Computer Vision for Video On Demand

Dataingeniør

Institutt for data- og realfag

Fakultet for ingeniør- og naturvitenskap (FIN)

Innleveringsdato: 03-06-2019

Antall ord: 10984

Magnus Ødegård Bergersen - 181182
Mikail Orlov Andreassen - 181200
Jacob Ås - 181185

We confirm that the submitted work is independently produced and that all references and sources are clearly stated according to *Forskrift om studier og eksamen ved Høgskulen på Vestlandet, § **9-1.***

Fakultet for ingeniør- og naturvitskap
Institutt for data- og realfag

TITTELSIDE FOR HOVEDPROSJEKT

Rapportens tittel: Computer Vision for Video On Demand	*Dato: 03-06-2019*
Forfatter(e): Magnus Ødegård Bergersen, Mikail Orlov Andreassen, Jacob Ås	*Antall sider u/vedlegg: 53*
	Antall sider vedlegg: 8
Studieretning: Dataingeniør	*Antall disketter/CD-er:* 0
Kontaktperson ved studieretning: Violet Ka I Pun	*Gradering: Ingen*
Merknader:	

Oppdragsgiver: Vimond Media Solutions AS	*Oppdragsgivers referanse:* Ingen
Oppdragsgivers kontaktperson: Rune Jordal	*Telefon: +47 94430737*

Sammendrag:

Denne rapporten beskriver vi prosjektet vårt - Computer Vision for Video on Demand, for bedriften Vimond Media Solutions. Prosjektet baserer seg på å lage en applikasjon som bruker maskinlærings APIer til å analysere videofiler og hente metadata fra dem. Produktet som vi har utviklet og beskriver i denne oppgaven, vil være en løsning som kan videreutvikles av Vimond Media Solutions.

Stikkord:

Maskinlæring	.NET Core	Amazon Web Services

Høgskulen på Vestlandet, Fakultet for ingeniør- og naturvitenskap
Postadresse: Postboks 7030, 5020 BERGEN Besøksadresse: Inndalsveien 28, Bergen
Tlf. 55 58 75 00 Fax 55 58 77 90 E-post: post@hvl.no Hjemmeside: http://www.hvl.no

Forord

I denne rapporten vil vi gå igjennom prosjektet vårt - Computer Vision for Video on Demand, for bedriften Vimond Media Solutions. Prosjektet baserer seg på å lage en applikasjon som bruker maskinlærings APIer til å analysere videofiler og hente metadata fra dem.

Prosjektet er gjennomført av Jacob Ås, Mikail Orlov Andreasen og Magnus Ødegård Bergersen.

Vi ønsker å takke Vimond Media Solutions som har gitt oss denne muligheten til å jobbe med et slikt spennende prosjekt, og har bidratt med råd og veiledning under hele arbeidstiden.

Vi ønsker også å takke vår veileder, Violet Ka I Pun som har også kommet med hjelp og veiledning under prosjektarbeidet.

INNHOLDSFORTEGNELSE

Kapittel 1 - INTRODUKSJON

I dette kapittelet skal vi introdusere prosjektet vi har jobbet med i bachelor tiden. Vi skal gå gjennom prosjektets mål og motivasjon, der vi tar for oss problemstillingen, mål og motivasjon for oppgaven. Vi skal sette oppgaven i kontekst og gjøre rede for hvilke begrensninger vi har gjennomføring av prosjektet.

1.1 Mål og motivasjon

Figur 1 - Logoen til Vimond

Vimond Media Solutions jobber med løsninger for medieselskaper på verdensbasis. De utvikler og jobber med OTT (Over-the-top) media løsninger for den nye digitale TV bransjen. Vimond ble grunnlagt i 2011 i Bergen, og jobber med store verdensledende TV selskaper som Comcast, Reuters, TV2, TV.AE, iflix og flere. De jobber tett med disse selskapene for å adoptere og vokse på den digitale fronten for den nye generasjon TV-seere.

Vimond har fortsatt sitt hovedkontor i Bergen, i Nøstegaten 72, men har også etablert kontorer i New York, Dubai og Sydney. Vimond har nå i dag omtrent 74 ansatte på verdensbasis, hvorav mesteparten befinner seg i Bergen.

