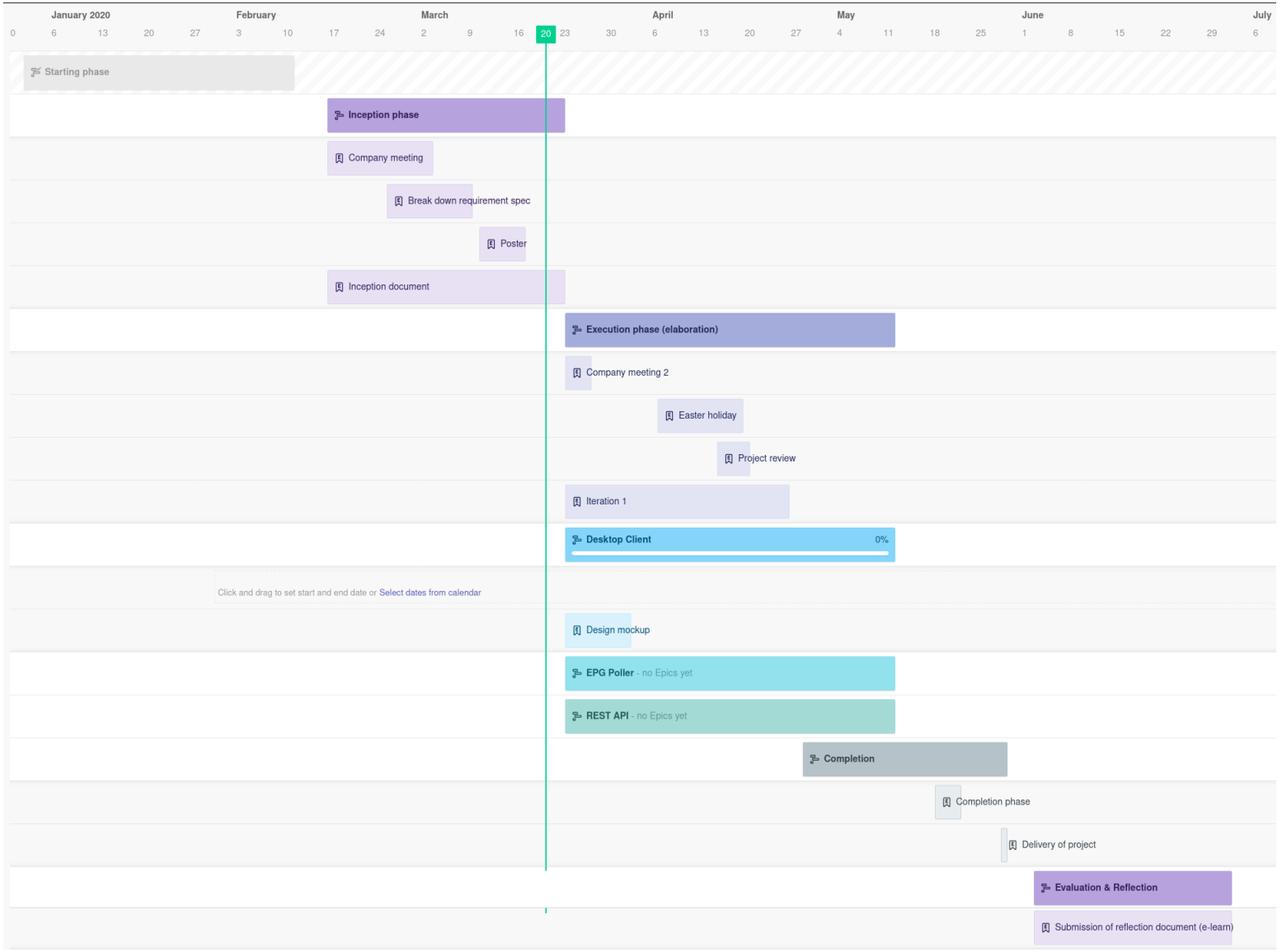
Systemet skal så vidt muligt skrives i programmeringssproget Java.
Krediterings-data skal lagres i en database, og i dette projekt skal den brugte database være SQL baseret. Der skal bruges PostgreSQL.
Systemet forventes ikke at være et færdigt system, men en række forslag til løsninger der opfylder systembehovet. Forslagene skal inkludere:

- Krav
- Analyse
- Design
- Implementering
- Test

Producere der kan tilføje og redigere i krediteringerne, skal kun have mulighed for at redigere i de produktioner, de selv ejer.
Det forventes at krediteringssystemet er kompatibelt med andre systemer (f.eks. fra Stofa, YouSee etc.).

1.1.2 Tidsplan

Tidsplanen har til formål at skabe overblik og styring over projektet. Den giver gruppemedlemmerne et overblik over, hvornår de forskellige dele af projektet skal starte og slutte, og derved bliver det hurtigt klart hvis tidsplanen skrider.



Figur 1: Tidsplan for projektet

1.1.3 Igangsættende Problem

TV2 ønsker at frigøre 30 sekunders krediterings tekster efter hvert program, så de i stedet kan bruge tiden på at vise reklamer. Problemet består i at disse krediterings tekster, så skal vises på en anden platform. I tabel **??** ses kravene fra TV2's projektcase:

Beskrivelse	**Type**
"Vi har brug for et krediterings system der kan håndtere dansk TV content"	En vag opgave
"Dette inkluderer muligheden for at oprette nye krediteringer i systemet, når en ny produktion bliver lavet, samt at have mulighed for at søge efter en given produktion og få en liste af krediteringer, forbundet til denne. Det burde også være muligt at se hvilken rolle en given person har haft i en produktion, eftersom en person kan have flere forskellige roller i forskellige produktioner."	Ønske om en bestemt løsning
"Producers/TV-stationer burde være i stand til at redigere krediteringer for programmer/produktioner de ejer. De burde også være i stand til at redigere disse produktioners ID. Systemadministratorer skal kunne vedligeholde (oprette, læse, opdatere og slette) personer, krediteringer og personer."	Ønske om en bestemt løsning
"Til slut skal systemet kunne offentliggøre en service som andre systemer kan bruge. Disse systemer kan f.eks. være en hjemmeside eller en applikation. Disse andre systemer skal også kunne bruge API'et, så data'et kan blive brugt i allerede eksisterende systemer (såsom TVTID.dk - TV 2's TV-Guide)."	Ønske om en bestemt løsning
"En form for adgangskontrol skal implementeres, til de beskyttede dele af systemet (oprettelse, opdatering, slettelse, osv. af data)	Ønske om en bestemt løsning
"Der skal være en offentligt tilgængelig del af systemet, hvor det er muligt at se krediteringer uden at logge ind."	Ønske om en bestemt løsning
"Nuværende løsning er begrænset til 30 sekunder, og dermed kan alle krediteringerne ikke altid vises i praksis"	Et problem

Tabel 2: Krav fra TV2s projektcase

1.1.4 Identifikation af Problemet

Som det er nu bliver krediteringer vist i slutningen af et program. Ifølge reglerne for visning af krediteringer, må krediteringer ikke vises mere end 20 sekunder for produktioner under 60 minutter, og 30 sekunder for produktioner over 60 minutter. Dette giver en del problemer. For det første betyder den begrænsede varighed, at ikke alle medarbejdere kan krediteres. Dette ender ud i at der skal prioriteres i krediteringerne, før de bliver vist på TV. Derved får alle

medarbejdere ikke den anerkendelse de burde. Hvis krediteringer flyttes til et eksternt system, og derved ikke bliver vist på TV, kan man undgå at skulle prioritere. Alle kan derved få den fortjente kredit. Derudover vil det også give mulighed for at vise noget andet, som f.eks. reklamer eller promoveringer for eget indhold. [**?**]

Et sådan eksternt system vil også hjælpe med oprettelsen af nye krediteringer, ved at gøre processen hurtigere og nemmere, samt mere overskueligt. Dertil har gruppen valgt at arbejde med samtlige/alle problemstillinger givet af TV2 i tabel **??**, og lave en prototype til et funktionsdygtigt system. Angående valget med at arbejde med samtlige problemstillinger præsenteret af TV2, har gruppen konkluderet det som værende realistisk jævnfør figur **??**. Denne prototype vil kunne bruges som et udkast til et endeligt system.

1.2 Formål med Projektet

Projektet har til formål at gøre gruppemedlemmerne i stand til at tilrettelægge og gennemføre udviklingen af en softwareapplikation i en agil udviklingsproces. Den udvilklingsproces der tages i brug i dette projekt, er Unified Proces. I tæt sammenspil med den agile udvilkingsmetode SCRUM, opnåes viden om, hvordan man går fra objektorienterede systemmodeller til implementering af kode. Ydermere er formålet, at gruppen opnår viden omkring databaser, og hvordan man udarbejder et databasedesign, laver databaseforespørgelesr og bruger disse i applikationen.

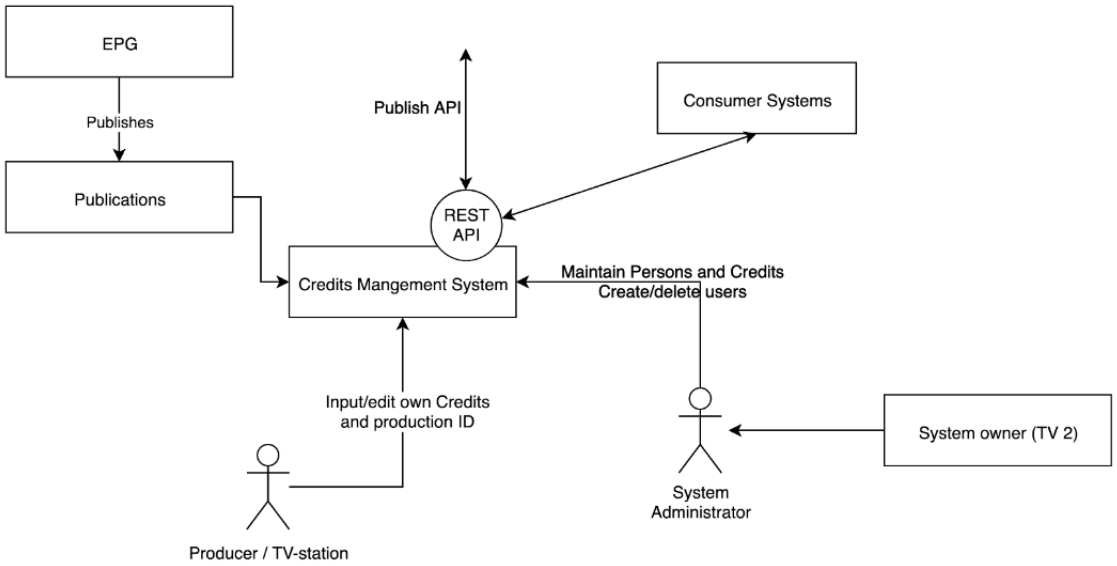
1.3 Problemformulering & Afgrænsning

Ud fra overstående er projektgruppen er kommet frem til følgende problemformulering med tilhørende underspørgsmål:

Hvordan kan vi udvikle et samlet krediteringssystem, der giver mulighed for at erstatte rulletekster efter et endt program?

1. Hvem skal kunne håndtere krediteringer?
2. Hvordan skal krediteringerne gøres tilgængelige, og hvordan skal seerne refereres dertil?
3. Hvordan kan man oprette et system som kan indeholde krediteringer?

Projektgruppen har valgt at afgrænse dette projekt, ved at konstruere en prototype til et system. Det er blevet valgt at lave et system der ligger tæt op ad det oprindelige foreslag fra TV2s projektcase, hvor systemtegningen kan ses på figur **??**, og et sådan system vil indebære alle kravene i tabel **??**.



Figur 2: Foreslag til systemtegning - © TV2

I et produktionsklart system vil det være ideelt at have en webside, men dette har vi konkluderet som værende uden for projektet.

2 Metoder & Værktøjer

Dette afsnit har til formål at beskrive de metoder og værktøjer der er blevet brugt i løbet af udviklingen af produktet og udarbejdelsen af rapporten.

2.1 Metoder

2.1.1 Brugsmønsterdiagram

Det første skridt i udviklingen af et brugsmønster er at finde og definere de forskellige aktører der vil interagere med systemet. En aktør kan defineres som alt der kommunikerer med systemet og ikke selv er en del af systemet. Et eksempel på dette kunne være en kunde på en webshop. Disse aktører opstilles i en tabel sammen med de brugsmønstre hver aktør kan tilgå.

Da kravindsamling er en evolutionær aktivitet, bliver alle aktører ikke nødvendigvis identificeret i første iteration. Det er muligt at identificere primærer aktører i løbet af første iteration, og først senere i forløbet blive i stand til at identificere sekundære aktører, når man får mere viden om systemet. *Primære* aktører interagerer med systemet for at opnå påkrævede systemfunktioner, og ud fra det, få noget ud af at bruge systemet. *Sekundære* aktører støtter systemet så de primærer aktører kan gøre deres arbejde. Når aktørerne er fundet kan brugsmønstrene findes. Et brugsmønster angiver et scenarie en aktør kan interagere med.

Når både aktører og brugsmønstre er fundet, kan man opstille et brugsmønsterdiagram for at give en visuel forståelse for hvilke aktører der kan tilgå hvilke brugsmønstre. Et eksempel på et brugsmønsterdiagram kan ses på figur **??**

Brugsmønstermodellen hjælper udvikleren med at forstå brugeren, så systemet kan konstrueres. Idet der er et samlet billede af hvordan systemet skal se ud, vil udviklingen være mere målfast da kravene og deres forhold til brugeren er klare.

2.1.2 FURPS+

FURPS er en model til klassificering af ikke-funktionelle krav, og er med til at give en detaljeret beskrivelse af kravene. Akronymet står for:

Functionality: Hvad kunden vil have. Dette inkluderer også sikkerhedsforanstaltninger.

Usability: Hvor effektivt er produktet fra brugerens synspunkt? Er produktet æstetisk acceptabelt? Er dokumentationen fyldestgørende?

Reliability: Hvad er den mest acceptable system nedetid? Er systemfejl forudsigelige? Er det muligt at demonstrere hvor præcise resultaterne er? Hvordan bliver systemet gendannet?

Performance: Hvor hurtigt skal systemet være? Hvad er den maksimale responstid? Hvad er gennemløbet? Hvor meget hukommelse bruger systemet?

Supportability: Kan systemet testes? Er det muligt at konfigurere systemet, udvide det, installere det, og yde service på systemet.

+-tegnet står for supplerende behov kunden kan have, og omfatter:

Design constraints: Har I/O enheder eller database management systemer indflydelse på hvordan softwaren skal opbygges?

Implementation requirements: Er det nogle standarder programmørerne skal overholde? er testdrevet udvikling nødvendigt?

Interface requirements: Hvilke downstream feeds skal der laves? Hvilke andre systemer skal systemet samarbejde med?

Physical requirements: Hvilken hardware skal systemet implementeres på?

2.1.3 MoSCoW

MoSCoW er en vigtig prioriteringsmodel indenfor softwareudvikling, da den beskriver hvilke dele af systemet der som minimum skal laves før produktet kan accepteres. De brugsmønstre, udvilkerne af systemet skal tage udgangspunkt i, bliver prioriteret med kunden, så det bliver klart hvilke dele af systemet der skal udvikles først. Disse bliver opstillet i en MoSCoWmodel, så der let skabes et overblik. MoSCow er et akronym der står for:

Must have - betyder skal have og er det som der minimum skal være med for at softwaren virker og kan accepteres af kunden.

Should have - betyder det som burde være med og det kunden gerne vil have med.

Could have - betyder det som kunne være med, hvis der er tid nok.

Won't have (this time) - betyder det som der ikke skal prioriteres nu, men måske i en anden iteration.

2.1.4 UP & Scrum

UP
'UP' er en forkortelse af 'unified process' og består af fire faser. De fire faser er beskrevet i den rækkefølge som de udføres.

Inceptionsfasen er der for at finde ud af om projekt overhovedet kan gennemføres, bestemme hvilket anvendelseområde systemet har og identificere vigtige krav og kritiske risici.

Elaborationfasen er for at lave en iterativ udvikling af de forskellige krav, design, analyse og test ud fra den overordnede kravspecifikation og den prioritering der er lavet. I Elaborationfasen vil der blive brugt Scrum.

Konstruktionfasen, hvor fokus vil være på udvikling af komponenter og andre funktioner. I fasen vil der blive brugt UML til at identificere hvilke klasser og komponenter der skal være. Det er i denne fase kodeningen kommer til at ske og den første iteration af software produktet.

Overgangfasen, hvor der vil være fokus på at få et færdigt softwareprodukt. Det vil man gøre ved, at se om man har implementeret de aftalte funktioner og i dialog med kunden finde ud af om de er tilfredse med produktet.

Scrum
Scrum er en agil udvilkingsmetode der benyttes til at lede og kontrollere leverancer af produkter. Scrum består af 3 faser:

1. Forberedelse
2. Eksekvering
3. Idriftsættelse

I forberedelsesfasen er der 3 aktiviteter: Produktvision, der er en overordnet beskrivelse af løsningen og dens omfang, product roadmap, der er en overordnet plan for hvornår vigtige funktioner forventes leveret, release plan, der er en inddeling af product roadmap i en eller flere udgivelser, hvor den første udgivelser består af de minimumsfunktioner der er beskrevet i MoSCoW modellen.

I eksekveringsfasen er der tre artefakter og tre ceremonier. Artefakterne består af product backlog, der er en samling over alle krav for systemet, sprint backlog, der består af de krav gruppen skal implementere i den kommende sprint, burndown chart, der er en visualisering af gruppens fremskridt. Ceremonierne består af sprint planlægning, hvor planlægningen af de enkelte sprints finder sted, dagligt stand up, der er et dagligt koordineringsmøde hvor gruppemedlemmerne snakker om hvad de har lavet siden sidste møde og hvad der skal laves til næste møde, sprint review, der er en gennemgang af hvordan sprintet er forløbet og bruges til at lave eventuelle rettelse i planlægningen af næste sprint.

I idriftsættelsesfasen bliver sprintet idrisftsat, hvilket betyder at der bliver frigivet til kunden i form af et nyt softwareprodukt eller en opdatering til en eksisterende produkt.

Scrum vil blive brugt i elaborationsfasen til at nedbryde de krav der blev defineret i inceptionsfasen. Der vil blive benyttet en sprintperiode på 2 uger, og da sprintperioden er kort (normalt bruges 1-4 uger) er det vigtigt at kravene bliver brudt ned til User Stories der kan nåes indenfor 2 sprints. Gruppen vil i projektet benytte værktøjet ZenHub til GitHub for at integrere Scrum ind i projektet. Dette giver mulighed for at samle projekt management og kode på gruppens GitHub siden (https://github.com/creditoro).
Til projektet bliver der benyttet et scrum board med følgende kolonner:

New Issues	Icebox	Backlog	In Progress	Done	Closed
	Issues med lav prioritet	Kommende issues	Igangværende issues	Færdige issues der bliver lukket næste sprint møde	

Tabel 3: Scrum Board

Alle issues er sorteret fra top til bund alt efter prioritet.

2.2 Værktøjer

I tabel **??** ses de værktøjer projektgruppen har benyttet under inceptionsfasen og de værktøjer, der skal bruges fremadrettet.

Værktøj	**Beskrivelse**
PostgreSQL	PostgreSQL er en open-source objekt-relationel database server.
GitHub	GitHub er en web-baseretkollaborations platform henvendt til software udviklere, der gør det muligt at versions-kontrollere projekter.
Overleaf	Overleaf er en online skriveplatoform for **LaTeX**, hvor man kan være flere brugere der skriver samtidig.
UML	Unified Modeling Language
IntelliJ	Integreret udviklings miljø, som primært bruges af gruppens medlemmer.
ZenHub	ZenHub er en platform der gør det lettere at anvende Scrum i praksis.
Scrum Board	Et Scrum Board er et værktøj, der har til formål at gøre opgarverne i Sprint og Backlog synlige og overskuelige.
Pair Programming	Pair programming er en softwareudvilkingsteknik, hvor to programmører arbejder sammen ved én computer.
Klassediagram	Bruges til visuelt at vise hvordan softwaresystemer er opbygget. I diagrammet beskrives systemets klasser, metoder og værdier klassen indeholder, samt klassernes relationer til hinanden.
SonarCloud	Online service til scanne kode for bugs, vulnerbilities og code smells.

Tabel 4: Værktøjer til projektarbejdet

3 Planlægning

Dette afsnit har til formål at fremvise gruppen planlægning af eleborationsfasen sammenlignet med det faktiske arbejde, samt komme nærmere ind på hvordan gruppen har planlagt arbejdet i de enkelte sprints og iterationer, samt rollefordelingen i projektgruppen.

3.1 Rollefordeling i projektgruppen

I projektets opstartsfase tog gruppemedlemmerne en Belbin-test. Resultatet af denne test kan ses i bilag **??** side 26.

3.2 Plan for Elaborationsfasen og Det Faktiske Udviklingsarbejde

Semesterprojektet er delt op i 2 iterationer. Til første iteration er planen at få sat REST API'et og databasen op. Disse skal indeholde vores brugere, kanaler, produktioner, personer og krediteringer så det kan hentes via API'et, som skrives i python og databasen er en relationel database i PostgreSQL. For alle API'ets moduler skal metoderne POST, GET, PATCH, PUT, DELTE og UPDATE implementeres. Disse metoder skal kunne tilgås via koden, så vi nemt og hurtigt kan hente dataet.

Til desktop-klienten skal der i første iteration laves design mockup, til en del af programmet, så gruppen har en idé om hvordan programmet skal se ud. Her skal der laves fxml dokumenterne for login-siden, gennemse kanaler og gennemse produktioner. Der skal også implementeres login, så brugerne kan logge ind via API'et. På siden for "gennemse kanaler"skal kanalerne hentes fra API'et, som er hentet fra TVTid.dk via en EPG Poller. Derved kan det ses hvilke kanaler der skal vises i programmet.

Det er valgt at fokusere på at implementere brugsmønstrene fra MoSCoW-modellen i API'et i første iteration, frem for desktop-klienten. Dette skyldes at metoderne i desktop-klienten skal kalde metoderne til API'et, så brugergrænsefladen nemt kan udskiftes. Dertil er det nået at implementere brugsmønstrene log in, log ud samt at gennemse kanaler i desktop-klienten. I API'et kan der oprettes, slettes, hentes og redigeres brugere, kanaler, produktioner, personer og krediteringer.

I anden iteration implementeres der roller i systemet. Indtil videre har målet være at vise programmet fra systemadministratorens synspunkt, men i anden iteration er planen at brugere nu skal have forskellige roller. Dette indebærer at ikke alle brugere ser samme knapper og information som f.eks. systemadministratoren. Systemadministratoren skal bl.a. have mulighed for at oprette nye kanaler, hvor kanaladministratoren ikke skal.

Der skal laves design mockup til resten af programmet, bl.a. Produktions-siden og kanal siden. Derudover skal der, til anden iteration, kunne trykke på kanalerne og se hvilke produktioner der hører til. Dertil skal der også kunne importeres nye produktioner fra TVtid via EPG Polleren, samt oprette nye produktioner og nye krediteringer hertil.

3.3 Backlogs

Gruppen bruger et værktøj der hedder ZenHub til at administrere Scrum Boardet, som nævnt i afsnit **??**. Det er besluttet at dele hver iteration op i 2 sprints, så en sprint løber over 2 uger.

3.3.1 1. Iteration

Sprint 1, med startdato d. 24. marts, kommer til at bestå af rapportskrivning. Her laves der blandt andet domæneklassediagram, analysemodel og supplerende krav bliver beskrevet. Der lægges også vægt på brugsmønstrene, da disse har indflydelse på resten af projektets forløb.

I sprint 2, med startdato d. 7. april, begynder implementeringen. Her bliver api'et og epg polleren blive implementeret, og dele af desktop-klienten, på baggrund af analyse- og designfasen, som fandt sted før/i første sprint. Der fokuseres på, at det skal være muligt at oprette, slette, hente og redigere brugere, kanaler, produktioner, personer og krediteringer i api'et. EPG polleren skal færdigimplementeres, så der kan hentes data fra TVTid.dk. Til sidst i sprinten bliver der lagt vægt på rapportskrivning, hvor der bliver skrevet på de afsnit der hører til hovedteksten.

3.3.2 2. Iteration

Sprint 3, med startdato d. 28. april, kommer til at bestå af implementering af resterende brugsmønstre (funktioner), som ikke blev implementeret i første iteration. I api'et skal der implementeres brugerroller, så brugernes rettigheder stemmer overens med projektets brugsmønstre. I desktop-klienten skal resterende views, models, og modelviews implementeres, så funktionerne fra api'et kan tilgåes. Dette er blandt andet productionsview, så en bruger har mulighed for at se produktionerne. Slutteligt skal de resterende unittests implementeres.

I sprint 4, med startdato d. 12. maj, skal rapporten skrives færdig. Her kommer der til at være fokus på afsnit som diskussion, konklusion, perspektivering og procesevaluering.

3.4 Ceremonier

De ceremonier der bliver anvendt i dette projektforløb er sprintplanlægning og en alternativ version af de daglige stand-up møde, som bliver benævnt i afsnit **??**. Sprintplanlægningen kommer til at finde sted før hver sprint, og bruges til at skabe et overblik over hvad arbejdet i kommende sprint kommer til at bestå af. Man kan under dette møde aftale, hvordan man har tænkt sig at arbejde, for at nå tidsfristen for sprintet.

3.5 Scrum-buts

Projektgruppen anvender Scrum, men det er besluttet at der ikke skal holdes et dagligt stand-up møde. I stedet afholdes der et stand-up møde hver tirsdag, da dette er projektdagen. Mødet kommer til at foregå som det ellers er ment.

Det er besluttet at der ikke skal afholdes sprint review i slutningen af en sprint, men i stedet gøre det løbende, så gruppemedlemmerne er holdes opdateret gennem implementeringsprocessen.

4 Faglige Vidensgrundlag

Dette afsnit har til formål at dække over den faglige viden gruppen skal have, for at kunne udføre projektet.

4.1 JAVA

At have kendskab til Java er en vigtig forudsætning for udarbejdelsen af projektet. Systemet vil hovedsageligt blive programmeret i sproget Java. Det faglige niveau svarer til 2. semesterstuderende på Softwareteknologi. Dette indebærer blandt andet forståelse af JavaFX og Scenebuilder. Arbejdet med Java i projektet forudsætter derudover forståelse for basale programmeringsprincipper og forståelse for det objektorienterede programmeringsparadigme.

4.2 Python

Et grundlæggende kendskab til Python kræves for at kunne forstå samt implementere REST Api'et.

4.3 Database

Systemet vil indeholde en lang række data, som skal lagres i en database. Det er derfor nødvendigt at have forståelse for databaser, databasestrukturer, relationelle SQL-databaser og SQL-queries. Databasen der vil blive benyttet i projektet er PostgreSQL, en basal viden om databasesproget/programmeringssproget SQL er derfor nødvendigt. Al nødvendig viden er givet i SDU's 'Data Management' kursus pensum.

4.4 JSON

JSON står for JavaScript Object Notation, og det er et åbent standardfilformat og dataudvekslingsformat der bruger tekst til at gemme og transmittere dataobjekter, der består af attribute-value par og array datatyper.

4.5 Ubuntu & Docker

En basal forståelse for Ubuntu (eller andet Linux baseret distro) og Docker kræves for opsætning af REST Api'et og databasen.

4.6 Relevante Eksisterende Løsninger

IMDb
IMDb (Internet Movie Database) er en online database bestående af film, serier, medvirkende m.m. Man har mulighed for at søge efter informationer ved at referere til blandt andet førnævnte titler. IMDb har også et ratingsystem, der gør det muligt at bedømme film, serier etc.
Rotten Tomatoes
Rotten Tomatoes er på lige fod med IMDb, en database for film, serier, medvirkende m.m. Man kan på Rotten Tomatoes også søge information. Rotten Tomatoes distancerer sig fra IMDb, ved både at tage ratings fra sine brugere og et panel af anmeldere.

5 Overordnede Krav

Dette afsnit har til formål at uarbejde de overordnede kravspecificationer for systemet. Kravene er udformet udfra projektcasen, virksomhedsmøderne med TV2 samt gruppens egne tilføjelser. Afsnittet vil indeholde et resume af de overordnede krav, et overordnet brugsmønsterdiagram samt en række supplerende ikke-funktionelle krav.

5.1 Resume af Overordnede Krav

(Taget fra inceptionsdokumentet (se bilag **??** *side 7))*

Systemet afspejler det system TV2 har lagt op til i projektcasen. Der er tale om et system, hvor alle kan se - og nogle kan redigere krediteringer for programmer. Systemet skal kunne tilgås via en dansk brugergrænseflade. Det skal indeholde forskellige brugerroller; systemadministrator, kanaladministrator, producer, royalty bruger og gæst.

Systemadministratoren skal have rettigheder til at gøre alt. Dette gælder f.eks. at oprette krediteringer, kanaladministratorer og producerer. Kanaladministratoren skal kunne redigere, oprette og slette krediteringer for egen kanal. En producer skal kunne tilføje og redigere i krediteringerne for egne produktioner, og en gæst skal kunne se krediteringer for alle programmer. Det skal være muligt at kombinere personer som refererer til den samme person i den virkelige verden. Når to forskellige producere vil oprette en kreditering for et program, skal krediteringen være associeret med en person og vedkommendes rolle. Det betyder altså, at det skal være muligt at oprette personer der kan sammenflettes (f.eks. med UUID).

TV2 har ikke lov til at lagre persondata, såsom et CPR-nummer eller et telefonnummer, så det skal være muligt at identificere personer i systemet og sikre at krediteringerne er korrekt forbundet til de rigtige personer. Databasen skal være søgbar, så det er nemt at finde personer, programmer og lignende. Det er vigtigt at systemet er nemt at bruge, så seerne nemt kan se krediteringerne for det program de lige har set.

TV2 kunne være interesseret i at integrere systemet med andre systemer (YouSee Tv, Boxer Play osv.), og det er derfor vigtigt at systemet er kompatibelt med krediteringer i andre systemer. Det kunne også være interessant at have muligheden for at få notifikationer når noget nyt sker i systemet. Samrådet for Ophavsret og Producentforeningen kunne også være interesseret i at modtage en form for meddelelse hver gang der er blevet tilføjet noget nyt til systemet, hvor de kan godkende krediteringerne og ud fra disse udbetale royalties. Derudover kunne det også være interessant at brugergrænsefladen kunne undestøtte flere sprog.

For at beskytte dele af systemet (tilføjelse/redigering/sletning af data osv.), skal der indføres en form for adgangskontrol. Der skal være en offentligt tilgængelig del af systemet, hvor det er muligt at se krediteringerne for et et program uden at skulle logge ind.

Systemet skal både kunne importere og eksportere data. Importering skal være mulig vha. en EPG Poller, der trækker EPG data via TVTid.dk. Brugere af systemet skal kunne eksportere krediteringsdata til forskellige formater såsom XML og CSV.

I tabel **??** ses kravene opsummeret i en tabel for bedre overblik.

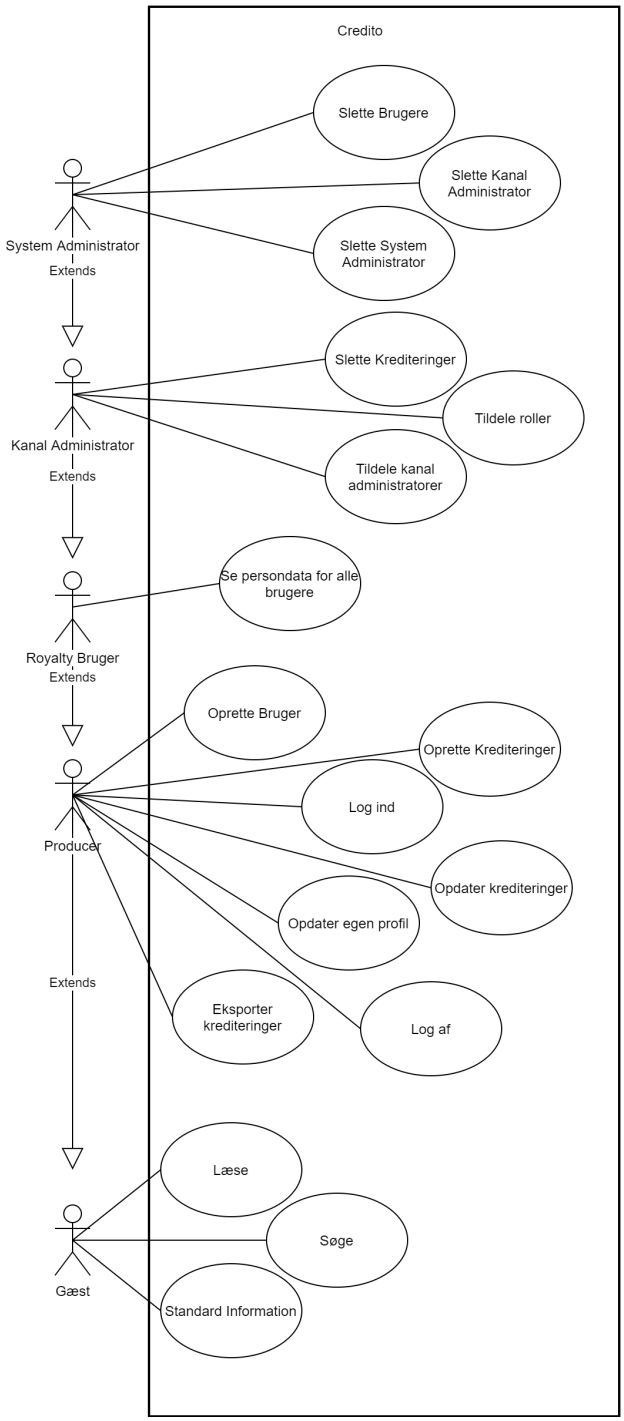
ID	Navn	Beskrivelse
K01	Brugergrænseflade	Systemet skal tilgås via en dansk brugergrænseflade
K02	Brugerroller	Systemet skal indeholde brugerroller
K03	Tildel roller	Kanaladministrator skal kunne tildele producer- og kanaladministrator roller
K04	Slet bruger	Systemadministratoren skal kunne slette brugere
K05	Se krediteringer	Alle skal kunne se krediteringer
K06	Søg efter krediteringer	Alle skal kunne søge efter og se krediteringer for alle programmer
K07	Opret krediteringer	Specielle brugere, kanaladministratore og systemadmin skal kunne oprette krediteringer for et givent program
K08	Rediger krediteringer	Specielle brugere, kanaladmin og systemadmin skal kunne redigere krediteringer for egne programmer
K09	Slet kreditering	Kanaladmin og systemadmin skal kunne oprette/redigere/slette krediteringer under egen kanal
K10	Søg efter personer	Alle skal kunne søge efter personer
K11	Knyt personer til krediteringer	Personer skal kunne knyttes til krediteringer så man kan se hvilke programmer en person har deltaget i Systemadmin, kanaladmin og producer skal kunne se persondata som email og tlf. nr.
K12	Link personer i den virkelige verden	Det skal være muligt at linke personer i krediteringer til personer i den virkelige verden, så der krediteres korrekt
K13	Eksporter data	Brugere skal kunne eksportere data til forskellige formater såsom XML og CSV
K14	Importering af data	Systemet skal kunne importere EPG data via TVTid.dk
K15	Integration	Systemet skal kunne integreres med andre systemer (Yoursee Play, Boxer Play, osv.)
K16	Notifikationer	Systemet skal sende notifikationer til relevante brugere
K17	Sprogvalg	Systemet skal kunne understøtte flere sprog

Tabel 5: Liste af krav fra overordnet kravspecifikation

5.2 Overordnet Brugsmønsterdiagram

I figur **??** ses det overordnede brugsmønsterdiagram (use case model). Diagrammet giver et visuelt overblik over systemets overordnede brugsmønstre og hvilke aktører der kan benytte disse.

I diagrammet ses systemets fem aktører (se aktørliste i bilag **??** side 8) - System Administrator, Kanal Administrator, Royalty Bruger, Producer og Gæst. Diagrammet viser arvehierakiet mellem aktørerne, jo højere oppe på diagrammet aktøreren er placeret i diagrammet, jo flere rettigheder har den enkelte aktør. "System Administrator"har flest rettigheder og er derfor placeret øverst. Ligeledes har "Gæst"færrest rettigheder, og er placeret nederst i brugsmønsterdiagrammet.



Figur 3: Overordnet brugsmønster over Creditoro systemet
Menneskene er aktører.
Cirklerne beskriver handlinger aktørne kan lave.
Pilende betyder Extends, hvilket vil sige aktørerne arver funktionalitet

5.3 Supplerende krav

På baggrund af den udlerede projektcase, samt de to virksomhedsmøder med TV2, er en række ikke-funktionelle supplerende krav blevet udformet. Systemet skal gøre det muligt for aktører med rettighederne dertil (se figur **??**), at kunne kreditere produktionsroller som overholder DRs krediteringsregler samt databeskyttelsesloven (GDPR).

Systemet skal give brugeren, uanset rettighedsniveau, mulighed for at kunne skifte imellem forskellige sprog i brugergrænsefladen. Brugergrænsefladen skal derfor være responsiv, og kunne ændres dynamisk, alt afhængig af brugerens færden.

tilfælde af systemets servere genstarter, starter del-systemerne igen automatisk. Altså genstartes systemets server, vil REST Api, database og EPG Poller automatisk starte op igen. Det vil derfor ikke være nødvendigt at lukke systemet ned, for at foretage back-up.

Systemet skal benytte sig af en database som skal kunne op til 10000 nye brugere, samt 15000 krediteringer årligt uden af de hyppigste kald til REST Api'et får nedsat funktionalitet. Hvilket vil sige med en responstid på mere end 300 ms. Databasen skal kunne overholde dette krav årligt i 25 år. Systemet skal derudover kunne håndtere 5000 brugere inden hvert minut.
Systemet skal være installerbart vha. Docker cia Docker-compose. Det vil være muligt at konfiguere systems indstillinger via en `.env` fil. Det vil i kildekoden være muligt at finde en opsætningsguide.

I tabel **??** ses en oversigt over de ovennævnte ikke-funktionelle krav. Tabellen benytter FURPS modellen.

FURPS	**Krav**
Functionality	Skal kunne kreditere produktionsroller som er angivet af DRs Krediteringsregler.
	Skal overholde GDPR.
Usability	Systemet skal kunne understøtte flere sprog.
	Systemet skal have en responsiv brugergrænseflade (UI).
Reliability	Hvis serveren til systemet genstarter, startes del-systemerne igen automatisk. Der vil ikke være behov for at lægge systemet ned regelmæssigt for at kunne foretage backup.
Performance	Databasen skal kunne håndtere 10000 nye brugere - samt 15000 krediteringerer årligt i 25 år, uden at ofte brugte kald til REST Api'et bliver sløvt (reponsetid på mere end 300 ms).
	Systemet skal kunne håndtere 5000 brugere indenfor et minut.
Supportability	Systemet er installerbart vha. Docker via Docker-compose.
	Det vil være muligt at konfigurere system indstillinger via en `.env` (miljø) fil. En opsætningsguide vil være at finde sammen med kildekoden.

Tabel 6: FURPS

6 Detaljerede Krav

Detaljerede krav er en uddybelse af foregående afsnit **??** "Overordnede Krav". I dette afsnit vil der udarbejdes detaljerede brugsmønsterbeskrivelser udfra de allerede udarbejdede overordnede brugsmønsterbeskrivelser. Der vil I dette afsnit blive fokuseret på det mest essentielle brugsmønster for systemet.
Afsnittet vil derudover indeholde en detaljeret beskrivelse af de supplerende ikke-funktionelle krav. Til dette bliver FURPS+ modellen benyttet.

6.1 Detaljerede Brugsmønstre

I tabel **??** ses en detaljeret beskrivelse af det essentielle brugsmønster "Opret Kreditering". Beskrivelsen er udarbejdet ud fra aktørlisterne og brugsmønster beskrivelserne i inceptionsdokumentet (se bilag **??** afsnit 3). Detaljerede brugsmønster beskrivelserne bruges til at danne et overblik over hver enkel brugsmønsters cyklus. Her nævnes brugsmønstrets primære og sekundære aktører, de præ- og postkonditioner der er knyttet til brugsmønsteret - hvad skal være opfyldt, for at brugsmønsteret sættes i gang, og hvad sker der når brugsmønsteret er gennemført. Herudover beskrives hoved- og alternative hændelsesforløb. Hovedhændelsesforløbet beskriver forløbet, hvis der ingen problemer opstår. Alternativ hændelsesforløb beskriver eventuelle alternative hændelser i specifikke trin/steps i hovedhændelsesforløbet.

Brugsmønstrene nedenfor er blevet valgt på baggrund af MosCow analysen i inceptionsdokumentet (se bilag **??** side 15) baseret på systemets brugsmønstre. Det mest essentielle brugsmønster er valgt ud fra brugsmønstrene i kategorien "must have". Det er vurderet at brugsmønstret "Opret Kreditering"er det mest essentielle bnrugsmønster, da det er denne funktion systemet er bygget op omkring.

6.1.1 Opret Krediteringer

Brugsmønster: Opret kreditering
ID: UC07
Primære aktører: Systemadministrator, kanaladministrator, Producer
Sekundære aktører:
Kort beskrivelse: Produceren opretter en kreditering. Heri angives alle der har bidraget til produceringen af TV-programmet, filmen el. lign.
Prækonditioner (Pre conditions): Aktøren skal være logget ind på systemet
Hovedhændelsesforløb (main flow): 1. Brugsmønstret starter når en administrator eller producer vil oprette en kreditering 2. Aktøren trykker på knappen 'Opret Kreditering' 3. Systemet checker aktørens rolle 4. Aktøren er forbundet til en kanal, og angiver programmets titel 5. Systemet checker om der allerede findes et program med den angivne titel 6. Aktøren krediterer alle der har medvirket i produktionen af programmet 7. Aktøren sender den færdige kreditering videre til godkendelse
Postkonditioner (post conditions): En kreditering er blevet sendt videre til godkendelse

Alternative hændelsesforløb (alternative flow):
tep *: Aktøren kan til enhver tid afbryde oprettelsen af krediteringen Step 4: Hvis aktøren er systemadministrator, er vedkommende ikke forbundet til en kanal, og kan skifte hvilken kanal krediteringen skal oprettes ved. Step 5: Hvis programmets titel allerede eksisterer, gøres aktøren opmærksom på dette. Step 8: Hvis krediteringen afvises, laves de fornødne ændringer, og den nye kreditering sendes videre til godkendelse.

Tabel 7: Brugsmønster: Opret kreditering

6.2 Detaljeret beskrivelse af supplerende krav

I tabel **??** ses de detaljerede beskrivelser af supplerende ikke-funktionelle krav i form af en FURPS+ model. modellen er med til at give et overblik over hvilke supplerende ikke-funktionelle krav systemet skal overholde, for at være tilstrækkeligt i forhold til TV2 og gruppens forventninger. FURPS+ er en udviddet udgave af FURPS modellen. Udover kravene i FURPS modellen i afsnit **??**, er der i tilføjet 4 nye kategorier - Design constraints, Implementation requirements, Interface requirements og Physical requirements. Derudover har hver af de ikke-funktionelle krav fået tildelt et nummer for lettere reference.
Af de tilføjede kategorier er nye krav udformet. Systemet skal benytte en relationel databse. Databasen skal skrives i programmeringssproget SQL og laves i PostgreSQL. Desktop clienten bygges i programmeringssproget Java og REST Api'et bygges i Python.
Systemet vil blive unit testet, for at sikre en høj kvalitet af implementeret kode. Systemet vil blive forbindet til det centraliserede fejlognings system `Sentry`.
Systemet skal kunne trække EPG data fra TVTid.dk via en EPG Poller og sende dataen til REST Api'et.

FURPS+	#	**Krav**
Functionality	S01	Skal kunne kreditere produktionsroller som er angivet af DRs krediteringsregler.
	S02	Skal overholde GDPR.
Usability	S03	Systemet skal kunne understøtte flere sprog.
	S04	Systemet skal have en responsiv brugergrænseflade (UI).
Reliability	S05	Hvis serveren til systemet genstarter, startes del-systemerne igen automatisk. Der vil ikke være behov for at ligge systemet ned regelmæssigt for at kunne foretage backup.
Performance	S06	Databasen skal kunne håndtere 10000 nye brugere - samt 15000 krediteringerer årligt i 25 år, uden at ofte brugte kald til REST Api'et bliver sløvt (reponsetid på mere end 300 ms).
	S07	Systemet skal kunne håndtere 5000 brugere indenfor et minut.
Supportability	S08	Systemet er installerbart vha. Docker via Docker-compose.
	S09	Det vil være muligt at konfigurere system indstillinger via en `.env` (miljø) fil. En opsætningsguide vil være at finde sammen med kildekoden.
Design constraints	S10	Systemet skal benytte en relationel database. Databasen skal laves i PostgreSQL og implementeres i programmeringssproget SQL.