1.1.2 Problemstilling

Når en bruker Vimonds streaming tjenster, må vedkommende skrive inn en rekke informasjon om videofilen slik at den kan bli kategorisert på rett plass. Dette gjøres for å gi brukeren muligheter til å kategorisere videofilen ved et senere tidspunkt. Denne prosessen kan kreve en del tid fra brukeren, ettersom det er store mengder med videofiler som vanligvis går igjennom en slik streamingtjeneste. Ettersom filmstreaming har gått over fra

å være nyhet til råvare i dagens samfunn, må Vimond være mer frampå med ny teknologi. Vimond har derfor valgt å se på mulighetene for bruk av maskinlæring for å se om det kan brukes til å hente informasjon (metadata) for brukeren. Problemstillingen er derfor å integrere maskinlæring til å hente metadata fra en pipeline av videofiler og integrere denne funksjonen i Vimond sine eksisterende tjenester.

1.1.3 Mål

Vårt nåværende mål er å bygge en selvstendig tjeneste som kan ekstrahere metadata fra video. Denne tjenesten skal senere kobles opp mot en AWS Step Function basert workflow. Andre delmål inkluderer integrasjon av tjenesten i Vimond sine eksisterende systemer og utvikling av et grafisk grensesnitt for applikasjonen som skal bli brukt til å vise den fram på Expo.

1.2 Kontekst

1.2.1 Prosjekt

Vimond har kommet fram til at bruk av maskinlæring i sine systemer ved utføring av monotone handlinger kan gjøre hverdagen lettere for både ansatte og kunder. Derfor har de valgt å undersøke mulighetene til å utvikle en slik tjeneste for henting av metadata fra en videokilde.

1.2.2 Gruppen som en del av bedriften

Vimond er delt opp i forskjellige avdelinger. Vi er del av Vimonds team som heter “Video-team”. Vi er på et vis uavhengig og selvstendig, i den forstand at de ikke er avhengige av vårt resultat for deres drift. Samtidig blir vi fortsatt invitert og høyt oppfordret til å delta på ukentlige stand-ups med Video-team, for å fortelle om hvordan vi ligger an på prosjektet.

1.3 Begrensninger

1.3.1 Kunnskap om teknologien

For at prosjektet skal være vellykket, kreves det at gruppen er godt oppdatert og opplært innenfor de ulike teknologiene vi skal bruke. Heldigvis er Vimond villig til å tilby opplæring og veiledning gjennom utviklingen av prosjektet om det skulle være nødvendig. Det finnes også mye dokumentasjon rundt alle rammeverkene vi skal bruke.

1.3.2 Produkttesting

Ettersom applikasjonen som skal utvikles skal være en selvstendig modul, betyr dette at vi må utføre vår egen produkttesting. Skal applikasjonen derimot benyttes av Vimond og integreres i deres systemer må applikasjonen følge deres kravspesifikasjoner slik at den feilfritt kan implementeres til deres systemer. Ved hjelp av XUnits enhetstesting kan vi lage automatiske enhetstester som vil sikre for at applikasjonen vår også er robust nok til å kunne bli tatt i bruk.

1.4 Ressurser

Oppgaven baserer seg på bruk av maskinlæring for å få et ønsket resultat. Disse algoritmene krever store mengder data for å være nyttige og nøyaktige, derfor trengs det en del rådata som videofiler. Ved store mengder videofiler klarer vi å finjustere maskinlæring modellene våre til å få et ønsket resultat som kan brukes til sluttproduktet. Heldigvis på grunn av Vimond driver med streamingtjenester, så vil de ha mange videofiler tilgjengelig for vårt bruk.

Det er også viktig å påpeke at vi skal ikke skriver våre egne maskinlæringsalgoritmer. Vi skal ta i bruk to av Amazon sine webtjenester (AWS) kalt Amazon Rekognition og Amazon Transcribe. Det er et API utviklet av Amazon som gir oss tilgang til robuste og effektive algoritmer for analyse av video.

Under arbeidet kommer vi til å forholde oss til et agilt prosjektrammeverk. Under bruk av dette, kommer vi til å bruke diverse tekniske verktøy for planlegging av sprints og sporing av milepæler. Vi skal også forholde oss til en Test Driven Development teknikk for å opprettholde kvalitet og konsistens på koden.

Som nevnt i kapittel 1.3.1, så har vi mulighet å få assistanse under arbeidet ved hjelp av Vimond sine utviklere. Dette vil gjelde veiledning i forhold til prosjektmetodikken og tekniske spørsmål. For å gjøre det mest mulig enkelt for dem, vil vi benytte lignende tekniske verktøy som utviklere hos Vimond, det vil si Jetbrains IDE og C# som programmeringsspråk. For versjonskontroll, kommer vi til å bruke Git + Github. Vi vil også kunne delta på ukentlige statusmøter hvor vi kan presentere fremgangen av prosjektet og få kommentarer og forslag fra møtedeltakerne.