Implementation requirements	S11	Systemet vil indeholde unit tests, og komme med en rapport over hvor stor en procentdel der er dækket af dette.
	S12	Systemet vil blive forbundet til det centraliserede fejllognings system `Sentry`.
	S13	Desktop klienten implementeres i programeringssproget Java. REST API implementeres i Python.
Interface requirements	S14	Systemet skal trække EPG data fra TVTid.dk via en EPG Poller.
Physical requirements	S15	Ingen.

Tabel 8: FURPS+

7 Analyse

I dette afsnit bliver overvejelser, beslutninger og resultater vedrørende den statiske side af analysemodel og den dynamiske side af analysemodel berørt. Der bliver hovedsageligt fokuseret på desktop-klienten, grundet projektcasen scope, og analyseafsnittet for REST api'et og EPG polleren er derfor placeret i bilag **??**, og kan med fordel læses hvis der ønskes viden om hvordan projektgruppen analyserede disse to delsystemer.

7.1 Brugsmønsteranalyse

Der vil i dette afsnit blive lavet en brugsmønsteranalyse på de detaljerede brugsmønstre fra afsnit **??**, der har til formål at finde klasse, struktur og adfærd i et system. Det første der gøres er at finde klasser:

7.1.1 Klassekandidater

Når man skal finde potentielle klasser, kan der tages brug af en navneordsanalyse, hvor navneord i de detaljerede brugsmønster er potentielle klasserkandidater. Listen af potentielle klasser fra navneordsanalysen af brugsmønstrene ses i tabel **??**.

Klassekandidat	**Attributter**	**Kommentar**
Bruger	`uuid`, `e-mail`, `password`, `rolle`	Indebærer alle aktørerne (Systemadmin, kanaladmin, producer og royalty bruger).
Produktion	`uuid`, `titel`, `kanal_id`	Et program indebærer alt hvad der bliver vist på TV. (Film, serier osv)
Kreditering	`person_id`, `job`, `produktion_id`	En kreditering er et job udført af en person til en produktion (hvor job kan være lydmand)
Kanal	`ID`, `navn`	Angiver hvilken TV-kanal et program bliver afspillet på
Person	Personinformation (navn, beskæftigelse, e-mail, tlf.nr. osv)	De der medvirker til produceringen af en produktion (kameramand, lydmand, etc.)

Tabel 9: Klasssekandidater

Her er der tildelt nogle attributter, blandt andet fra kasserede klassekandidater, som klasserne bør indeholde.

7.1.2 Kasserede klassekandidater

De resterende potentielle klasser der blev fundet under navneordsanalysen er enten synonymer, operationer eller egner sig bedre som attributter til de valgte potentielle klasser. Disse kan ses i tabel **??**

Kasserede klassekandidater	Kommentar
Gæst	Påvirker ikke systemet - Kan kun se krediteringerne
Systemet	Systemet
Rolle	Attribut til Bruger-klassen
Visning	En Operation
Film	Synonym til Produktion-klassen
Program	Synonym til Produktion-klassen
Knap	En del af Desktop-klienten
Programtitel	Attribut til Program-klassen
Vedkommende	Synonym for Bruger-klassen
Aktøren	Synonym for Bruger-klassen
Person-vinduet	En del af Desktop-klienten
Informationer (navn, beskæftigelse, email, tlf.nr., osv.)	Attributter til Person-klassen
Oprettelse	En operation
Desktopklienten	Systemet
Startsiden	En del af Desktop-klienten
Login-side	En del af Desktop-klienten
Login-oplysninger	Attribut til Bruger-klassen
Brugernavn	Attribut til Bruger-klassen
Password	Attribut til Bruger-klassen
Flow	Beskriver et hændelsesforløb
Navn	Attribut
Primær aktør	Synonym for Bruger-klassen
Personinformation	Attributter til Person-klassen
Godkendelse	En operation
Meddelelse	En operation
Systemadministrator	En rolle i systemet - bliver håndteret af Bruger-klassen
Kanaladministrator	En rolle i systemet - bliver håndteret af Bruger-klassen
Producer	En rolle i systemet - bliver håndteret af Bruger-klassen
Royalty Bruger	En rolle i systemet - bliver håndteret af Bruger-klassen

Tabel 10: Kasserede klassekandidater

Her er det besluttet, at kandidaterne 'Brugernavn' og 'Password' begge er attributter til brugerklassen, og kandidaten 'Gæst' ikke skal være en klasse for sig, da denne klassekandidat kun kan se krediteringerne. Gæster skal ikke have specielle rettigheder eller login-oplysninger, som brugernavn og password.

7.1.3 Klassekategori koncept

De potentielle klasser der arbejdes videre med kan opdeles i kategorier. Dette er med til at skabe et overblik over klassernes 'placering' i systemet samt skabe associationer mellem koncepter og potentielle klasser. Kategorilisten kan ses i tabel **??**.

Klassekategori koncept	Eksempel
Forretnings overførsel	Ingen
Overførselslinje ting	Ingen
Produkt	Ingen
Hvor bliver overførslen optaget	Ingen
Folkets roller	System- & kanaladministrator, Producer, Royalty Bruger
Sted for overførsel	Ingen
Bemærkelsesværdige begivenheder	Ingen
Fysiske objekter	Ingen
Beskrivelse af ting	Krediteringer
Kataloger	Database med krediteringer
Lager af ting	Databasen
Ting på lageret	Krediteringer
Andre samarbejdssystemer	Ingen
Optagelser af finans, arbejde, kontrakter og juridiske sager	Ingen
Finansielle instrumenter	Ingen
Arbejdsplaner, manualer, dokumenter der er regulært refereret til for at performere arbejde	Ingen

Tabel 11: Klassekategorier

7.1.4 Beskrivelse af klasser

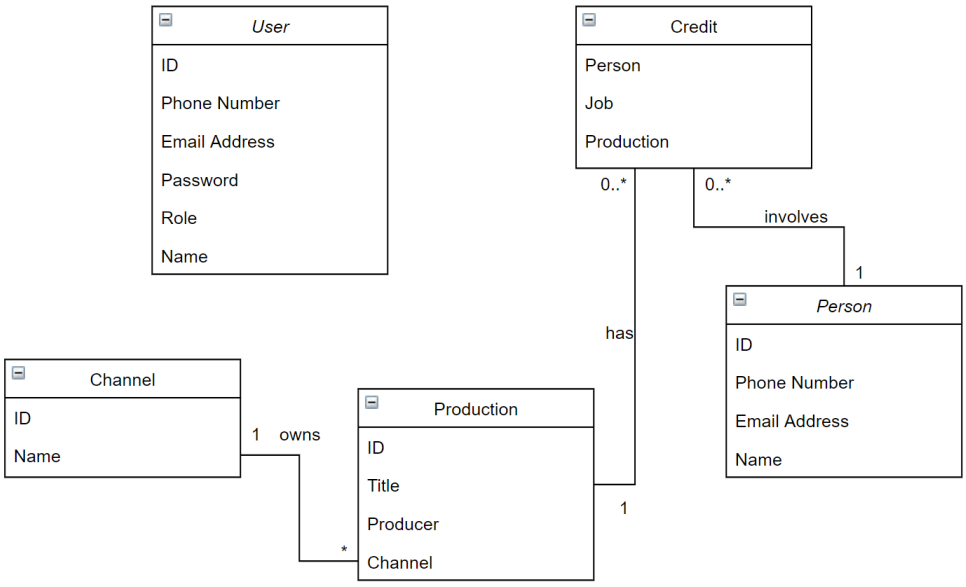
I tabel **??** er der givet et eksempel på, hvordan hver klasse kan komme til at se ud. De resterende kan findes i bilagene.

Navn	Bruger
Definition	En bruger har et UUID, en email-adresse, et password og en rolle.
Eksempel	En bruger kan se ud som følgende: `UUID`: fbf50f43-9f1c-41e1-abce-cca32b836ef0 `Email`: someperson@somemail.com `Password`: Passw0rd `Rolle`: Kanaladministrator
Andet	Nej

Tabel 12: Beskrivelse af Bruger-klassen

7.1.5 Analysemodel

Ud fra navneordsanalysen er de potentielle klasser og deres attributter fundet. Dette vises i analysemodellen, som kan ses på figur **??** , med tilhørende relationer mellem klasserne. Modellen har til formål at danne et overblik over klasserne, deres attributter og deres relationer i systemet, samt at fungere som en model der kan bruges til videreudvikling af systemet.



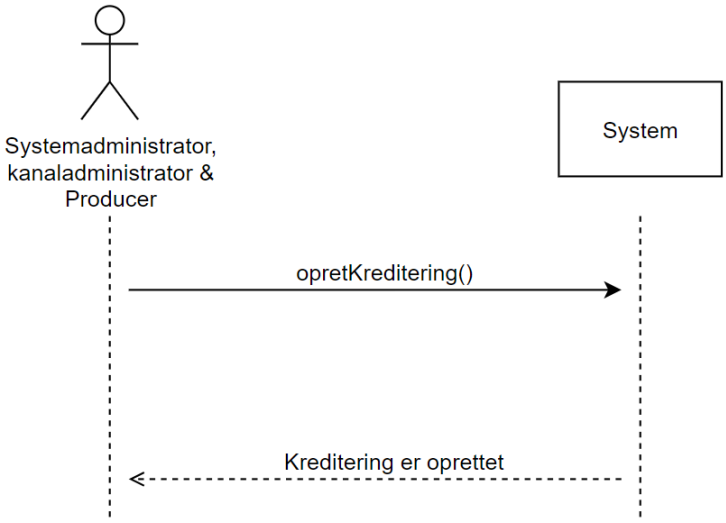
Figur 4: Analysemodel

7.2 Brugsmønsterrealisering

I dette afsnit vil der blive lavet en brugsmønsterrealisering, hvor de funktionelle krav som er udtrykt i brugsmænstrene fra afsnit **??** vil blive realiseret.

7.2.1 Systemsekvensdiagram

Et systemsekvensdiagram er et sekvensdiagram der viser systemhændelserne for ét scenarie af et brugsmønster. Diagrammet viser hvordan aktørerne interagerer med systemet for at opfylde brugsmønstret. Diagrammet viser systemet som en 'black box', hvilket betyder at man ikke kan se hvad der sker inde i systemet, men kun hvad der sker udenfor systemet. På diagrammet ses det, hvordan aktørerne genererer systembegivenheder og hvad systemets output er. Ydermere viser diagrammet den 'tidslinje' begivenhederne sker i.



Figur 5: Systemsekvensdiagram for "Opret Kreditering"

På figur **??** ser man, at aktøren (Systemadministrator, kanaladministrator eller Producer) kalder metoden opretKreditering ind til systemet, som returnerer at en kreditering er blevet oprettet. Man kan altså ikke se hvordan krediteringen bliver oprettet, og det holdes dermed til hvad der sker uden for systemet. Ud over det ovenstående (system)sekvensdiagram er det valgt at medtage systemsekvensdiagrammer for *godkendKreditering*, *opretProducer*, *sePersonInfo* og *læsKreditering*, som kan findes i bilag **??**.

Efter systemsekvensdiagrammerne er lavet, finder man de operationer systemsekvensdiagrammet kræver. Man kigger altså her ikke længere blot udenfor systemet, men ser på præcis hvordan 'flowet' for brugsmønstret udvikles. Til dette skal man kigge på kontrakter for systemfunktioner.

7.2.2 Kontrakter for systemfunktioner

En systemoperationskontrakt beskriver en operations ansvar, altså hvad en operation har forpligtet sig til. Kontrakten lægger vægt på hvad en operation ændrer på, og ikke på hvordan det ændre sig. En kontrakt kan derfor anses som værende en formel beskrivelse af en operation.
Systemoperationskontrakten indeholder navnet på operationen, krydsreferencer til de relevante brugsmønstre, beskrivelse af ansvaret, det output operationen genererer, samt pre- og postkonditioner for operationen.

Pre- og postkondinitionerne er stilbilleder af systemet på det givne tidspunkt operationen bliver kaldt. De beskriver altså systemets tilstand før og efter at operationen har kørt. Postkonditioner skal altid noteres i datid, som for eksempel "Købet blev foretaget", da det er en ting der er sket, og ikke sker.

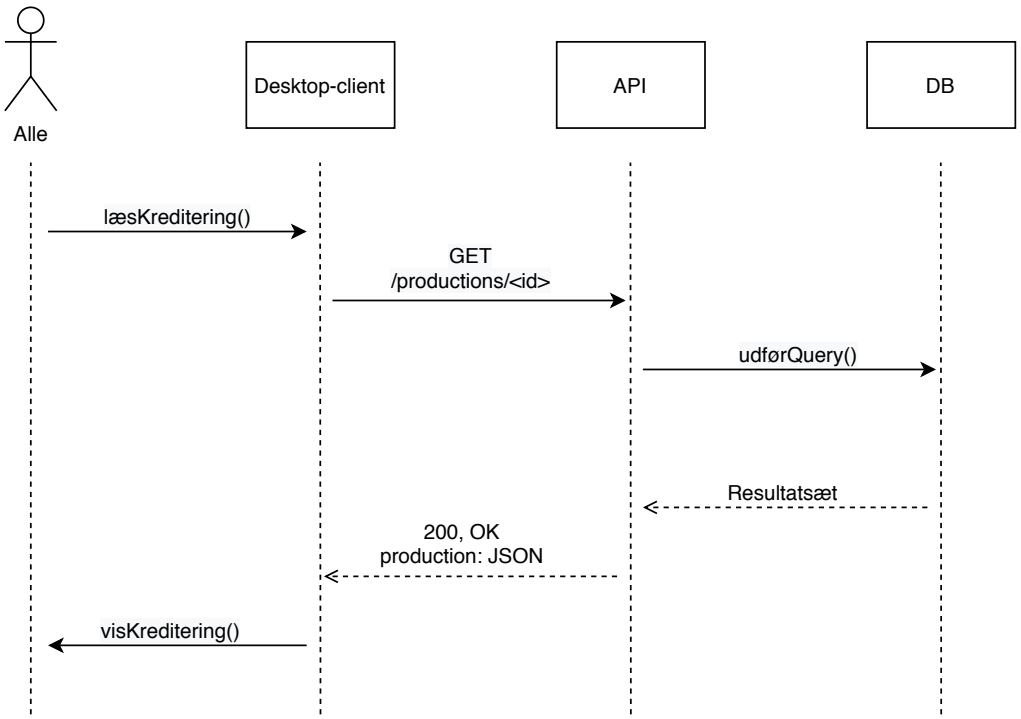
Det er valgt at lave operationskontrakter for de samme operationer som ved systemsekvensdiagrammerne, da disse operationer viser essensen af den logik der skal implementeres. I tabel **??** ses et eksempel på en sådan kontrakt for operationen: *opretKreditering*. De resterende kontrakter kan findes i bilag **??**.

Opret kreditering	
System operation	**opretKreditering**
Krydshenvisning	Use case: Opret Kreditering
Ansvar	At oprette krediteringer og sende den videre til godkendelse hvis prækonditionen er opfyldt, og følgende betingelser er sande: 1. Minimumskravene er opfyldt 2. Alle oplysninger er indtastet korrekt Hvis ikke ovenstående er sande, oprettes krediteringen ikke, og brugeren bliver informeret herom
Output	1. Produceren får besked om krediteringen er oprettet 2. System- og/eller kanaladministrator får besked om en nyoprettet kreditering
Prækonditioner	Logget ind som kanal- eller systemadministrator
Postkonditioner	En ny kreditering er oprettet i systemet

Tabel 13: Systemfunktionskontrakt 'Opret kreditering'

7.2.3 Operationssekvensdiagram

Operationssekvensdiagrammet viser systemet som en "white box", hvor man kan se hvad der sker inde i systemet. Sekvensdiagrammet bruges til at identificere systemfunktioner, da de begivenheder der vises i diagrammet er de funktioner systemet skal indeholde. Et eksempel på et operationssekvensdiagram for "Læs Kreditering"kan ses på figur **??**.



Figur 6: Operationssekvensdiagram: Læs kreditering

På figur **??** ses det, hvad der finder sted i systemet når brugeren vil læse en kreditering. Brugeren starter ud med at kalde en metode i desktop-klienten, der kalder en `GET` metode i REST api'et, hvor id'et eller navnet på en produktion er med som parameter. Api'et udfører derefter SQL-forespørgelse i databasen, for at hente den specifikke kreditering. Databasen sender et resultatsæt tilbage, og api'et sender en HTTP kode sammen med et JSON object til desktop-klienten. JSON objektet er den produktion brugeren gerne vil læse krediteringerne til. Desktop-klienten håndterer derefter JSON objektet, så det pænt vises til brugeren. De resterende operationssekvensdiagrammer kan findes i bilag **??**.

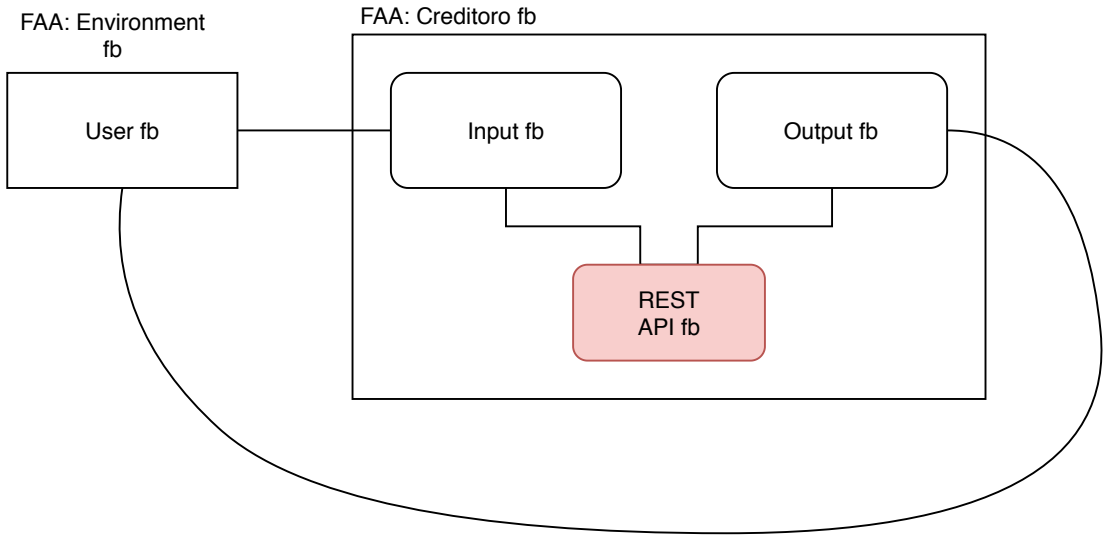
7.3 Cyber-Physical systems

Inden for Cyber-physical systemer differentierer man mellem funktionelle og ikke-funktionelle krav. Funktionelle krav er krav, der viser *hvad* systemet skal kunne gøre. De beskriver altså en opførsel (metode). De funktionelle krav for systemet kan deles op i de forskellige subsystemer. Desktop-klienten og Rest API'et slås sammen, da disse deler funktionalitet. Desktop-klienten kalder til API'et hvori logikken håndteres, og logik i klienten mindskes derved. EPG Polleren har egne funktionelle krav. Disse krav kan findes i bilag **??**. Kravene er bl.a fundet ud fra den udleverede projektbeskrivelse, og et møde med TV2 hvor spørgsmål er blevet besvaret.

Ikke-funktionelle krav beskriver *hvordan* et system virker. Det kunne fx være hvor lang tid et API kald må tage fra man trykker på knappen til klienten opdateres. Også her deles kravene op i subsystemer. De ikke-funktionelle krav for desktop-klienten kan fx være at brugeren skal have besked om fejl via en popup-boks, eller at ofte brugte knapper skal virke inden for 300 millisekunder. Disse krav kan findes i bilag **??**.

Ud fra de fundne funktionelle krav kan der udarbejdes EAST-ADL modeller. Disse modeller skal vise hvordan de pågældende subsystemer hænger sammen. Hver model indeholder flere forskellige funktions bokse, der repræsenterer de vigtigste komponenter i systemet, associeret med det enkelte krav. Disse funktionsbokse har relationer der viser hvordan de forskellige subsystemer interagerer med hinanden. Hver funktionsboks repræsenterer henholdsvis en miljø (environment) funktionsboks og en analyse funktionsboks. I EAST-ADL modeller vil analyse funktionsboksen indeholde flere funktionsbokse, og vise interaktionerne herimellem.

I figur **??** ses et eksempel på den primære EAST-ADL model. Resten af de udarbejdede modeller kan ses i bilag **??**. Modellen har bruger (user) funktionsboksen som miljø, og Creditoro funktionsboksen som system på analyse niveauet. I analyse boksen ses det, at gives der en forespørgsel fra brugeren til systemet (input fb), kommunikeres der til Rest API'et, som sender den kaldte data som er forespurgt videre (output fb), så brugeren får det tilgængeligt.



Figur 7: EAST-ADL model - Main

8 Design

I dette afsnit vil overvejelser, beslutninger og resultater vedrørende softwarearkitektur, subsystemdesign og design af persistens blive beskrevet.

8.1 Softwarearkitektur

8.1.1 Overvejelser

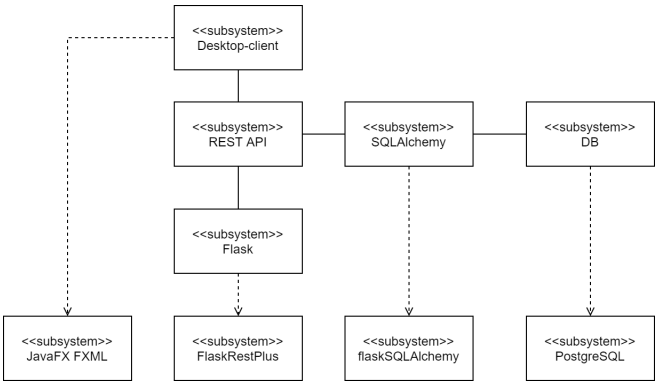
Da projektcasen blev fremlagt fra gruppen var der bred enighed om, at lave et system der lægger tæt op ad det oprindelig forslag. I projektcasen fremgik et forslag til en systemtegning, som kan ses på figur **??**. Et sådan system ville indeholde et REST api, som i dette projekt er uden for scope, og dette brugte gruppen meget tid på at diskutere. Hvis projektet skulle omhandle et REST api, måtte det ikke gå ud over kvaliteten på resten af projektet, og hvis det på et tidspunkt viste sig, at det ikke ville være muligt at implementere api'et, skulle det skrottes og der skulle findes en anden løsning. For at vise krediteringerne til brugerene, blev det overvejet om der skulle laves en wepapplikation og en desktop-klient. Både system- og kanaladministratore, producere, royalty brugere og gæster ville skulle have adgang til denne del af system. For at hente data om kanaler og produktioner, blev det overvejet om der skulle laves en EPG poller, der ville hente data fra TVTid.dk, og derved få alle kanaler og deres tilhørende produktioner.

8.1.2 Beslutninger

På baggrund af gruppens overvejelser blev det valgt af sætte et nyt domæne diagram op over systemet der viser hvordan opsætningen er tiltænkt. I systemet bliver det en REST API der binder hele systemet sammen. REST API vil håndtere input fra Desktop-client gennem subsystemet flask og sende samt hente information fra databasen via subsystemet SQLAlchemy.

8.1.3 resultater

Ud fra beslutningerne kunne dannes et Domændediagram af software arkitekturen som kan ses på figur **??**.



Figur 8: Software Arkitektur Diagram

På diagrammet kan de forskelliges subsystemer relationer ses samt de afhængigheder de har.

8.2 Subsystemdesign

Overvejelser, beslutninger og resultater for designet af REST api'et og EPG-Polleren kan findes i bilag **??**.

8.2.1 Desktop Klient

Overvejelser
Den overvejelse der fyldte mest var beslutningen omkring hvilket designmønstre, vi skulle designe vores applikation med hensyn til.
I undervisningen er vi blever undervist i et simplificeret designmønster, der opdeler applikationen i tre lag; præsentation, domæne - og persistens lag. Hvor domæne laget indeholder alt vores forretningslogik. En anden mulighed der minder lidt om denne er MVC (model-view-controller), der også indeler systemet i tre lag, eller MVVM (model-view-viewmodel), hvor viewet ikke kender noget til modellen og vice versa, men i stedet forbindes via viewmodel laget, og hvor det meste UI kan opdateres fra viewmodel laget gennem to-vejs data-bindings.

Beslutninger
Vi endte med at benytte MVVM mønstret, da det trods dets overhead at sætte op, vil være et godt valg til vores applikation og gruppe. Vi kunne godt se det smarte i at kunne udvikle på view delen (view controller og FXML) separat fra vores viewmodel og model lag. Samt er det også fedt at man yderligere kan opdele arbejdet med viewmodel og modellen, efter man har designet en "kontrakt"ved at blive enige om de forskellige metoder modellen skal implementere i et interface.

Resultater
Ud over de tre lag, lavede vi også en netværksmappe der står for at kommunikere med API'et. Ved at separere denne logik har vi også haft mulighed for at udvikle på desktop-klienten sideløbende med API'et. Da vi blot kunne lave en "dummy"klient, der returnerer statiske objekter, i stedet for det faktiske data som den får fra API'et.

Ud fra de nye opdagelser i forhold til design kunne analysemodellen opdateres til en design klassemodel som kan findes i bilag **??**.

8.3 Persistens design

Da persistens skulle designes, var der en stor overvejelse om hver klient skulle have deres egen database, eller om der skulle stræbes efter at få et mere virkelighedstro system (et system der vil afspejle det TV2 lagde op til), der indeholder en centraliseret database. Det blev valgt at stræbe efter det sidstnævnte, og i forbindelse med dette blev der foretaget et valgt at kommunikationen mellem klient og database skulle foregå gennem et REST API så klienterne ikke har direkte adgang til databasen. På denne måde sikres det også at brugere af systemet kun kan foretage de handlinger som deres respektive rolle giver adgang til, da valideringen ligger på server-siden (så brugeren ikke har muglighed får at sniffe brugernavn og kode til databasen og få adgang af andre veje).

9 Databasedesign

Denne sektion har til formål at fremvise overvejelser, beslutninger og resultater vedrørende tabeldesign og SQL-forespørgelser.

9.1 Overvejelser

9.1.1 Tabeldesign

Eftersom systemet skal kunne håndtere visninger af kanaler, produktioner med tilhørende krediteringer og de personer der bliver krediteret, vil det give god mening at lave et table til hver af disse. Altså kanaler, produktioner, krediteringer og personer. Formålet med at opdele dataet i disse tabeller er at opnå en høj normalisering, for at gøre dataet uafhængig af hinanden. Det vil være en fordel hvis alle tabeller indeholder et field til et unikt id, som kunne være et UUID, så alle instanser har et unikt id man kan søge på og referere til.

I kanaltabellen kunne det være en god idé at opbevare et navn på den pågældende kanal og kanalens logo. I produktionstabellen vil information som en titel på produktionen være relevant, så man kan søge efter bestemte produktioner. Det vil også være en god idé at opbevare informationer som hvilken producer der har produceret titlen og hvilken kanal produktionen er lavet til. Til sidst kan man også opbevare information om hvilken dato produktionen er udgivet på. I krediteringstabellen vil det være relevant at gemme information som produktions id, så man kan koble de enkelte krediteringer sammen med en produktion. Ud over produktions id'et, kan information som person id og jobtitel være relevant, så man kan koble en bestemt person sammen med en jobtitel, som for eksempel "Lydmand; identifikationsnummer på en person". I persontabellen kan der opbevares personoplysninger, som navn, telefonnummer og email, så man har et navn på en person der skal krediteres og kontaktoplysninger til vedkommende.

Der skal også være personer der har adgang til systemet, som for eksempel en systemadministrator og en producer. Derfor vil det give god mening at have en tabel hvor brugerene er placeret. De informationer der skal opbevares om en bruger kan for eksempel være navn, email, kodeord og den rolle vedkommende har i systemet. Hver bruger skal have et unikt id, så man let kan kende forskel på brugere, og så der kan kendes forskel på brugere med samme navn.

9.1.2 SQL-forespørgelser

For at opnå den ønskede funktionalitet skal funktionerne hent, opret, opdater og slet være tilgængelige på samtlige tabeller der oprettes. Som udgangspunkt skal det være muligt at søge efter `navn/titel` på den pågældende tabel, hvilket betyder at hent funktionen skal implementeres for alle brugere af systemet. Internt i systemet vil der med fordel kunne søges efter `identifier` på den pågældende tabel. Opret, updater og slet funktionerne skal kun være tilgængelige for visse brugere, alt efter deres rolle i systemet. En producer skal for eksempel kun have rettigheder til at opdatere egne krediteringer, hvor en kanaladministrator har rettigheder til at opdatere alle krediteringer for den kanal vedkommende administrerer.

Når man åbner oversigten over alle kanaler i desktop-klienten, vil det være en god idé at den automatisk lave en query til databasen, så logoerne bliver vist. Man skal derefter kunne søge på kanalernes titler, så hvis man for eksempel skriver "t"i søgefeltet, skal man få vist alle kanaler hvis title indeholder et 't'. Queries bør ikke være case-sensetive. Ved at lave queries til produktioner og producere, vil brugeren af programmet være i stand til at finde produktioner

efter titel eller producerens navn. Det kunne også være en mulighed, at man skal kunne finde en produktion, ved at søge efter skuespillere, lydmænd og lignende. Derfor kunne det være en god idé at lave queries efter personernes navne og/eller jobtype.

9.2 Beslutninger

9.2.1 Tabeldesign

Det blev besluttet at der skal laves 5 tabeller, `channels`, `users`, `people`, `productions` og `credits`, som beskrevet nedenfor:

`channels` kommer til at bestå af 3 fields, `identifier`, `name` og `icon_url`. `identifier` er en primary key af typen `UUID` (version 4 UUID), og har til formål at sirke, at alle kanaler har et unikt id. `name` er af typen `VARCHAR` og har til formål at gemme navnet på en kanal, så man kan vise navnet frem for et `UUID` i desktop-klienten. `icon_url` er af typen `VARCHAR` og har til formål at gemme url-adressen til logoet for en given kanel. Dette skal bruges til at gemme kanalikonet til desktop-klienten. Ydermere skal der laves et `index` på `name`, så man hurtigt kan søge på en bestemt kanal.

`users` kommer til at bestå af 7 fields, `identifier`, `name`, `email`, `phone`, `password`, `role` og `created_at`. `identifier` er af samme type som i channels, har samme formål og er også en primary key i `users`. `name` er af typen `VARCHAR` og har til formål at gemme navnet på en bruger. `email` er af typen `EMAIL` og `phone` er af typen `VARCHAR`. Disse fields har til formål at gemme kontaktoplysninger om de enkelte brugere. `password` er af typen `VARHCAR` og bruges til at gemme det kodeord (SHA256 krypteret) en bruger har valgt. `role` er af typen `ROLE` og har til formål at give en bruge en bestemt rolle i systemet, som for eksempel "producer". `created_at` er af typen `TIMESTAMP` og har til formål at gemme hvornår en bruger er blevet oprettet i systemet. Ydermere skal der laves et `index` på `email`, så man hurtigt kan søge på en bestemt bruger med en email, og `name`, så man hurtigt kan søge på en bestemt bruger med et navn.

`people` kommer til at bestå af 4 fields, `identifier`, `name`, `email` og `phone`. `identifier` er af samme type som i channels, har samme formål og er også en primary key i `people`. `name` er af typen `VARCHAR` og har til formål at gemme navnet på en person, der senere hen skal krediteres i en produktion. textttemail og `phone` er af samme typer som i `users` og har samme formål. Ydermere skal der laves et `index` på `name`, så man hurtigt kan søge på en bestemt person med navnet på personen.

`productions` kommer til at bestå af 5 fields, `identifier`, `titel`, `created_at`, `producer_id` og `channel_id`. `identifier` er samme type som i `channels`, har samme formål og er også en primary key i `productions`. `titel` er af typen `VARCHAR` og er titlen på en produktion, som for eksempel "Bonderøven", og har til formål at koble en titel til `identifier`. `created_at` er af typen `TIMESTAMP` og har til formål at gemme hvornår en produktion er tilføjet systemet. `producer_id` er af typen `UUID` og refererer til `users` (`identifier`). Dette field har til formål at koble en producer på en produktion, så det bliver tydeligt hvem der har produceret titlen. `channel_id` er af typen `UUID` og refererer til `channels` (`identifier`). Dette field har til formål at koble en produktion til den kanal produktionen tilhøre. Ydermere skal der laves et `index` på `titel`, så man hurtigt kan søge på en bestemt produktion med titlen, og `identifier`, der skal sikre at gøre søgningen på produktioner på en bestemt kanal hurtigere.

`credits` kommer til at bestå af 4 fields, `identifier`, `job`, `production_id` og `person_id`. `identifier` er af samme type som i channels, har samme formål og er også en primary key i `credits`. `job` er af typen `VARHCAR` og kommer til at være den jobbeskrivelse der passer til den person der bliver krediteret, som for eksempel "Lydmand". `production_id` er af typen `UUID` og refererer til `production` (`identifier`), og har til formål at koble en kreditering på en produktion. `person_id` er af typen `UUID` og refererer til `people` (`identifer`), og har til formål at koble en person på den jobtitel der bliver lavet i den produktion den tilføjes til. Ydermere skal der laves et `index` på `job`, så man hurtigt kan søge på en bestemt jobtitel i en produktion, og `identifier`, der skal sikre at visningen af krediteringer på en produktion sker hurtigere.

9.2.2 SQL-forespørgelser

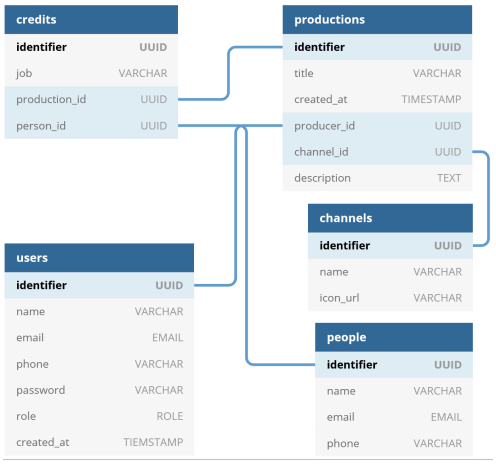
Det er besluttet at der skal laves en `GET` SQL-forespørgelse på samtlige tabeller der oprettes, som er tilgængelige for alle brugere af systemet. Dette gør brugeren i stand til for eksempel at søge efter en bestemt produktion. Forespørgelsen skal kunne håndtere forskellige query properties, alt efter hvad der søges efter.

For brugere med et login skal det være muligt at lave `POST`, `PATCH`, `GET`, `DELETE` og `PUT` forespørgelser på samtlige tabeller i databasen. `POST` gør det muligt at oprette nye instanser i den pågældende tabel, `PATCH` gør det muligt at opdatere fields i den pågældende tabel, `GET` henter en instans ved brug af instansens `identifier`, `DELETE` sletter instansen ved brug af instansens `identifier` og `PUT` tilføjer en instans på den valgte URI (uniform resurseidentifikator).

9.3 Resultater

9.3.1 Tabeldesign

Resultatet ud fra de beslutninger der blev truffet kan ses i figur **??**. Ønkses de enkelte tabeller med indsat data at ses, kan dette gøres i bilag **??**, hvor sql'en for at oprette tabellerne og indexes også kan ses.



Figur 9: Tabeldesign

9.3.2 SQL forespørgelser

Det er muligt at indsætte data, opdatere data, slette data og hente data fra tabellerne, ud fra de beslutninger der er blevet taget. Eksempler på hvordan disse SQL forespørgelser kan se ud,

kan ses i bilag **??**.

10 Implementering

Denne sektion skal indeholde: // Kristian skriver :-)

10.0.1 Database implementering

Overgang fra design til implementering var forholdsvis smerteløs givet vores design dokument med oversigt over hvilke endpoints og hvad hvert endpoint skulle returne.
Selve implementeringen blev foretaget via migration scripts, her er den primære fordel at gøre det let at tiføje evt. nye features uden manuelt at skulle eksekvere SQL commands på serveren hvor systemet kører. I takt med at vi implementerede et endpoint (fx. `users`. Lavede vi en migration for dette endpoint. Migrationer i projektet er implementeret vha. en Python fil der indeholder et revisions nummer, samt hvilken revision den peger på (hvilken revision der skal blive til den nuværende revision ved 'downgrade'. Python scriptet består af to funktioner: `def downgrade` og `def upgrade`. Der hver især peget på henholdsvis en `downgrade.sql` - og en `upgrade.sql` fil. Upgrade scriptet køres vha. `flask db upgrade`, databasen ved hvilken revision der er den nuværende ved at kigge i tabellen `alembic_version` der indeholder den nuværende revision i kolonnen `version_num`, tabellen inderholder (og vil altid kun indeholde en række). Når upgrade scriptet køres kigger det på den nuværende revision i databasen, derefter kigger den Python filerne igennem i mappen `migrations/versions` for at finde en fil der har `down_revision` sat til den nuværende situation, hvis den finder sådan et, så opgradereres databasen ved at køre `upgrade` funktionen for den fil der blev fundet.

- Omfatter afsnittet overvejelser, beslutninger og resultater vedr. konvertering fra design til kode illustreret gennem udvalgte centrale eksempler, samt andre vigtige implementeringsbeslutninger.
- Implementering af database.

Hvilke overvejeleser har vi haft under implementeringsfasen? - Har vi tænkt at det forskellige subsystemer skulle implementeres på en bestemt måde? - Er der nogle 'regler' vi har tænkt på at vi skal følge under implementeringen, det kan være specifikke koderegler eller f.eks. måder vi sætter constructers (alt+ins :-)) og metoder op på?

Hvilke beslutninger har vi taget, på baggrund af vores overvejelser. Var der noget der ikke virkede som det burde og skulle laves om?

For at sikre en relativ høj kodekvalitet, var noget af det første vi gjorde i implementeringsfasen at sætte CI (continous integration) op. I praksis vil det sige at hver gang der blev commited kode, vil vores pipeline kører automatisk. CI pipelinen består i alle tre repositories af at eskekvere repositoriets Unit test, dernæst statisk analyse vha. Sonarcloud, som fortæller os om "code smells", evt. sikkerhedsproblemer, samt kode dupplikationer og kode kompleksitet. Ydermere er der et ekstra step i API repositoriet, der uploader en ny version af dets Docker image til DockerHub, hvis test steppet kørte med succes.

Ud fra vores beslutninger, hvordan er vores kode så kommet til at se ud? Hvordan er systemet blevet (brugergrænseflade, api'et, epg-polleren og vores database? - Vis specifikke eksempler = brug code snippets

```
DROP INDEX users_email_idx;
DROP INDEX users_name_idx;
DROP TABLE users CASCADE;
DROP DOMAIN IF EXISTS email;
DROP EXTENSION IF EXISTS citext RESTRICT;
DROP TYPE role;
```

Listing 1: Downgrade.sql

```
CREATE EXTENSION citext;
CREATE DOMAIN email AS citext
    CHECK ( value ~
            '^[a-zA-Z0-9.!#$%&''+/=?^_`{|}~-]+@[a-zA-Z0-9](?:[a-zA-Z0
                -9-]{0,61}[a-zA-Z0-9])?(?:.[a-zA-Z0-9](?:[a-zA-Z0
                -9-]{0,61}[a-zA-Z0-9])?)$' );
CREATE TYPE role AS ENUM ('royalty_user', 'channel_admin', 'system_admin'
    );
CREATE TABLE users
(
    identifier UUID          NOT NULL,
    name       VARCHAR(512),
    email      email UNIQUE NOT NULL,
    phone      varchar(254),
    password   VARCHAR(256),
    role       role          NOT NULL,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (identifier)
);
CREATE INDEX users_email_idx ON users USING btree (email);
CREATE INDEX users_name_idx ON users USING btree (name);
```

Listing 2: users.sql

For at give et konkret eksempel på en af vores migrationer, bruger vi ųsers migrationen. Som det kan ses i figur ¡insert-txt¿ Update scriptet laver en tabel med identifier som primær nøgle, typen er 'UUID' (universally unique identifer), som vi har valgt at bruge generelt frem for auto incrementing integer. Grunden til at vi har favoriseret UUID's er at man ellers kan udelade hvor mange brugerer der er i systemet, samt kan det foretage problemer hvis man forsøger at skrive to transaktioner til databasen på samme tid. Kolonnen password er ikke gemt i plain text, men i stedet SHA256 hashed og salted. Der er ikke nogen speciel grund til at vi har valgt denne hashing algoritme, og vi burde nok have valgt en algoritme der er ment til passwords i stedet som fx PBKDF2.
I stedet for at skulle skrive plain SQL i vores high-level Python API, bruger vi istedet en ORM (object relational mapping), der via SQLAlchemy biblioteket, mapper vores PostgreSQL database's tabeller til Python klasser.

11 Test

11.1 1. Iteration af desktop-klient testning

Efter første iteration blev der implementeret en klient der håndterer efterspørgelser hos api'et. For at teste klientens metode blev der oprettet en klasse kaldet DummyClient, derudover dækker testen alle implementerede brugsmønstre idet det er klienten der i sidste ende håndtere dem. Ved at teste den klasse kan funktionaliteten af den reelle klient også afgøres. Denne test er den mest essentielle test af desktop-klienten i iteration 2 og er derfor den eneste medtaget. Resten kan findes i kilde-koden.

Testen starter med at oprette et dummyClient objekt i constructeren.

```
14 DummyClient dummyClient;
15
16 public DummyClientTest() {
17     dummyClient = new DummyClient();
18 }
```

Listing 3: DummyClientTest.java

De første to test der bliver udført, er login og register. Login tester om et falsk login returnerer null og register tjekker om at register ikke kaster en exception når metoden bliver kaldt.

```
20 @Test
21 void login() {
22     assertNull(dummyClient.login("someUser", "somePassword"));
23 }
24
25 @Test
26 void register() {
27     assertDoesNotThrow(() -> dummyClient.register(new User("10-10-10", "
       50505050", "string@string.dk",
28             "MyName", "My Admin Role")));
29 }
```

Listing 4: DummyClientTest.java

De næste fire test tjekker om metoder givet en tom parameter ikke returnerer null. Så hvis testen skal bestås skal metoderne returnere et objekt og ikke null. SearchChannels returnerer en liste af kanaler, searchProductions en liste af produktioner, getPeople en liste af personer, getCredits en liste af krediteringer.

```
31 @Test
32 void searchChannels() {
33     assertNotNull(dummyClient.searchChannels(""));
34 }
35
36 @Test
37 void searchProductions() {
38     assertNotNull(dummyClient.searchProductions(""));
39 }
40
41 @Test
42 void getPeople() {
43     assertNotNull(dummyClient.getPeople(""));
44 }
45
46 @Test
47 void getCredits() {
48     assertNotNull(dummyClient.getCredits(""));
49 }
```

Listing 5: DummyClientTest.java

De næste to test tjekker funktionaliteten af oprettelse og modificering af krediteringer. PatchCredits tjekker om metoden returnere et objekt efter at have sendt forespørgslen om modificering af det gældene kredit. PostCredit tjekker ligesom den anden test om det returnerede objekt er null denne forespørgsel er dog en oprettelse af en kreditering.

```
51 @Test
52 void patchCredits() {
53     assertNotNull(dummyClient.patchCredits("/helloID/", Map.of("jobtype", "
           MyMan", "User", new Object())));
54 }
55
56 @Test
57 void postCredits() {
58     assertNotNull(dummyClient.postCredits(new Credit("ID-1-1-1-11",
59             new Production("ID-3-3-3", "title-10", "Description LONG",
60                     new User("ID-5-5-5", "30303030", "Mail@Mail2.gg", "
                           ProducerName", "admin"),
61                     new Channel("id-4-4-4", "DR1", "https://api.creditoro.
                           nymann.dev")),
62             new Person("ID-2-2-2", "505055050", "Mail@mail.gg", "
                   MYNAMEISNOTHING"),
63             "jobtype")));
64 }
```

Listing 6: DummyClientTest.java

Den sidste test er deleteCredit som tjekker om den boolean som returneres er false. Klienten bliver bedt om at slette en ikkeeksisterende kreditering hvilket skal returnere false.

```
66 @Test
67 void deleteCredit() {
68     assertFalse(dummyClient.deleteCredit("test"));
69 }
```

Listing 7: DummyClientTest.java

11.2 2. Iteration af desktop-klient testning

11.2.1 Test af endpoint klassen

En essentiel del af desktop-klient er at brugmønsteret login (se tabel **??** i bilag) kan håndteres. For at teste implementeringen af dette brugsmønster bliver nedenstående test udført.

```
8  UsersEndpoint usersEndpoint;
9
10 public UsersEndpointTest() {
11     usersEndpoint = new UsersEndpoint(new HttpManager());
12 }
```

Listing 8: UsersEndpointTest.java

UserEndpoint testen starter ved at den i dens constructer danner et nyt userEndpoint objekt. postLogin testen tester om en der kan logges ind i systemet. Det gør den ved at hente den token man får tilbage når man logger ind. Det token er en string der er null hvis brugeren ikke kunne logge ind eller en lang string som giver brugen adgang til applikationen i sat stykke tid.

```
34 @Test
35 void postLogin() {
36     var token = usersEndpoint.postLogin("string@string.dk", "string");
37     assertNotEquals(null, token);
38     token = usersEndpoint.postLogin("wronglogin@string.dk", "string");
39     assertNull(token);
40 }
```

Listing 9: UsersEndpointTest.java

Efter at have dannet token objektet tjekker den om at den ikke er null. Det gør den ved at bruge assertNotEquals metoden der tjekker om den værdi man giver den ikke er den forventede værdi. Så hvis token objektet ikker null består testen.