1.5 Organisering av rapporten

Rapporten er delt opp i ni kapitler som skal forklare bachelorprosjektets gjennomgang;

Kapittel 2 - Går igjennom det mer praktiske med prosjektet. Her får man bedre oversikt over hvordan prosjektet skal utføres ut ifra Vimonds rammeverk.

Kapittel 3 - Går igjennom hvordan prosjektet skal utføres i mer detaljer. Her vil vi gå igjennom utviklingsverktøy og programmeringsspråk og eventuelle løsninger som kan brukes om ting skulle gå galt.

Kapittel 4 - Går igjennom hvilken produktdesign vi har brukt for å komme fram til resultatet og hvordan det tekniske er satt opp i prosjektet.

Kapittel 5 - Går igjennom evaluering av prosjektet. Dette innebærer både sluttresultatet og framgangen frem til sluttresultatet.

Kapittel 6 - Går igjennom konsekvenser for valg gjennom prosjektet og hvilken påvirkning det hadde på prosjektet.

Kapittel 7 - Går igjennom konklusjon fra hele oppgaven og hva som kunne gjøres bedre om det skulle blitt mer tid satt av til oppgaven.

Kapittel 8 - Inneholder referansene som ble brukt til å lage rapporten.

Kapittel 9 - Appendix som inneholder eksterne materialer og modeller til rapporten, blant annet vårt GANT-diagram og risikomatrise.

Kapittel 2 - BAKGRUNN

2.1 Praktisk bakgrunn

2.1.1 Prosjekt-eier

Vimond Media Solutions ble stiftet i 2011. Selskapet driver mest med utvikling og markedsføring av skreddersydde applikasjoner for TV bransjen. Vimond har hovedkontor i Bergen, men har også internasjonale kontor i New York, Dubai og Sydney. Kundene til Vimond er hovedsakelig TV selskaper som Comcast, Thomson Reuters og TV2.

Under utvikling av prosjektet, vil Vimond ha alle rettighetene knyttet til løsningen. Siden løsningen vil være integrert i deres eksisterende systemer, vil det være naturlig at de vil få eierskap av prosjektet. Vi vil å ha mulighet til å presentere løsningen offentlig, samt omtale det i rapporten.

2.1.2 Tidligere arbeid

Vimond har over tid utviklet en stor variasjon av video og streaming baserte tjenester. Selv om løsningen vår vil starte som en uavhengig applikasjon, vil det være veldig aktuelt å integrere det i noen av disse tjenestene. En av de mest aktuelle ville vært en pipeline basert applikasjon kalt Orchestration. Orchestration er en sett av mikrotjenester som tar inn videofiler og transkoder disse, slik at de er klar til streaming. Løsningen vår vil kunne bli brukt i dette systemet til å hente metadata fra strømmen av videofiler og behandle den dataen sømløst.

Vår produktdesign vil basere seg på å utvikle en API som vil kunne sømløst plugges inn AWS Step Functions basert arbeidsflyt. Dette vil tillate Vimond å ta i bruk tjenesten vår uten å måtte bruke ekstra ressurser på å tilpasse det til sine egne systemer.

2.1.3 Innledende kravspesifikasjon

Vimond har stilt oss en mengde krav som skal følges under utviklingen av prosjektet. Disse kravene omhandler både design, metoder og utviklingsverktøy, og er essensielle til oppnåelse av ønsket sluttresultat.

2.1.3.1 Design og tekniske spesifikasjoner

Applikasjonen vi utvikler må være en uavhengig tjeneste som vil på et senere tidspunkt kunne bli integrert i AWS Step Functions basert arbeidsflyt. Tjenesten skal ta i bruk AWS Rekognition og AWS Transcribe APIer til å ekstrahere informasjon fra video filer og lagre det i en database (DynamoDB eller Elasticsearch). Informasjonen vi skal hente vil være:

- Gjenstand, scene og aktivitetsinformasjon
- Ansiktsgjenkjenning
- Tale-til-tekst behandling

For å gjøre det lettest mulig for Vimond å ta i bruk applikasjonen senere vil vi skrive i C#. Flere av Vimond sine applikasjoner og systemer er allerede gjort i C# og da spesielt tilknyttet til AWS. Vi prøver å forholde oss til Vimond sine kodestandarder. Det vil si at all kode vi skriver skal være veldefinert og lett å tolke.