Herefter bliver der testet for det modsatte. Der bliver igen oprettet et token objekt med et forkert login så token objektet skulle gerne være null. Efter objektet er oprettet bliver det testet af assertNull der tester om værdien er null. Så hvis værdien er null består testen.

11.2.2 Test af loginViewModel

Testen af loginViewModel sker ved at der i constructeren bliver oprettet en loginViewModel

```
16 private final LoginViewModel loginViewModel;
17
18 /**
19  * Instantiates a new Login view model test.
20  */
21 public LoginViewModelTest() {
22     var clientFactory = new ClientFactory();
23     var modelFactory = new ModelFactory(clientFactory);
24     var viewModelFactory = new ViewModelFactory(modelFactory);
25     loginViewModel = viewModelFactory.getLoginViewModel();
26 }
```

Listing 10: LoginViewModel.java

Herefter bliver setUp metoden kørt før hver test.

```
28 @BeforeEach
29 void setUp() {
30     loginViewModel.emailProperty().setValue("string@string.dk");
31     loginViewModel.passwordProperty().setValue("string");
32 }
```

Listing 11: LoginViewModel.java

Her bliver loginViewModels email samt password sat. Der er i alt fire test metoder, emailProperty, passwordProperty, loginResultProperty og clearFields.

MailProperty test metoden checker om den email fra setUp metoden er den samme som den forventede værdi.

```
34 @Test
35 void emailProperty() {
36     assertEquals("string@string.dk", loginViewModel.emailProperty().get());
37 }
```

Listing 12: LoginViewModel.java

Det sker ved at kalde assertEquals metoden der tjekker om den forventede værdi er lig den givende værdi. I dette tilfælde tjekker den om emailen er **string@string.dk** og hvis den ikke er fejler testen.

PasswordProperty test metoden gør det samme som emailProperty test metoden, men i stedet for en email er det et password der bliver testet for.

```
39 @Test
40 void passwordProperty() {
41     assertEquals("string", loginViewModel.passwordProperty().get());
42 }
```

Listing 13: LoginViewModel.java

LoginResultProperty test metoden bruger assertNull metoden, der fejler testen hvis det input den får er det samme som null. Her bliver loginViewModellens loginResponseProperty kaldt, for at se om den kan logge ind med de oplysninger som blev defineret ved setUp metoden. Så hvis den ikke kunne logge brugeren ind med de oplysninger den fik, returnerer den null og testen fejler.

```
44 @Test
45 void loginResultProperty() {
46     assertNull(loginViewModel.loginResponseProperty().get());
47 }
```

Listing 14: LoginViewModel.java

ClearFields test metoden kører først loginViewModel's clearFields metode, der fjerner tekst i email og passwordProperty. Herefter tjekker den med assertEqueals om de to er tomme, ved at sammenligne dem med et empty string. Så hvis email og password felterne er tomme består testen.

```
49 @Test
50 void clearFields() {
51     loginViewModel.clearFields();
52     assertEquals("", loginViewModel.emailProperty().get());
53     assertEquals("", loginViewModel.passwordProperty().get());
54 }
```

Listing 15: LoginViewModel.java

12 Diskussion

Ved starten af projektet, har gruppen haft en forventning om at få konstrueret en prototype til et fuldt funktionelt krediteringssystem. Denne prototype indebærer desktop-klient, Rest API og EPG Poller, og skal kunne indeholde offentligt tilgængelige krediteringer. Man skal som producer af en tv-produktion kunne oprette krediteringer. Derudover skal alle kunne søge efter krediteringer.
Generelt set er disse indledende forventninger opnået. Gruppen har konstrueret en desktop-klient, der gør det muligt at oprette, søge efter og læse krediteringer for TV-produktioner listet på TVTID.dk. Denne data er hentet via en EPG Poller, og gemt i en database med Rest API som formidlingsstation. Projekt beskrivelsen lægger op til at systemet, optimalt set, skal bygges som en webapplication, men grundet at tidligere undervisning har været fokuseret på JavaFX og Scene Builder, har gruppen valgt at konstruere en desktop-klient.
Inceptionsdokumentets bilag **??** viser MoSCoW-modellen, som er en opdeling og prioritering af de forskellige brugsmønstre systemet, optimalt set, skulle indeholde. Da prototypen ikke er fuldendt, er ikke alle disse brugsmønstre opnået. Gruppen har i desktop-klienten ikke fået implementeret funktionalitet der bl.a. gør det muligt for brugeren at logge af, oprette brugere, se personinformationer og godkende/afvise nye krediteringer. Derimod er disse blevet implementeret i API'et, med undtagelse af 'log af' funktionalitet. De forskellige brugerroller (System- og kanaladministrator, royalty bruger og producer) er ligeledes kun implementeret i API'et. Disse brugerroller vil skulle validere hvorvidt visse knapper og funktionaliteter er tilgængelige i klienten, ud fra hvilke rettigheder den pågældende brugerrolle har. Derudover er der lavere-prioriterede brugsmønstre der ikke er blevet implementeret, herunder bl.a. muligheden for at se personprofiler og oprette kanal- og systemadministrator.
EPG Polleren skal gerne køre én i døgnet, for at hente nye produktioner og de tilhørende krediteringer. Dette er dog ikke implementeret, og skal derimod køres manuelt. Dertil er planen, at nye krediteringer skal oprettes direkte i klienten, hvilket gør at EPG Polleren kun skal bruges til at hente nye kanaler og produktioner. Ikke nye krediteringer.

Systemets lagdeling er modelleret ud fra et design mønstret, MVVM. Denne lagdeling gør at det meste af koden er separeret i forskellige pakker, og giver derved en lav afhængighed mellem disse. Dette gør dog også at det kan være besværligt at udvide programmet med ny funktionalitet, da der skal implementeres mange forskellige metoder og klasser. Samtidig kan MVVM strukturen være tidskrævende at sætte sig ind i. Systemets opdeling gør at man nemt kan udvide med en ny slags/form for klient, fx en webside eller en web app. Dette skyldes at pakkerne kommunikerer igennem interfaces og metodekaldene foregår til API'et, hvilket gør det muligt at udskifte klienten.
Som produktionerne er inddelt nu, er hver episode i en serie tildelt sin egen produktion. Optimalt set vil hver produktion indeholde tilhørende sæsoner og episoder, som hver indeholder dertilhørende krediteringer.

13 Konklusion

Det lykkedes gruppen at udvikle en prototype til et krediteringssystem, der giver mulighed for at erstatte rulletekster, efter endt TV-program. Med dette system kan man søge efter og finde krediteringer for programmer på forskellige kanaler, samt oprette nye krediteringer. Disse krediteringer skal være offentligt tilgængelige, og kunne redigeres og tilføjes af producere. Systemets opdeling af pakker følger design mønstret, MVVM, og gør det muligt at udskifte klienten til fx en website/webapplikation, da klienten kalder til et Rest API og gemmer i en database. Denne opdeling gør også at klasserne har en lav afhængighed. Allerede eksisterende data kan hentes via en EPG Poller fra TVTID.dk. De indledende forventninger er at gruppen har kunnet lave en fuldt funktionel prototype, hvilket er blevet opnået. Enkelte funktionaliteter mangler dog i desktop-klienten. Denne prototype er udviklet ved at følge Unified Process modellen og de tilhørende faser. Udformningen af kravspecifikationen og analysen viser hvilke krav og brugsmønstre systemet skal bygges op om, for at opfylde TV2's projektbeskrivelse.
INDSÆT NOGET OM DESIGN DELEN
INDSÆT NOGET OM TEST DELEN
Gruppen har, ud fra de udarbejdede resultater i krav, analyse og design, udviklet prototypen til et krediteringssystem.

- Hvordan skal folk refereres = ????????

14 Perspektivering

Produktet gruppen har udviklet giver mulighed for at TV2 og andre kanaler kan flytte rullekster fra TV og slutningen af programmer til en anden platform. Denne frigjorte plads kan derved udfyldes med reklamer og promoveringer for andre programmer. Eftersom systemet er bygget op omkring et Rest API, er det hurtigt og nemt at udskifte klienten til fx en website eller webapplikation. Derudover er systemet bygget op omkring muligheden for at tilføje flere kanaler end blot TV2, og er derfor en funktionel løsning på tværs af TV-stationer. Dette indebærer at oprette 'kanaler' til Boxer Play, Yousee, osv.
Løsningen gruppen har fundet frem til, har givet indsigt i hvorfor og hvordan man kan benytte et Rest API og EPG Poller. Dertil har brugen af design mønstret MVVM vist sig fordelagtigt til brug i større projekter med flere bidragsydere, grundet den lave afhængighed mellem klasser.

Til fremtidigt viderearbejdelse af projektet ville der først fokuseres på at færdiggøre prototypen, så den er fuldt funktionel, og indeholder alle brugsmønstre i MoSCoW analysen (se inceptionsdokumentet bilag **??**). Dette vil indebære at implementere brugerroller (system- og kanaladministrator, royal bruger, producer og gæst), så brugere tildelt en bestemt brugerrolle, vil have forskellige rettigheder i systemet, og derfor forskelligt indhold og redigerings/tilføjelses muligheder. Dette betyder, at muligheden for at oprette producere og system- og kanaladministrator skal implementeres, således at systemadministratorer kan oprette nye systemadministratorer, kanaladmistratorer og producere, og kanaladministratorer kan oprette producere. Derudover skal det være muligt at se personinformationer, godkende/afvise krediteringer og ændre sprog. Systemet vil skulle samle alle episoder for en produktion under én titel, og inddele disse i sæsoner. Det vil også give mening at lave en website/webapplikation til systemet. Websiten/webapplikationen vil have samme funktioner som desktop-klienten, altså kunne søge, oprette og redigere i krediteringer.

15 Procesevaluering

Denne sektion har til formål at fremvise gruppens evaluering af udviklingsprocessen og skriveprocessen. Sektionen er delt op i to delsektioner, den ene er gruppeevalueringen og den anden er gruppemedlemmernes individuelle evalueringer.

15.1 Gruppeevaluering

Nedenstående afsnit handler om, hvordan vi har evalueret processen i forhold til gruppearbejde, arbejdsformer, metoder og slutteligt hvad vi ville have gjort anderledes hvis vi skulle starte forfra med projektet.

15.1.1 Samarbejdet internt i gruppen

Arbejdet igennem projektforløbet har været generelt imødekommende. Samarbejdet har afspejlet det, idet at gruppen har været åbne for alle foreslag og tanker ligemeget hvilken opgave der var i fokus. Det var altid hjælp hvis man søgte det og det var ingen problemer med at afsætte tid til projektet. I de tilfælde at tid var en vanskelighed var gruppen tilbøjelig til at ændre mødetider eller sætte tiden andetstedes. Dog blev der respekteret at hvert medlem havde et liv uden for uddannelsen.

I starten af projektet brugte gruppen meget tid på at forventningsafstemme. Det var med til at skabe et fælles udgangspunkt for hvordan opgaven skulle gribes an, og skabte tidligt i processen et fælles mål at arbejde mod. Vi mener at det har gavnet vores projekt positivt. Efter at have dannet et fælles fundament blev gruppens Belbin roller taget i perspektiv, det viste sig dog at rollerne var vanskelige at afhænge af.
Vi fik hver især foretaget en Belbintest inden semesterstart, som vi skulle danne projektgrupper ud fra. Disse roller, mener vi, har ikke gavnet gruppearbejdet yderligere, da vi ikke føler at det er noget vi aktivt har benyttet os af. Vi mener, at teamrollerne på nuværende tidspunkt ikke er så sigende som de kunne være, da det stadig er meget nyt for os. Det, at det er vores medstuderende der har været med til at skabe vores Belbinprofil, gør også, at resultatet kan være afvigende fra hvordan man som enkelt person egentligt agere i et gruppeprojekt. Det kan skyldes at de fleste af os ikke har særlig meget erfaring med problemorienteret gruppearbejde, og derved ikke kan evaluerer hinandens roller i et projektarbejde.

Ledelse af projektet
Hvis vi skal kigge lidt på ledelsen af projektet, så fik vi i gruppen aldrig udvalgt en bestemt leder. Dette viste sig at have en negativ effekt på vores effektivitet i løbet af sprint 3. I denne periode var tællende aktivitet i de andre fag, hvilket gjorde at vi ikke var så produktive med implementeringen af brugsmønstrene. Vi manglede her en igangsætter/leder, der kunne sørge for at de issues vi havde i vores backlog blev lukket og derved at vores deadline blev overholdt. Dette er en erfaring vi tager med til fremtidige projekter, for at undgå lignende scenarier.

Arbejdsfordelingen i projektet
Da vi forventninsafstemmede, snakkede vi meget om hvor vores kompetencer lå, for at kunne opdele projektet bedst muligt, når nu vi valgte at lave et REST api og en EPG-Poller ud over desktop-klienten. Da en af vores gruppemedlemmer havde kompetencerne til at udvikle et REST api, fik vedkommende ansvaret for denne. Dette har gjort, at api'et primært er skrevet af en person, og at få af de andre har været med til at udvikle det. Dog har vi alle haft indflydelse

på funktionaliteten, og nogle af os været inddraget i udviklingsprocessen. Det samme gør sig gældende for EPG-Polleren. Set i retrospekt kunne vi med fordel have været mere inddraget i udviklingsprocessen i de to subsystemer, fremfor at vi nu skal sætte os ind i koden. Dette var dog noget vi som gruppe aftalte fra starten, og derfor er det ikke noget vi ser som et reelt problem.

Da vi i projektet benyttede os af værktøjet ZenHub, var vi meget opmærksomme på at få uddelegeret de issues vi havde i vores backlog, så alle havde lignende mængde arbejde foran sig. Der er nogle af os der har skrevet flere linjer kode end andre, men dette ser vi som uundgåeligt i et projekt som dette. Det kode man har haft ansvaret for at få skrevet, er blevet gennemgået af mindst et gruppemedlem, da vi på GitHub har sat reviewers på vores pull requests som skulle godkendes, inden koden kunne blive merget med vores master branch. Dette har gjort at vi let har kunne gennemgå hinandens kode og komme med eventuelle forbedringsforslag, for at optimere desktop-klienten.

15.1.2 Samarbejdet med vejleder

Overordnet set mener vi, at vores samarbejde med vejleder har været fint. Vi har for det meste kunne få fyldestgørende svar på de spørgsmål vi kom med, dog mener vi at feedback har været mangelfuld til tider. Dette gjorde sig blandt andet synlig efter afleveringen af 1. iteration, hvor vi godt kunne have ønsket mere uddybende feedback fra vejleder. Dette er dog ikke noget vi på noget tidspunkt har gjort vejleder opmærksom på, så vejleder har ikke haft mulighed for at rette op på det.

At vi har været nødsaget til at holde vejledermøder over Zoom, grundet den lidt kedelige situation, føler vi også at har haft en negativ effekt på kvaliteten af de møder vi har haft. Vi ser det som værende lettere at modtage vejledning når man sidder i samme rum, sammenlignet med før vi blev hjemsendt. Dette kan skyldes at alle 45 minutter sat af til vejledermødet blev brugt på at snakke, fremfor over Zoom, hvor vi sjældent kom over 15 minutter.

15.1.3 Projektarbejdsformen

Projektarbejdet har generelt været fint. Gruppen følte dog at arbejdsbyrden i starten var større end i det forrige projekt. Det kom af at vi blev introduceret for UP og dens process. Efter at have sat sig ind i hvad det indebar, var det dog ligetil at finde ud af hvordan projektet skulle gribes an. not sure pls come with input

15.1.4 Arbejdsformer

I skriveprocessen er der blevet anvendt to forskellige arbejdsformer, individuel skrivning og parskrivning. De har hver haft deres fordele og ulemper, blandt andet har det været godt at skrive selv, for så skal man ikke sidde og snakke om hvad man har tænkt sig at skrive, men på den anden side kan man godt være i tvivl om hvad man skal skrive. Her har parskrivning en af sine styrker, i at man kan sidde og diskutere hvad indholdet skulle handle om. Implementeringsprocessen har foregået lidt på samme måde, hvor man har lavet parprogrammering hvis der var steder man var lidt i tvivl om, ellers programmerede man individuelt. Parprogrammeringen har været med til at skabe løsninger, der ellers ikke ville blive udviklet, da man har siddet med to forskellige syn på hvordan noget skulle programmeres.

15.1.5 Metoder

De metoder der er anvendt i dette projekt er brugsmønsterdiagrammer, FURPS+ og UP & Scrum. Vi mener at formålet med brugsmønstrene er klart, og at vi har draget nytte af dem under inceptionsfasen. De har hjulpet under analysefasen og gjort det simpelt at holde styr på hvad hvert krav gik ud på og hvilke klasser og attributter der skulle bruges (dette blev synligt under navneordsanalyserne).
FURPS+ blev brugt til klassificering af ikke-funktionelle krav, og var med til at give en detaljeret beskrivelse af kravene. Dette har været med til synliggøre de krav der ikke umiddelbart påvirker funktionaliteten af systemet. vi kunne dog have gjort mere brug af FURPS+
MoSCoW blev brugt til at prioritere brugsmønstrene, og var med til at hjælpe gruppen med at fokusere hvad der skulle implementeres under implementeringen. Vi kunne dog have holdt os mere til vores MoSCoW, da ikke alle af vores musts blev implementeret. Dette er en erfaring vi vil tage med os til fremtidige projekter.
UP & Scrum er blevet brugt i sammenhæng i dette projekt, hvor vi ved hjælp af et scrumboard kunne holdte styr på udviklingen af artefakter for de enkelte faser i UP. Vi føler at vi har brugt UP efter hensigten, dog syntes vi, at der i starten af projektet var for meget fokus på analysedelen, og ikke for meget fokus på implementeringsdelen. Vi kunne godt have brugt Scrum mere seriøst. Med det menes der, at vi kunne have bedre til at udnytte de aktiviteter der finder sted i Scrum, såsom stand-up møderne og sprint review, da det kunne have hjulpet os med at undgå situationer som tidligere beskrevet, hvor der ikke blev arbejdet nok på projektet.

15.1.6 Skriveprocessen

Ved starten af projektet havde de fleste grupper et ønske om at rapportskrivning ville ske parralelt med udviklingen af produktet. Det kom af at medlemmerne havde erfaring med en stresset skriveprocess i forrige semester. Derfor blev der lagt vægt på at den skriftlige del af processen, blev gjort når det var muligt. Gruppen fik skrevet meget af rapporten under afleveringen af 1. iteration, dette har haft en positiv indflydelse på gruppens skriveproces op til afleveringen af det samlede projekt. En faktor der hjalp med at lette arbejdsbyrden, var en god arbejdsopdellingen, idet det var meget nemt at sætte sig på en del af opgaven og få feedback efter den var skrevet.

15.1.7 Den tidsmæssige styring af projektet

Til den tidsmæssige styring af projektet blev der taget udgangspunkt i den udleverede projektrammeplan. Det var ud fra denne af vi lavede vores egen tidsplan, hvor delafleveringer og andre epics blev indsat, så vi havde et overblik over de enkelte faser. Vi benyttede os af zenhub til dette, hvilket har haft en positiv indflydelse på styringen af projektet, da vi hele tiden har haft den opdaterede tidsplan. Da anvendelsen af Scrum var et krav fra projektets side, har denne metode også haft indflydelse på hvordan vi har planlagt projektet, som også blev beskrevet under metoder.

15.1.8 Hvad ville vi gøre anderledes?

Hvis vi skulle starte forfra med projektet, ville vi
Mere tid på implementering i starten af implementeringsprocessen fra størstedelen af gruppen
Fulgt vores burndownchart for at komme mere clean i mål skal laves efter konklusion

15.2 Individuelle evalueringer

Nedenstående afsnit handler om, hvordan det enkelte gruppemedlem evaluere processen, udover det der er beskrevet i gruppeevalueringen. Her kan der blandt andet læses om, hvordan medlemmerne selv evaluerer deres egen indsat og deres læringsmæssige udbytte.

15.2.1 Jakob Rasmussen

Personligt føler jeg at jeg har været godt involveret i projektet, fra start til slut. Der har dog været dele af projektet hvor jeg har følt at jeg ikke kunne bidrage, bl.a. Rest API'et. For det første er dette skrevet i python, hvilket gør det vanskeligt for mig at hjælpe. Derudover var hele konceptet nyt for mig, hvilket også vanskeliggør hjælp fra min side. Beslutningen om at bruge designmønstret MVVM har også gjort at jeg skulle bruge en del ekstra tid på at sætte mig ind i hvordan projektet var bygget op, men med det sagt har jeg også lært en del af det. Jeg har fået et fint kendskab til både Rest API og MVVM, og har fået en lidt bedre forståelse for python. Set tilbage burde jeg nok have spurgt mere ind til de ting jeg ikke forstod. Da jeg ikke selv var ansvarlig for API og EPG Poller, burde jeg have spurgt nærmere ind til hvordan de var bygget op. Det ville give et bedre overblik over systemet, da der er dele af det jeg ikke helt forstår. Gruppens størrelse mener jeg selv har fungeret fint. Jeg har selv lidt kodeerfaring, og har derfor lettere ved at sætte mig ind i andres kode. Havde jeg ikke haft kodeerfaring ville det nok have været en for stor mundfuld at skulle arbejde så mange sammen, på så stort et projekt.

15.2.2 Kenneth Christiansen

Hvordan har min egen indsat været? Hvad har jeg lært? Hvad har jeg ikke lært, som jeg måske burde have lært? Hvordan synes jeg at gruppestørrelsen har påvirket læringen fra semesterprojektet?

15.2.3 Kevin Petersen

Hvordan har min egen indsat været? Jeg synes jeg godt kunne have været mere igang til at starte med, til slut var jeg okay med, synes det ar ærveligt det allerade er slut. Hvad har jeg lært? Jeg har lært at code er ikke altid bare at skrive, det er at finde ud af hvordan man løser det bedst. Hvad har jeg ikke lært, som jeg måske burde have lært? Jeg burde have lært hvordan api'et authenticator personer og giver det tilbage som de må vide af data. Hvordan synes jeg at gruppestørrelsen har påvirket læringen fra semesterprojektet? Jeg synes gruppe størrelsen godt kunne have gjort det lidt uoverskueligt, hvad de andre arbejder på og hvad deres kode gør. Det kommer lidt til at lignde en blackbox. Jeg synes dog jeg har været godt til at læse hvad de har lavet og forstå det. Mangler dog at læse lidt mere på api'et. Der var mange ting jeg ikke vidste hvad var i python jeg var nød til at finde ud af først, men det gør jeg kan mentor de andre senere for at forstå koden.

15.2.4 Kristian Jakobsen

Hvordan har min egen indsat været? Hvad har jeg lært? Hvad har jeg ikke lært, som jeg måske burde have lært? Hvordan synes jeg at gruppestørrelsen har påvirket læringen fra semesterprojektet?

15.2.5 Mathias Rasmussen

Som helhed vurderer jeg projektet til at være forløbet nogenlunde smertefrit. Da jeg ikke har meget kodeerfaring, og stadig kæmper med basale færdigheder, har det resultereret i meget ekstra tid spenderet på at lære og forstå design mønsteret, MVVM, som gruppen har valgt at benytte. Når dette er sagt, har jeg lært en masse ved at være skubbet ud i at skulle kode mere kompliceret og virkelighedsnær kode. Jeg har derudover lært, at det at arbejde i en gruppe med gruppemedlemmer, der har mere programmeringserfaring end jeg selv, kan være med til at forårsage tvivl omkring kvaliteten af egen kode. Set i bakspejlet burde jeg have været bedre til at spørge om hjælpe, i stedet for at bruge unødvendig lang tid på at finde frem til en mulig tilgang. Da jeg ikke har været involeret i Rest API'et og EPG Polleren, ville jeg gerne have været mere inde over disse. Dette ville have givet et bedre overblik over det fulde system, og ikke blot desktop-klienten.
Jeg synes generelt arbejdsprocessen har været fin, og de andre gruppemedlemmer har været gode til at hjælpe til, når der blev spurgt til råds. Jeg synes dog at gruppens størrelse har gjort det en smule besværligt at holde overblikket, specielt da enkelte personer alene har lavet subsystemerne Rest API og EPG-Poller.

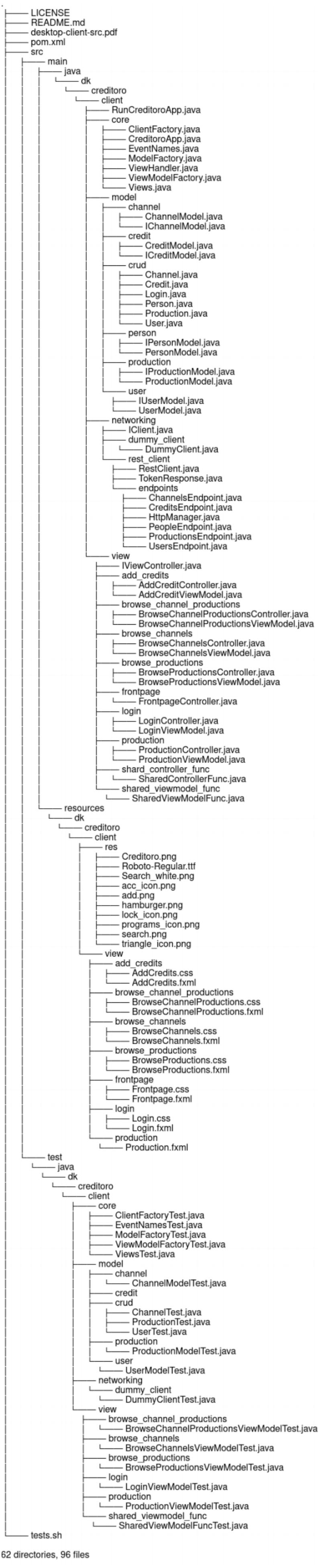
15.2.6 Simon Jørgensen

Hvordan har min egen indsat været? Hvad har jeg lært? Hvad har jeg ikke lært, som jeg måske burde have lært? Hvordan synes jeg at gruppestørrelsen har påvirket læringen fra semesterprojektet?

A Oversigt Over Kildekode

Begynder på næste side.

A.1 Filstruktur - desktop-klient



62 directories, 96 files

A.2 Filstruktur - EPG-Poller

43 directories, 49 files

A.3 Filstruktur - REST api

28 directories, 108 files

B Brugervejledning

REST api

Installationsvejledning
Kan læses på https://github.com/creditoro/api

Brugervejledning
Kan læses på https://github.com/creditoro/api/wiki/User-Manual

EPG-Poller

Installationsvejledning
Kan læses på https://github.com/creditoro/epg-poller

Brugervejledning
Kan læses på https://github.com/creditoro/epg-poller/wiki/User-Manual

Desktop-klient

Installationsvejledning
Kan læses på https://github.com/creditoro/desktop-client

Brugervejledning
Login detaljer:
Email: string@string.dk
password: string

Kan læses på https://github.com/creditoro/desktop-client/wiki/User-Manual

C Samarbejdsaftale

Forventninger & Mål

- Forventer alle yder sin del
- Der forventes der bliver lagt en passende mængde tid i arbejdet. fx 2 timer er ikke nok pr. Uge.
- Ambitionsniveauet er alle gør sit bedste, så vi i sidste ende føler at vi har gjort en hel hjertet indsats.

Gruppens Arbejdstider

- Kører med det akademiske kvarter
- Vi mødes kl 08, her under gælder det akademiske kvarter.
- Sygdom Læge eller lignende, skal man informere gruppen. Inden kl 08 på dagen.
- Gruppen forkrost pause, efter vejleder møde. 30 min frokost pause.
- Hvis der gives hjemme arbejde, skal det laves til det aftalte tidspunkt.
- Går hjem kl 14:00 medmindre andet aftales.

Gruppemøder

- Gruppen mødes fast.
- Møder forgår som udgangspunkt på SDU hver tirsdag, men der er muligheder for at kunne aftale andet i løbet af projektet hvis der er brug for det.
- Hvis man kommer for sent, kan der pålægges en straf, med mindre oversagen kan beggrundes.
- Straffen kan være: spiselige eller drikkelige ting.
- Hvis man ikke dukker op til møderne, skal det meddeles til medlemmerne helst inden vi mødes kl 08:00.
  - Hvis man gentagende gange udebliver, skal det løses internt i gruppen, ellers skal vejlederen inddrages.
- Hyggesnak godtages i passende mængder, men tiden skal bruges fornuftigt.

Organisering af møder

- Ordstyrer bliver valgt på dagen.
- Referent bliver valgt på dagen.
- I Gruppen vil man forsøge at skrive referat og logbog samlet.
  - Der er en der har hoveddeansvaret for referat og logbog, resten supplere til dem.
- Alle deadlines og indbyrdesaftaler overholdes og hvis det ikke er muligt, skal det skrives ind logbogen.
- Hver mødes skal skrives ind i logbogen og det vil sige dagsorden, eventuelt problemer osv.

Arbejdsindsats

- Der bliver aftalt fra gang til gang hvordan der skal arbejdes.
- Pair programmering
- GitHub: Merge request uden reviews

Vejledermøde

- Der holdes vejleder møde hver eneste uge
- Der sendes senest en møde indkaldelse med lokale, link til dagsorden, samt materialer, via outlook til vejlederen senest senest 23.59 Fredag.

Kursusdeltagelse

- Det forventes af hver gruppemedlem har læst op og fået styr på de emmer, vi arbejder med, inden møderne og arbejdsdagene.
- Der skrives så hvidt muligt kommentar i koden.
  - Kommentarende skrives på engelsk.

Brug & Revision Samarbejdsaftalen

- Samarbejdsaftalen tages i brug løbende.
- Aftalen tages i brug ved konflikter.
- aftalen kan blive revidereeget løbende, hvis der er behov for det.
  - Den nye aftale skal godkendes af alle medlemmer.

Værktøjer

- Primære kommunikationsmidler
  - Tekst: Discord, Mail
  - Samtaler: Discord
- Github
  - Kodebibliotek
  - Logbog
  - Referat
  - Aftaler
- Fildeling
  - OneDrive

Teamrolle	**Navn**	**Navn**	**Navn**
Idémand	Mathias	Kevin	Jakob
Kontaktskaber	Kevin	Kenneth	
Koordinator	Kristian	Mathias	
Opstarter	Simon	Mathias	
Analysator	Jakob		
Formidler	Kevin	Kristian	
Organisator	Kenneth	Simon	
Afslutter	Kenneth	Simon	
Specialist	Kristian	Jakob	

Tabel 14: Belbien Teamroller

Belbien Teamroller

Kontakt Oplysninger

- Jakob Rasmussen
  - jakra19@student.sdu.dk
  - 52 40 56 62
- Kenneth Munk Christiansen
  - Kechr19@student.sdu.dk
  - 28 67 66 78
- Kevin Kamper Meejach Petersen
  - kepet19@student.sdu.dk
  - 50 30 88 58
- Kristian Nymann Jakobsen
  - kjako19@student.sdu.dk
  - 22 80 53 26
- Mathias Nickolaj Rasmussen
  - mara816@student.sdu.dk
  - 28 12 89 41
- Simon Jørgensen
  - sijo819@student.sdu.dk
  - 42 83 25 60

D Vejlederaftale

- Der skal laves en gennemgang og dagsorden til hvert vejledermøde. Det skal sendes til vejleder senest fredag kl. 23.59.
- Alt skal så vidt muligt holdes i GitHub.
- Vejleder læser alle dokumenter og materialer igennen, vi sender, og dette skal gøres før vejledermødet.
- Vi tager ansvar for eget studie, og står selv for fremmøde til undervisning.
- Vejledermøde er som udgangspunkt hver tirsdag kl. 11.15.
  - Hvis tidspunktet ændres aftales det med vejleder via mail.
- Vejleder gør gruppen opmærksom på om gruppearbejdet er på afveje, samt at eventuelle deadlines ikke overholdes, så gruppen kan nå i mål uden konflikter.
- Vejleder giver besked via mail hvis han bliver forhindret i at afholde vejledermøde, eller hvis tidspunktet bliver ændret.
- Hvis gruppen skulle blive utilfreds med vejleders indsats, skal dette håndteres hurtigst muligt.
- Hvis gruppen kommer på afveje i forhold til samarbejdsaftalen, skal vejleder være i stand til at vejlede gruppen gennem eventuelle konflikter.

E Projektlog

Logbog - Github.com/creditoro/logbook

F Rapportkontrolskema

G Inceptionsdokument

Den kommer på den næste side.

H Andre Bilag?

I Detaljerede Brugsmønstre

Brugsmønster: Læs kreditering
ID: UC10
Primære aktører: Systemadministrator, kanaladministrator, producer, royalty bruger, gæst
Sekundære aktører:
Kort beskrivelse: Alle skal kunne se krediteringen for programmerne.
Prækonditioner (Pre conditions):
Hovedhændelsesforløb (main flow): 1. Brugsmønstret starter når en aktør vil se krediteringer for et program 2. Aktøren søger efter programmet 3. Aktøren trykker på det ønskede program 4. Systemet checker hvilken rolle aktøren har 5. Aktøren bliver omdirigeret til den passende visning af krediteringen
Postkonditioner (post conditions): En kreditering er blevet vist
Alternative hændelsesforløb (alternative flow): Step 2: Hvis programmet ikke findes, får vedkommende besked om at programmet ikke findes

Tabel 15: Brugsmønster: Læs kreditering

Brugsmønster: Opret producer
ID: UC03
Primære aktører: Systemadministrator, kanaladministrator
Sekundære aktører:
Kort beskrivelse: System- eller kanaladministrator opretter en producer, som kan oprette krediteringer i systemet
Prækonditioner (Pre conditions): Aktøren skal være logget på systemet
Hovedhændelsesforløb (main flow): 1. Dette brugsmønster starter, når en administrator vil oprette en ny producer 2. Administrator indtaster producerens oplysninger 3. Administratoren opretter produceren
Postkonditioner (post conditions): En producer er oprettet i systemet
Alternative hændelsesforløb (alternative flow): Step 3. Produceren eksisterer allerede, administrator får besked om dette ved oprettelse. Oprettelsen bliver ikke gennemført

Tabel 16: Brugsmønster: Opret Producer

Brugsmønster: Se Personinformationer
ID: UC04
Primære aktører: Systemadministrator, Kanaladministrator, Royality Bruger
Sekundære aktører:
Kort beskrivelse: System-, kanaladministrator og Royality Bruger får vist personinformation om en bestemt person.
Prækonditioner (Pre conditions):
Hovedhændelsesforløb (main flow): 1. Dette brugsmønster starter, når en primær aktør vil se information om en bestemt person 2. Den primære aktør søger efter en bestemt person ved at indtaste navn 3. Personer med det pågældende navn bliver vist 4. Den primære aktør vælger den person, der eftersøges 5. Informationen om den gældene person bliver vist
Postkonditioner (post conditions): Personinformationer er blevet vist
Alternative hændelsesforløb (alternative flow): step 4. Ingen person med det eftersøgte navn findes. Så bliver der angivet at der ikke findes nogle personer

Tabel 17: Brugsmønster: Se Personinformationer

Brugsmønster: Godkend nye krediteringer
ID: UC05
Primære aktører: Systemadministrator, kanaladministrator
Sekundære aktører:
Kort beskrivelse: Efter Produceren har sendt en kreditering ind til godkendelse, kan kanaladministratoren enten godkende eller afvise krediteringen. Hvis krediteringen godkendes, gøres krediteringen offentlig. Hvis krediteringen afvises, skal kanaladministratoren skrive en meddelelse om hvad der skal ændres.
Prækonditioner (Pre conditions): 1. Kanaladministratoren skal være logget ind 2. Produceren skal have sendt en kreditering ind til godkendelse
Hovedhændelsesforløb (main flow): 1. Dette brugsmønster starter når en producer har sendt en kreditering ind til godkendelse 2. Kanaladministratoren gennemgår krediteringen 3. Hvis alt er i orden, trykker kanaladministratoren på godkend.
Postkonditioner (post conditions): En kreditering er blevet godkendt og offentliggjort
Alternative hændelsesforløb (alternative flow): Step 3: Hvis noget skal ændres i krediteringen, trykker kanaladministratoren på afvis. Derefter kan kanaladministratoren skrive en meddelelse om rettelser

Tabel 18: Brugsmønster: Godkend nye krediteringer

Brugsmønster: Opret Person
ID: UC06
Primære aktører: Systemadministrator, kanaladministrator, Producer
Sekundære aktører:
Kort beskrivelse: Når en producer opretter krediteringer og en person der ikke allerede findes i systemet skal krediteres, skal produceren oprette vedkommende i systemet.
Prækonditioner (Pre conditions): 1. System-, kanaladministrator eller producer skal være logget ind
Hovedhændelsesforløb (main flow): 1. Dette brugsmønster starter når en system-, kanaladministrator eller producer skal indskrive krediteringer 2. Systemet checker om vedkommende allerede findes i systemet 3. Aktøren trykker 'Opret Person' 4. Opret Person-vinduet popper op 5. Aktøren udfylder de nødvendige informationer (navn, beskæftigelse, email, tlf.nr., osv.) 6. Aktøren trykker 'Færdig' 7. Personen bliver automatisk indskrevet det pågældende sted i krediteringen
Postkonditioner (post conditions): En person er blevet oprettet i systemet
Alternative hændelsesforløb (alternative flow): Step 5: Hvis aktøren blot har lavet en stavefejl, og en person med samme email, tlf.nr. osv. Findes, bliver aktøren oplyst om dette og oprettelse af personen afbrydes. Step 1-6: Aktøren kan til enhver tid afbryde oprettelsen

Tabel 19: Brugsmønster: Opret Person

Brugsmønster: Login
ID: UC07
Primære aktører: Gæst
Sekundære aktører:
Kort beskrivelse: En gæst kan logge ind hvis vedkommende har en bruger i systemet.
Prækonditioner (Pre conditions): Vedkommende skal ikke være logget ind
Hovedhændelsesforløb (main flow): 1. Dette brugsmønster starter når en aktør vil logge ind 2. Aktøren trykker login 3. Aktøren bliver omdirigeret til login-siden 4. Aktøren udfylder login-oplysninger 5. Aktøren trykker 'Login' 6. Aktøren bliver omdirigeret til startsiden
Postkonditioner (post conditions): Aktøren er logget ind
Alternative hændelsesforløb (alternative flow): Step 6: Hvis login-oplysningerne er forkerte, forbliver aktøren på login-siden og får en meddelelse om at brugernavn eller password var forkert Step *: Aktøren kan til enhver tid afbryde login

Tabel 20: Brugsmønster: Login

Brugsmønster: Logout
ID: UC08
Primære aktører: Systemadministrator, kanaladministrator, producer, Royalty Bruger
Sekundære aktører:
Kort beskrivelse: En aktør kan logge ud af desktopklienten
Prækonditioner (Pre conditions): Vedkommende skal være logget ind
Hovedhændelsesforløb (main flow): 1. Dette brugsmønster starter når en aktør vil logge ud 2. Aktøren trykker logout 3. Aktøren bliver omdirigeret til startsiden
Postkonditioner (post conditions): Aktøren er logget ud
Alternative hændelsesforløb (alternative flow):

Tabel 21: Brugsmønster: Logout

/

J Beskrivelser af klassekandidater

Navn	Produktion
Definition	En produktion har et UUID, en titel og et `kanal_id`
Eksempel	En produktion kan se ud som følgende: `UUID`: fbf20f43-9f1c-41e1-abce-cca32b836ef0 Titel: Peter pedal `kanal_id`: TV2
Andet	Nej

Tabel 22: Beskrivelse af Produktion-klassen

Navn	Kreditering
Definition	En kreditering indeholder `person_id` og `job` for de personer der skal krediteres og et `produktion_id` for den produktion de tilhører
Eksempel	En kreditering kan se ud som følgende: Hans Hansen Lydmand `produktion_id`: 21004623541527
Andet	Nej

Tabel 23: Beskrivelse af Kreditering-klassen

Navn	Kanal
Definition	En kanal har et `ID` og et `navn`
Eksempel	En kanal kan se ud som følgende: `ID`: 840215538 `navn`: TV2
Andet	Nej

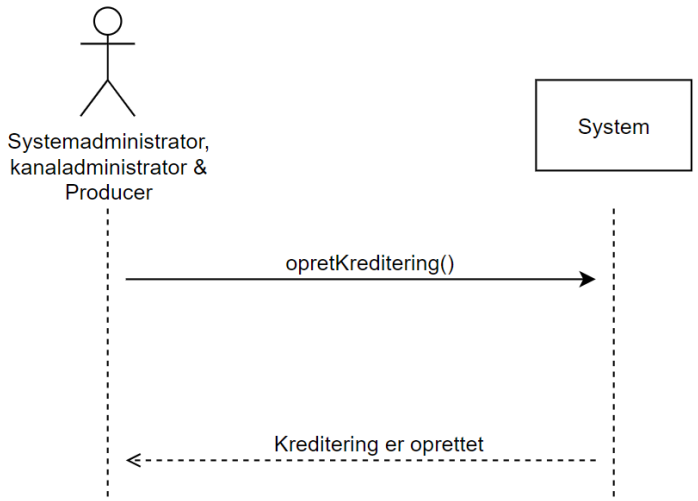
Tabel 24: Beskrivelse af Kanal-klassen

Navn	Person
Definition	En person har personinformationer (navn, beskæftigelse, e-mail, tlf.nr. osv)
Eksempel	En person kan se ud som følgende: Navn: Hans Hansen E-mail: hans@hansen.pm Telefonnummer: + 42 42 06 96 66 Beskæftigelse: Lydmand
Andet	Nej

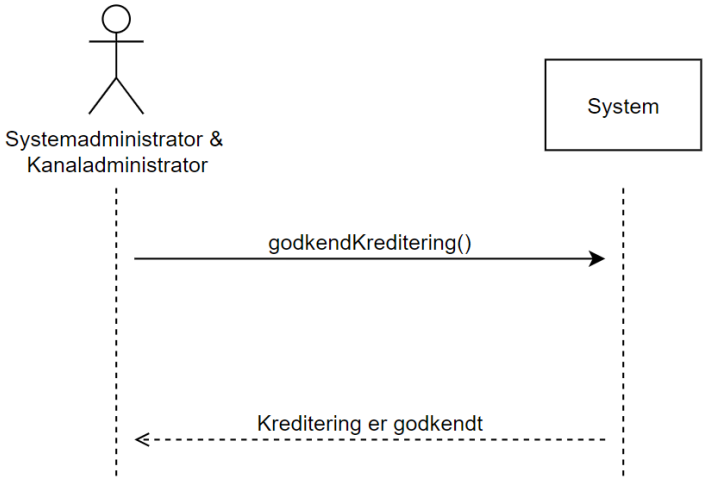
Tabel 25: Beskrivelse af Person-klassen

/

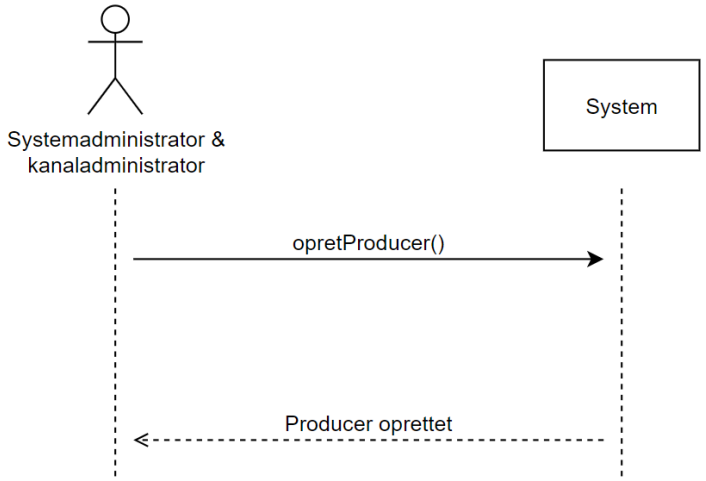
K Systemsekvensdiagrammer



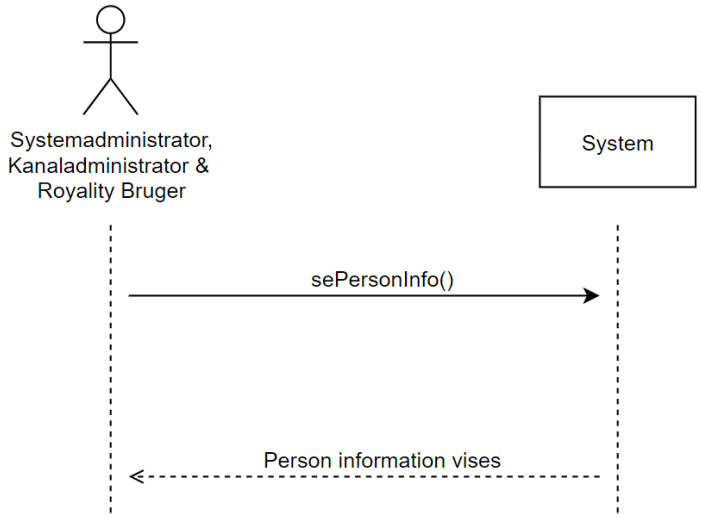
Figur 10: Systemsekvensdiagram for "Opret kreditering"



Figur 11: Systemsekvensdiagram for "Godkend kreditering"



Figur 12: Systemsekvensdiagram for "Opret producer"



Figur 13: Systemsekvensdiagram for "Se personinformation"

/

L Kontrakter for systemfunktioner

Læs Kreditering	
System operation	**læsKreditering**
Krydshenvisning	Use case: Læs kreditering
Ansvar	At vise eksisterende krediteringer, hvis følgende betingelser er sande 1. Den søgte kreditering findes i systemet Hvis ikke overstående er sande, skal det sendes en besked til brugeren at den søgte kreditering ikke findes i systemet
Output	1. Krediteringen vises Alternativt: Besked sendt ud til brugeren
Prækonditioner	Ingen
Postkonditioner	En kreditering vil blive vist

Tabel 26: Systemfunktionskontrakt 'læsKreditering'

Opret producer	
System operation	**opretProducer**
Krydshenvisning	Use case: Opret producer
Ansvar	At oprette en producer, hvis prækonditionen er opfyldt og følgende betingelser er sande: 1. Produceren eksisterer ikke i systemet i forvejen 2. Alle oplysninger er indtastet korrekt Derefter give brugeren besked om det lykkedes at oprette en producer eller ej
Output	Besked om der er blevet oprettet en producer eller ej
Prækonditioner	Vedkommende der prøver at oprette en producer skal være logget ind som 'Kanal- eller Systemadministrator'
Postkonditioner	En ny producer er oprettet i systemet

Tabel 27: Systemfunktionskontrakt 'Opret producer'

Se Personinformationer	
System operation	**sePersonInfo**
Krydshenvisning	Use case: Se personinformation
Ansvar	At vise respektive persondata om den søgte person, hvis prækonditionen og betingelsen er opfyldt: 1. Personen er logget ind som enten producer, kanal- eller systemadministrator Hvis ikke overstående er opfyldt, vises ikke personfølsomme informationer, som navn, roller i film/serier som personen har medvirket i
Output	Viser en information om en person
Prækonditioner	Person er fundet i systemet
Postkonditioner	Alt efter ens rolle i systemet vil forskelligt data blive vist. Begrænset hvis man er besøgende, alt hvis man er producer, royalty bruger, kanal- og systemadministrator

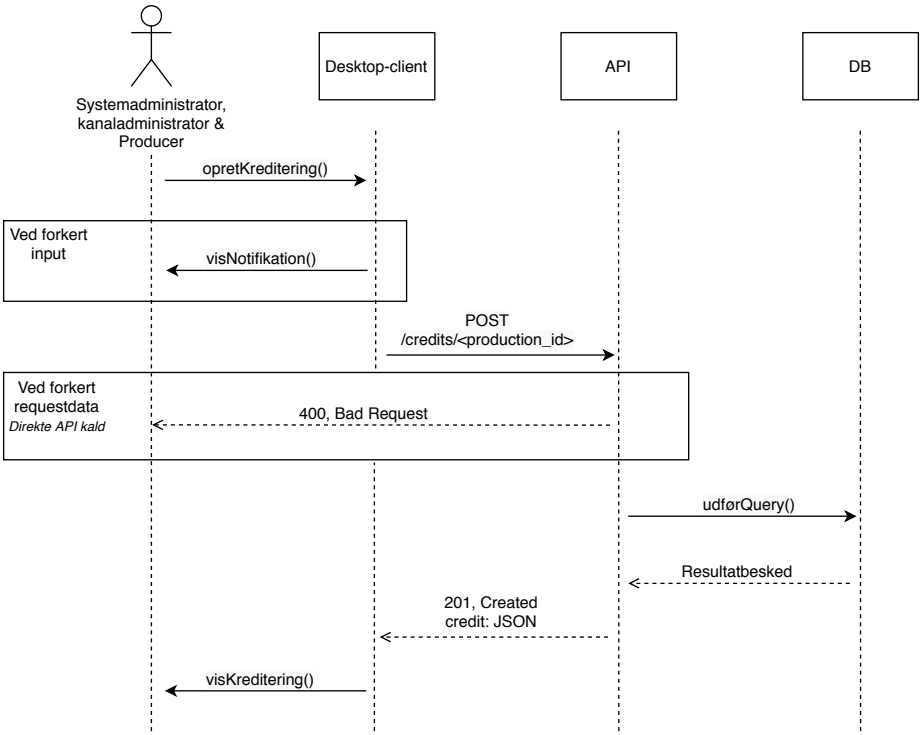
Tabel 28: Systemfunktionskontrakt 'se personinformationer'

Godkend ny kreditering	
System operation	**godkendKreditering**
Krydshenvisning	Use case: Godkend Kredtiteringer
Ansvar	At godkende eller afvise nye krediteringer hvis prædonditionerne er sande og betingelsen opfyldt: 1. En kreditering er oprettet af en producer Hvis ikke ovenstående er opfyldt, vil der ikke blive vist nye krediteringer til godkendelse.
Output	Besked om krediteringen er godkendt
Prækonditioner	1. Er logget ind som system- eller kanaladministrator
Postkonditioner	Krediteringen er enten godkendt eller afvist og den ansvarlige producer er bliver informeret herom

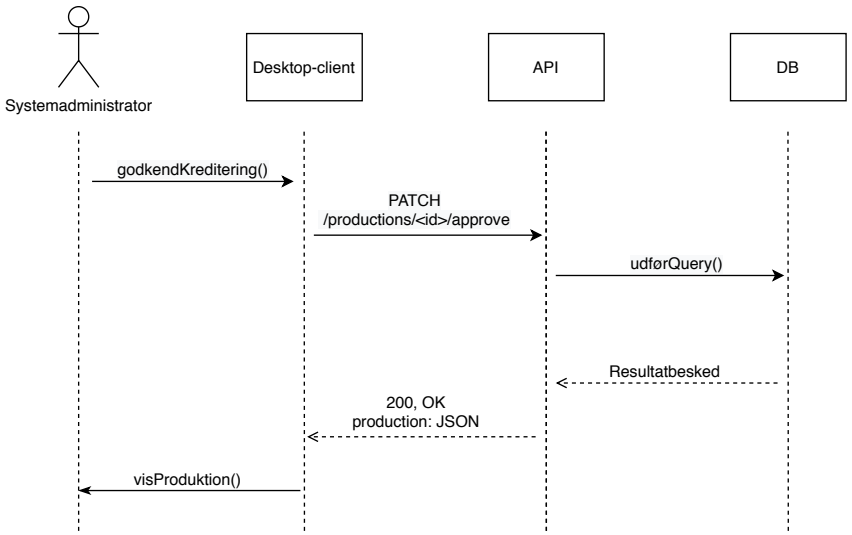
Tabel 29: Systemfunktionskontrakt 'Godkend nye krediteringer'

/

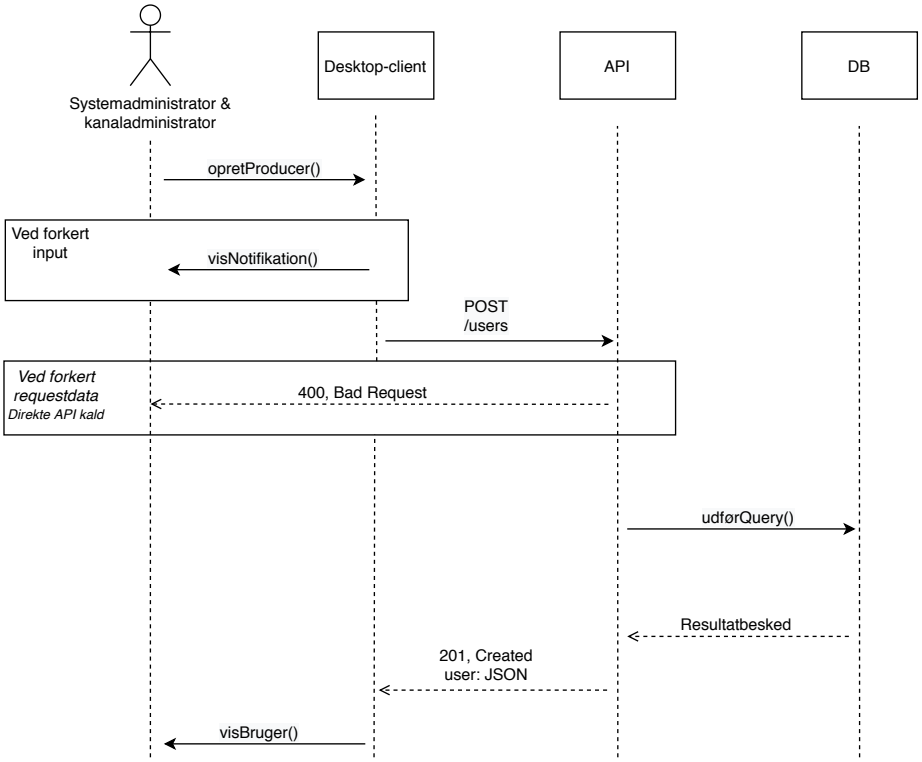
M Operationssekvensdiagrammer



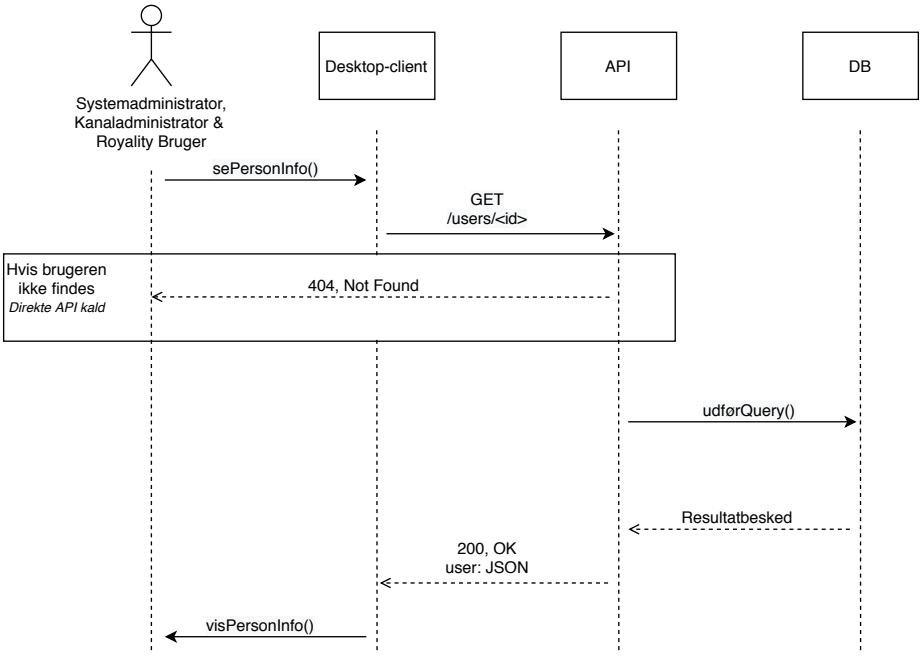
Figur 14: Operationssekvensdiagram for "Opret kreditering"



Figur 15: Operationssekvensdiagram for "Godkend kreditering"



Figur 16: Operationssekvensdiagram for "Opret producer"



Figur 17: Operationssekvensdiagram for "Se personinformation"

/

N Analyse - REST api & EPG-Poller

Dette afsnit har til formål at fremlægge analyseprocessen for REST api'et og EPG-Polleren.

Rest API
Kristian LIGE HER DO //TODO
Noget om hvad api'et skal bruges til. For at kunne opnå dette, skal api'et bestå af følgende:

api pakken
Api pakken Skal indeholde pakkerne: channels, credits, people, productions, users og models.

channels, **credits**, **people**, **productions** og **users Pakkerne**.
Disse pakker skal bestemme hvilken tabel der skal skal laves, hvordan tabellen skal sættes op, hvilke endpoints der skal være tilgængelige og hvilke forespørgsler der kan laves til databasen.

models pakken
Models pakken skal indeholde klasser der bestemmer måden hvorpå data bliver pakket på, inden det sendes ud til den der har lavet forespørgsler.

klassen auth.py
auth.py klassen skal håndtere verificeringen af brugere, altså hvem der har adgang til hvilke funktioner i systemet.

EPG Poller
For at kunne hente EPG data fra TVTid.dk skal der benyttes en poller, der har til formål at holde databasen til projektet opdateret med den data der er tilgængelig på TVTid.dk. EPG Polleren skal sende en forespørgsel til det api der er koblet op til TVTid.dk, og bagefter håndtere det data der modtages. For at opnå dette, skal EPG polleren bestå af tre pakker:

core pakken
Core pakken skal indeholde de klasser der indeholder logikken til at forespørge og håndtere de JSON objeter der modtages fra TVTid.dk api'et, og derefter sende objekterne videre til det api der er blevet udviklet af projektgruppen, kaldet Creditoro api'et.

networking pakken
Netværkspakken skal indeholde logik for at kunne hente data fra det forskellige http services, TVtid.dk api'et og Creditoro api'et. Dette skal ske ved hjælp af modeller, som er placeret i en undermappe:

models pakken
Model pakken skal indeholde de modeller, der skal bruges til at hente og få data fra de førnævnte http services. Der skal laves to modeller, en get og en post. Get modellen skal bruges til at kortlægge det data der bliver modtaget fra TVTid.dk api'et 1 til 1, så det bliver en tro kopi af det data. Post modellen skal bruges til at videresende det data til Creditoro api'et.
/

O Subsystemdesign - REST api & EPG-Poller

Rest API

Overvejelser
Processen for design og implementationen var ret klar, da en af vores gruppe medlemmer

arbejder med dette. Dog var prioriteringen af rækkefølgen som API'et skulle implementeres i, noget der blev overvejet nøje.

Beslutninger
For at undgå udviklings blockers for de andre systemer der skulle interagere med API'et (desktop klienten, og EPG Polleren) besluttede vi at lave alle endpointsne først, og senere implementere hvilke bruger roller det kræves for de forskellige operationer.

Resultater
Vi besluttede at udvide vores designklassediagram for API'et en smule ved at sætte labels på hvilke endpoint, de forskellige metoder for hver klasse hører til. På den måde var det let at se for de andre applikationer at se, hvad de kunne forvente af hvert endpoints.

EPG-Poller

Overvejelser
Da EPG-Polleren skulle designes, blev det overvejet om der skulle tages udgangspunkt i samme mappestruktur som i desktop-klienten, dog uden MVVM delen, da dette ikke ville være relevant i EPG-Polleren. Overvejelserne gik på at lave en mappestruktur, hvor hver modul fik sin egen pakke.

Beslutninger
Det blev besluttet at gå med samme mappestruktur som i desktop-klienten, for at holde det ens gennem projektet. Dette vil også have den fordel, at logikken for de forskellige moduler vil blive placeret i hver deres mappe.

Resultater
Overordnet har vi Creditoro, hvor der ligger 3 pakker, der hedder core, networking og models. I core mappen er der en klasse som indeholder al kernelogikken og læser miljø variabler, som bestemmer hvilket password den kører med. I networking mappen, er der klassen HttpManager. Den har logikken for at hente og sætte data. I models mappen er der de java datamodeller der bliver brugt til at hente og sætte data, de skal kortlægges en til en med json Objekterne.
/

P Databasedesign - Resultater

Tabeldesign

channels

```
1 CREATE TABLE channels
2 (
3     identifier  UUID NOT NULL,
4     name        VARCHAR(512),
5     icon_url    VARCHAR(512),
6     PRIMARY KEY (identifier)
```

```
);

CREATE INDEX channels_name_idx ON channels USING btree (name);
```

Listing 16: channels.sql

identifer	name	icon_url
a4226ce8-e973-4de3-906b-ac4a696c7cdb	TV2	https://epg-images.tv2.dk/channellogos/logo/3.png

Tabel 30: Tabeldesign: channels

users

```
CREATE TABLE users
(
    identifier  UUID NOT NULL,
    name        VARCHAR(512),
    email       email UNIQUE NOT NULL,
    phone       VARCHAR(256),
    password    VARCHAR(256),
    role        role NOT NULL,
    created_at  TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (identifier)
);

CREATE INDEX users_email_idx ON users USING btree (email);
CREATE INDEX users_name_idx ON users USING btree (name);
```

Listing 17: users.sql

identifer	name	email	phone	password	role	created_at
2f477718-3351-4d97-95dd-12518675ed52	Simon	string@string.dk	78652397	string	3	2020-04-20

Tabel 31: Tabeldesign: users

people

```
CREATE TABLE people
(
    identifier  UUID NOT NULL,
    phone   VARCHAR(256),
    email   email,
```

```
6       name      VARCHAR(256),
7       PRIMARY KEY (identifier)
8   );
9
10  CREATE INDEX people_name_idx ON people USING btree (name);
```

Listing 18: people.sql

identifer	name	phone	email
7fae265e-84d7-4b65-b14e-4ec8819109a7	Kristian	42066669	string2@string2.dk

Tabel 32: Tabeldesign: people

productions

```
1   CREATE TABLE productions
2   (
3       identifier  UUID NOT NULL,
4       title       VARCHAR(256) NOT NULL,
5       created_at  TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
6       producer_id UUID REFERENCES users (identifier),
7       channel_id  UUID REFERENCES channels (identifier),
8       description TEXT,
9       PRIMARY KEY (identifier)
10  );
11
12  CREATE INDEX production_title_idx ON productions USING btree (title);
13  CREATE INDEX production_identifier_idx ON productions USING btree (
        identifier);
```

Listing 19: productions.sql

identifer	title	created_at	producer_id	channel_id
58cb7c91-ed0a-4bd3-80b8-606690441293	Stuff	2008-11-11	2f477718-3351-4d97-95dd-12518675ed52	a4226ce8-e973-4de3-906b-ac4a696c7cdb

Tabel 33: Tabeldesign: productions

credits

```
1   CREATE TABLE credits
2   (
3       identifier      UUID NOT NULL,
4       job             VARCHAR(256) NOT NULL,
5       production_id   UUID REFERENCES productions (identifier),
6       person_id       UUID REFERENCES people (identifier),
```

```
7       PRIMARY KEY (identifier)
8   );
9
10  CREATE INDEX credit_job_idx ON credits USING btree (job);
11  CREATE INDEX credit_identifier_idx ON credits USING btree (identifier);
```

Listing 20: credits.sql

identifer	job	production_id	person_id
04f699d1-52b6-4eb2-ad88-04f48f93b771	Lydmand	58cb7c91-ed0a-4bd3-80b8-606690441293	7fae265e-84d7-4b65-b14e-4ec8819109a7

Tabel 34: Tabeldesign: credits

SQL forespørgelser

```
87  -- SQL queries
88  -- Insert data into tables
89  INSERT INTO users (name, email, phone, password, role)
90  VALUES ('Peter Leasy', 'en@mail.dk', '88888888', 'S0m3Pa55W0rD', '
        system_admin');
91
92  INSERT INTO channels (name, icon_url)
93  VALUES ('TV2', 'https://epg-images.tv2.dk/channellogos/logo/3.png');
94
95  INSERT INTO people (phone, email, name)
96  VALUES ('12312312', 'enAnden@mail.dk', 'Jakob Jakobsen');
97
98  INSERT INTO productions (title, producer_id, channel_id, description)
99  VALUES ('Bonderøven', '6ed67a28-f288-492b-8947-d9d0e9539608', '3ee08c85-466f
        -4577-addc-825815ae8ad1', 'Frank har bygget en stor æblepresser     men æ
        blerne skal lige samles, inden den kan tages i brug. Og så er der alle de
         andre opgaver, der presser sig på! Det når at blive snestorm, før der
        laves æblesaft. Til gengæld er den iskold!');
100
101 INSERT INTO credits (job, production_id, person_id)
102 VALUES ('Lydmand', '3ee08c85-466f-4577-addc-825815ae8ad1', '99bed7ec-8bf5-4
        be9-8514-fce0a37b028b');
103
104
105 -- Get specific production from productions table
106 GET * FROM productions WHERE title='Bonderøven';
107
108 -- Get name from email
109 GET name FROM people WHERE email='enAnden@mail.dk';
110
111 UPDATE credits SET job='Producer' WHERE identifier='3e8a6e59-5e08-4f02-b752
        -276713771a8a';
112
113 -- Delete a credit
114 DELETE FROM people WHERE identifier='3e8a6e59-5e08-4f02-b752-276713771a8a';
```

Listing 21: 206 SQL.sql

/

Q Implementering - REST api og EPG-Poller

REST api

EPG-Poller

/

R Test - REST api & EPG-Poller

1. Iteration af api testning

Test af Users klassen

UsersTest klassen importerer metoder fra klassen BestTestCase. Metoden test_all er den der foretager testen på users ved at bruge metoderne i BestTestCase.

```
1  from http import HTTPStatus
2
3  from tests.base_test import BaseTestCase
4
5  class UsersTest(BaseTestCase):
6      path = "users"
7
8      def test_all(self):
9          self.get_list()
10
11         response = self.post(json=self.json())
12         identifier = response.json.get("identifier")
13         self.patch(identifier=identifier, json=self.patch_json())
14         self.get(identifier=identifier
15         self.put(identifier=identifier, json=self.json())
16         self.delete(identifier=identifier)
```

Listing 22: UsersTest.py

Der startes med at kalde get_list på self, som er en metode importeret fra BestTestCase og ser således ud:

```
55 def get_list(self):
56     response = self._get(path=f"/{self.path}/")
57     self.assertTrue(response.status_code == HTTPStatus.OK)
58     return response
```

Listing 23: BestTestCase.py

Herefter bliver der lavet et response objekt, som bliver instantieret ved at kalde post metoden i BestTestCase med en parameter der bliver dannet i json metoden fra UsersTest klassen.

```
12 response = self.post(json=self.json())
```

Listing 24: UsersTest.py

```
19 def json(self, phone: str = None, email: str = None, name: str = None,
       password: str = None):
20     password = password or self.random_string()
21     return {
22         "phone": phone or "+45 12 12 12 12",
23         "email": email or f"{self.random_string()}@creditoro.nymann.dev",
24         "name": name or self.random_string(),
25         "password": password,
26         "repeated_password": password
27     }
```

Listing 25: UsersTest.py

Json har muligheden for at tage parametre, men behøver det ikke. Så hvis man ikke ønsker en specifik json får man en forudindstillet json. Det json der bliver oprettet har et telefonnummer, email, navn, password og repeated_password. Efter json er sat op kan det blive videregivet til post metoden i BestTestCase:

```
65 def post(self, data: dict = None, json: dict = None):
66     response = self._post(path=f"/{self.path}/", data=data, json=json)
67     self.assertTrue(response.status_code == HTTPStatus.CREATED)
68     return response
```

Listing 26: BestTestCase.py

Post metoden starter med at oprette et repsons objekt, ved at kalde metoden _post med det json der blev videregivet.

```
40 def _post(self, path: str, data: dict = None, json: dict = None):
41     return self.client.post(path, headers=self.headers, json=json, data=data
           )
```

Listing 27: BestTestCase.py

I _post metoden der kaldt en post metode på klienten, der returnerer et respons objekt.
I post metoden bliver respons objektet sat til det returnerede responsobjekt. Herefter bliver der testet om at HTTPStatus er CREATED ved brug af assertTrue der tjekker om response objektets status_code er det samme som OK. Eller fejler testen. Og til sidst returnerer response objektet til test_all metoden.

Efter at have oprettet response objektet, bliver der oprettet et identifier objekt. Dette objekt bliver sat til response objektets identifier værdi. Identifier bliver herefter brugt til at kalde patch metoden, der tager en identifier og værdien som patch_json metoden returnere som parametre.

```
14 self.patch(identifier=identifier, json=self.patch_json())
```

Listing 28: UsersTest.py

```
29 def patch_json(self, name: str = None):
30     return {
31         "name": name or self.random_string()
32     }
```

Listing 29: UserTest.py

Patch_json ændrer enten navnet til noget bestemt eller tilfældigt, og returnerer det reviderede json objekt. De to parametre, identifier og json, bliver så videre givet til patch metoden i base_test.

```
70 def patch(self, identifier: str, data: dict = None, json: dict = None):
71     response = self._patch(path=f"/{self.path}/{identifier}", data=data,
           json=json)
72     self.assertTrue(response.status_code == HTTPStatus.OK)
73     return response
```

Listing 30: BestTestCase.py

I patch metoden bliver der oprettet et nyt response objekt ved at kalde _patch metoden. _patch metoden returnere, ligeseom _post metoden, et response objekt. Response objektet bliver igen testest igennem assertTrue metoden, så hvis responsens status_code er OK, består testen.

Efter patch metoden bliver kaldt, bilver get metoden kaldt. Get metoden tager identifier objektet som parameter.

```
60 def get(self, identifier):
61     response = self._get(path=f"/{self.path}/{identifier}")
62     self.assertTrue(response.status_code == HTTPStatus.OK)
63     return response
```

Listing 31: BestTestCase.py

I get metoden bliver der, ligesom i patch metoden, oprettet et respons objekt. Her bliver det dog oprettet ved at kalde _get metoden. Efter oprettelsen bliver den, ligesom de andre eksempler, testet for dens status_code, der skal være `OK` for at testen lykkes.

Delete metoden bliver kaldt som den sidste i TetstUser test_all metoden, og tager identifier objektet som paramter. Først bliver response objektet oprettet ved at kalde _delete metoden, der returnere et response objekt. Herefter tester assertTrue metoden om reponse status_code er lig med no_content. De pågældene metoder kan ses nedenfor:

```
17 self.delete(identifier=identifier)
```

Listing 32: UsersTest.py

```
80 def delete(self, identifier: str):
81     response = self._delete(f"/{self.path}/{identifier}")
82     self.assertTrue(response.status_code == HTTPStatus.NO_CONTENT)
83     return response
```

Listing 33: BestCaseTest.py

Testning EPG-Poller

Test af HttpManger klassen

HttpMangerTest klassen importerer klasserne Test, LoadConfig, CreditoroChannel, Creditoro-Production, Assertions.*, LocalData, Arrays. Disse bliver brugt til at Checke om HttpManger Kan give de rigtige resultater.

```
1  package dk.creditoro.epg_poller.networking;
2
3  import org.junit.jupiter.api.Test;
4
5  import dk.creditoro.epg_poller.core.LoadConfig;
6  import dk.creditoro.epg_poller.models.CreditoroChannel;
7  import dk.creditoro.epg_poller.models.CreditoroProduction;
8
9  import static org.junit.jupiter.api.Assertions.*;
10
11 import java.time.LocalDate;
12 import java.util.Arrays;
13
14 class HttpManagerTest \{
15     private HttpManager httpManager;
16     private static String USER = LoadConfig.getLoadconfig().getUser();
17     private static String PASSWORD = LoadConfig.getLoadconfig()
18         .getPassword();
19     public HttpManagerTest() {
20         httpManager = new HttpManager();
21     }
```

Listing 34: HttpManagerTest.java

Her bliver der sat USER og PASSWORD, som skal bruges til at logge ind på api'et. LoadConfig står for at hente miljø variablerne. Ved at gøre det denne måde kan, man altid skifte i hvordan den læser passwords. Der efter bliver httpManager initialiseret.

```
23     @Test
24     void getTvTidChannels() {
25         var channels = httpManager.getTvTidChannels();
26         assertNotNull(channels);
27         var tvTidChannels = channels.getChannels();
28         assertNotNull(tvTidChannels);
29         assertTrue(tvTidChannels.length > 0);
30     }
```

Listing 35: HttpManagerTest.java

Her bliver det testet om den kan få kandler ned fra tvtid.dk Der bliver testet om den ikke er null. Checker om den kan give en liste tilbage på linje 28. Checker om længen er størrer end null på linje 29. Den ud pakker der bliver brugt, pakker null ud, hvis json object skifter. Der ved man at hvis tvtids data skifter eller man laver en opdatering der gør det ikke virker mere ville denne her test gøre det synligt.

```
32     @Test
33     void postChannel() {
34         httpManager.login(USER, PASSWORD);
35         //Create the channel that we are testing
36         var channel = new CreditoroChannel("testChannel", "testURL");
```

```
37          // Makes sure to delte the channel before we test if it can post.
38          var getRespone = httpManager.getChannels(channel.getName());
39          if (getRespone.length != 0){
40              httpManager.deleteChannel( getRespone[0].getIdentifier() );
41          }
42          // Post the first channel
43          var responeChannel = httpManager.postChannel(channel);
44          assertEquals(channel.getName(), responeChannel.getName(), "Makes
                sure it it gets the same channels as it posted");
45          // post the samme channel again
46          var dublicatedResponeChannel = httpManager.postChannel(channel);
47          assertNotEquals(channel, dublicatedResponeChannel, "Makes sure it
                can't post dublicate");
48          // Delte the channel
49          getRespone = httpManager.getChannels(channel.getName());
50          var httpRespone = httpManager.deleteChannel( getRespone[0].
                getIdentifier() );
51          assertEquals(204, httpRespone, "Checks if we can delte channel");
52      }
53
54  \}
55  Her bliver metoden postChannel på httpMangeren testet.
```

Listing 36: HttpManagerTest.java

S Funktionelle Krav

Desktop-client & REST API		
ID	**Name**	**Description**
D01	Login/out	You should be able to login and out of the system with ease by providing username and password.
D02	Search	You should be able to search for TV-show titles as well as persons
D03	Channel Browse	You should be able to browse channels registered in the system, and see which TV-shows are streamed on these
D04	Display credits associated with TV-program	You should be able to display all credits associated with a TV-program
D05	Create Channnel	The system admin should be able to create new channels.
D06	Add system admin	The system admin should be able to add other system admins.
D07	Add Channel admin	The system admin and channel admin should be able to add channel administrators to a channel. (Channel admin only for own channel)
D08	Add Producer	The system admin and channel admin should be able to add producers for a show (channel admin only for own channel)
D09	Create credit	The system admin, channel admin and producer should be able to create a credit (channel admin and producer only for own channel)
D10	Delete credit	The system admin and channel admin should be able to delete credits (channel admin only for own channel)
D11	Update credit	The system admin, channel admin and producer should be able to update and change credit (channel admin and producer only for own channel)
D12	Update person	System admin, channel admin and producer should be able to update a person, mainly which shows the person has helped produce
D13	Approve or disapprove credit	The system admin and channel admin should be able to approve or disapprove created credits before they are publicly available (channel admin only for own channel)
D14	Create royalty user	The system admin should be able to create a royalty user
D15	Update royalty user	The system admin and royalty user should be able to update royalty users
D16	Change language	You should be able to change the language of the user interface

Tabel 35: Desktop-client & REST API: Functional Requirements

EPG Poller		
ID	**Name**	**Description**
E01	Poll-data	The EPG Poller should be able to poll data from tvtid.dk
E02	Update Database	Update the database with credits

Tabel 36: EPG Poller: Functional Requirements

T Ikke-funktionelle Krav

REST API		
ID	**Name**	**Description**
NR01	Supportability	Centralized error reporting should be available via a common interface (such as Sentry).
NR02	Performance	Commonly used API calls should respond within 300 milliseconds.
NR03	Scalability	The system should be able to handle 10K new users yearly for 25 years.
NR04	Scalability	The system should be able to handle 15K new credits yearly for 25 years.
NR05	Configuration	Data persistence time should be configurable, to auto-cleanup data older than the configured value (defaults to 25 years).
NR06	Availability	The system should start automatically after server restarts.
NR07	Usability	API documentation should be available via Swagger UI.
NR08	Security	The server that the REST API is hosted on should only allow login via SSH.
NR09	Security	The server that the REST API is hosted on should only allow connections from the outside on port 443 (https), 80 (http) and 22222 (ssh).
NR10	Installability	REST API should be deployable within a container (such as Docker).
NR11	Configurability	REST API should be configurable using **environment** files.
NR12	Authentication	REST API should handle authentication via token based authentication with it's clients. A token is valid for 2 hours, and is refreshed automatically after a request with the token when it's time to expire is less than an hour.

Tabel 37: REST API: Non-functional Requirements

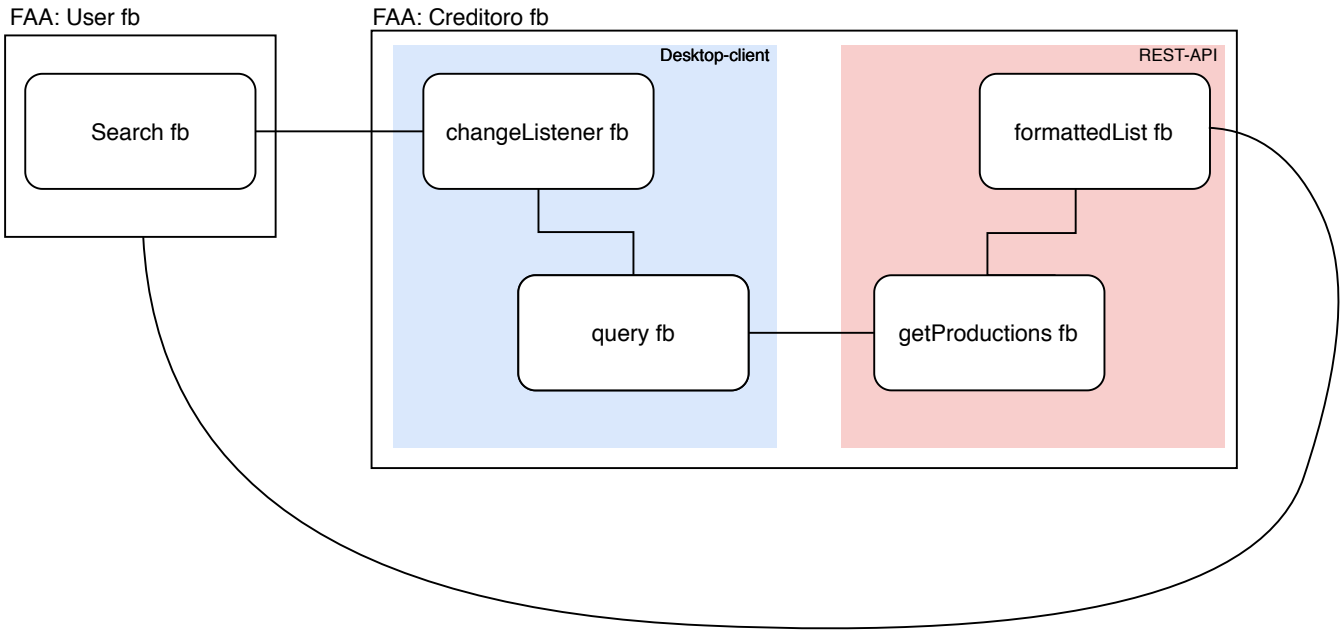
Desktop-client		
ID	**Name**	**Description**
ND01	Supportability	Error reporting should be available via pop-out box
ND02	Performance	Commonly used buttons should work within 300 milliseconds
ND03	Scalability	The system should have a nice overview of all credits
ND04	Scalability	The client should be able to query search results in 5 seconds
ND05	Configuration	Configuration should be configurable using **environment** files and GUI
ND06	Availability	The system should start automatically after server restarts.
ND07	Authentication	The system should redirect to the login page when receiving a HTTP 401 (unauthorized) response from the REST API.
ND08	Inactivity	The desktop-client should automatically detect if the authentication token is expired and redirect to login page.

Tabel 38: Desktop Client: Non-functional Requirements

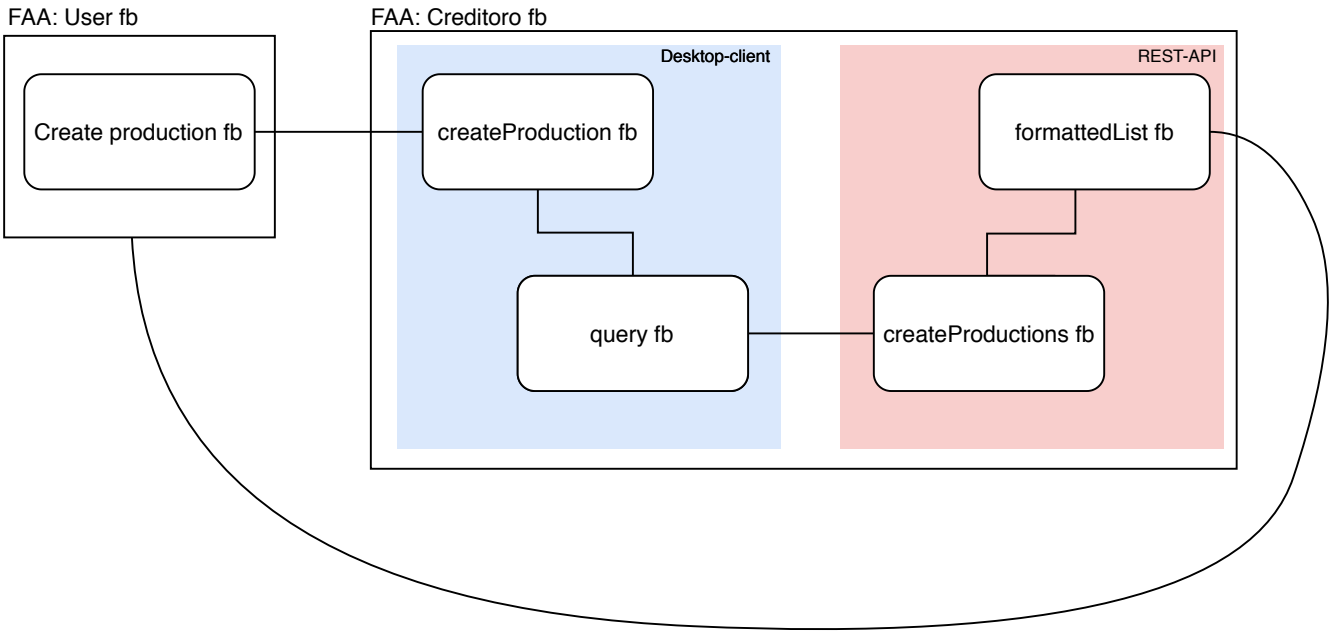
EPG Poller		
ID	**Name**	**Description**
NE01	Supportability	Centralised error reporting should be available via a common interface (such as Sentry).
NE02	Performance	It should be able to poll data every hour and finish within 15 min to update the data
NE03	Scalability	The system should be able to poll 15K new shows every year yearly, for 25 years.
NE04	configuration	How often the poller is run, where to poll data from and where to post data too should be configurable
NE05	Availability	The system should start automatically after server restarts.
NE06	Usability	The system should warn about show's without credit
NE07	Security	The server that the EPG poller is hosted on should only allow connections from the outside on port 22221(ssh).
NE08	Configurability	The EPG Poller should be configurable using **environment** files.

Tabel 39: EPG Poller: Non-functional Requirements

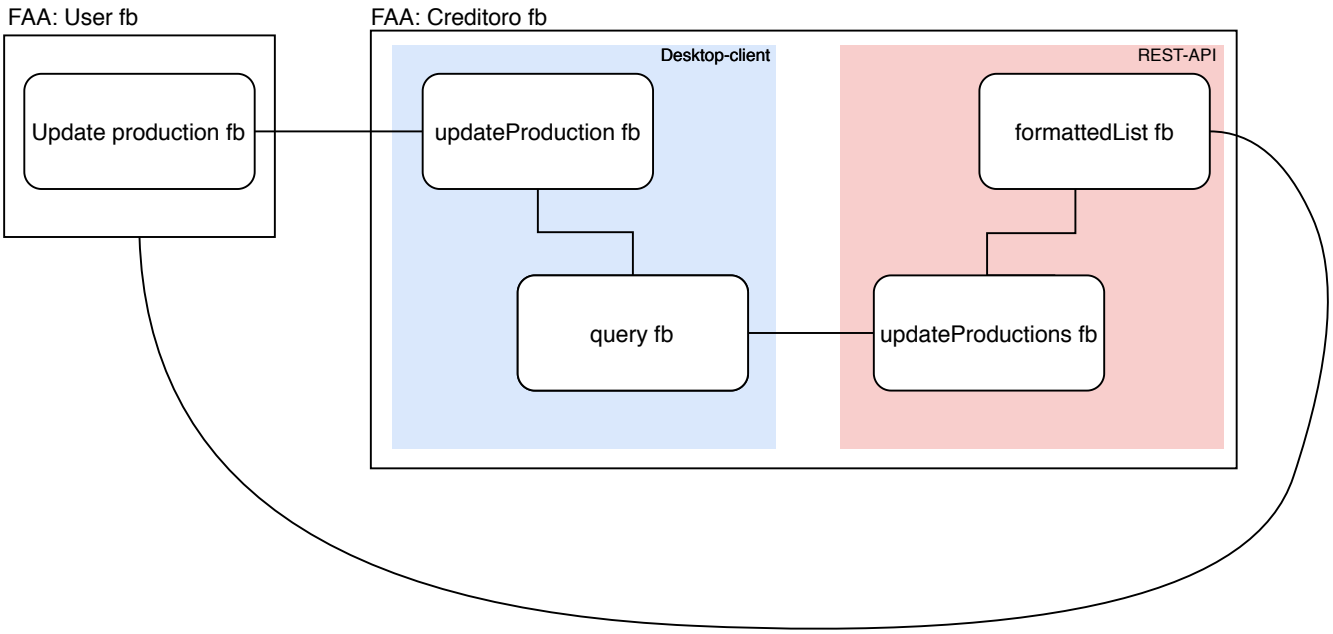
U EAST-ADL Modeller



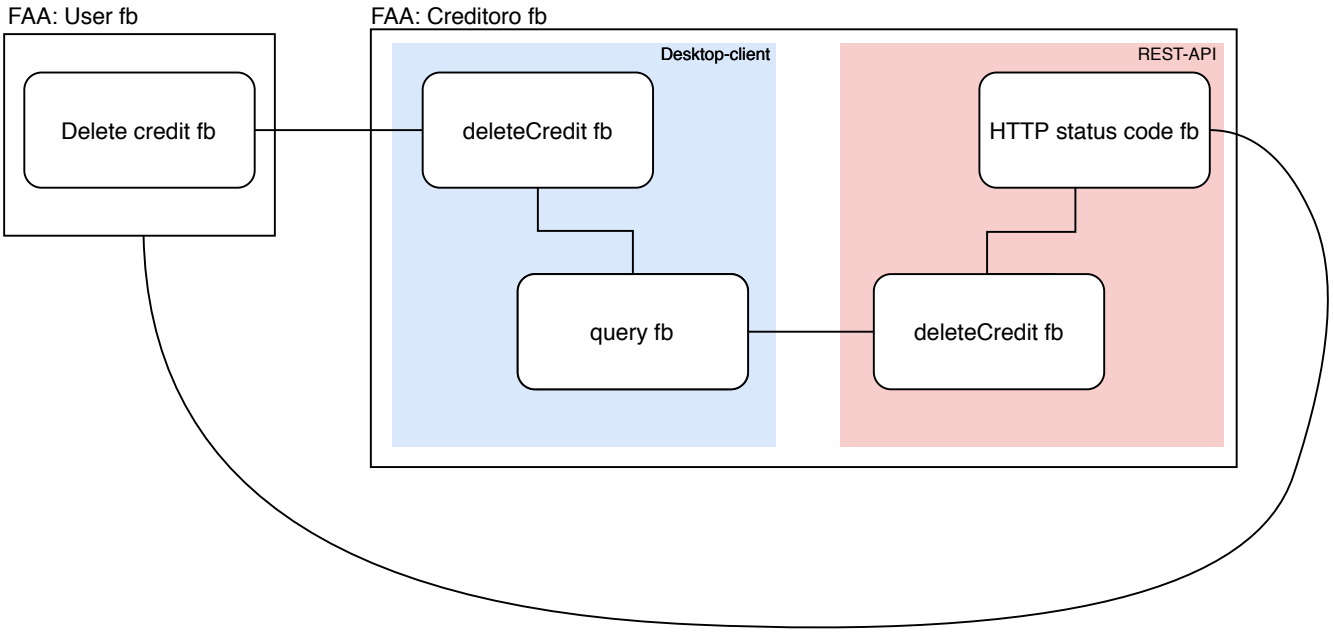
Figur 18: search



Figur 19: create



Figur 20: update



Figur 21: delete