2.1.3.2 Utviklingsmetodikk

Vimond anbefalte oss å benytte samme utviklingsmetodikk som de selv benytter, det vil si et agilt rammeverk Scrum. Scrum er et prosessrammeverk som brukes til å styre produktutvikling og gir arbeidsgruppen et verktøy for å holde kontroll på oppgaver også kalt sprinter. Arbeidsgruppen kan sette opp en hypotese om hvordan de tror noe fungerer, teste det ut, reflektere, og deretter gjøre nødvendige justeringer om nødvendig [21].

For å gjøre arbeidet med Scrum lettere fikk vi også en anbefaling å ta i bruk noen tekniske prosjektverktøy (i vårt tilfelle Trello, kapittel 3.3) som vil tillate oss å ha kontroll på sprintene, backloggen og milepælene.

Vi skal også prøve å holde en Test Driven Development metodikk under utvikling, det vil si å skrive enhetstester til metodene før vi skriver funksjonell kode [28].

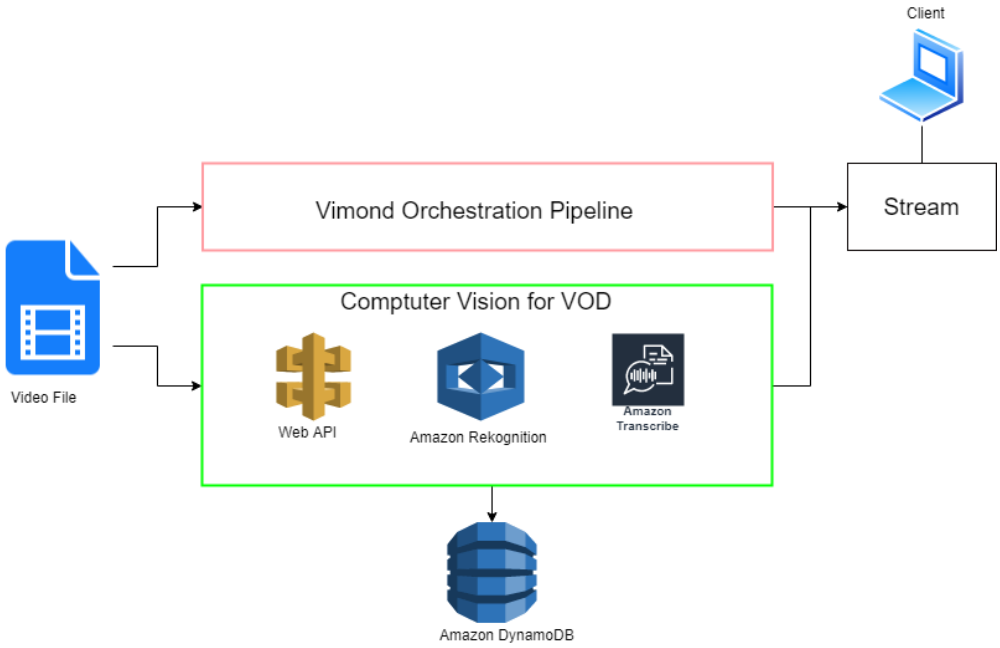
2.1.4 Innledende løsningsidé

Ideen til en potensiell løsning på problemstillingen er gitt til oss av bedriften. Den bygger seg på at Vimond ønsker å forenkle en prosess i en av deres systemer ved å ta i bruk ny teknologi.

2.1.4.1 Funksjonalitet

Som nevnt i kapittel 1.1.4, er behandling av metadata på Vimond sine tjenester hovedsakelig et menneske styrt prosess. Akkurat nå så må en bruker selv analysere video og skrive inn informasjon om filmen.

Løsningen vår vil optimalisere denne prosessen ved å implementere en maskinlæring API som vil hjelpe med å ekstrahere metadata fra filene som blir behandlet i Vimond sine tjenester. Vi tenkte å implementere dette på måten som vises i figuren:



Figur 2 - Illustrasjon av funksjonaliteten i prosjektet

Når en videofil kommer inn til Orchestration pipeline som er beskrevet i kapittelet 2.1.2, vil denne filen bli sømløst behandlet av tjenesten vår som er vist under Orchestration på figuren. Det vil si at filen blir behandlet samtidig av både Orchestration (rosa på figuren) og vår tjeneste (grønn på figuren).

Applikasjonen vi kommer til å utvikle kommer til å ta i bruk AWS Rekognition og Transcribe APIer. Disse vil kommunisere med hverandre for å hente nødvendig informasjon fra en videofil. Informasjonen som er hentet fra filen skal så lagres i en DynamoDB database, og vil kunne aksesseres med en tilpasset Web API som vi også kommer til å utvikle.

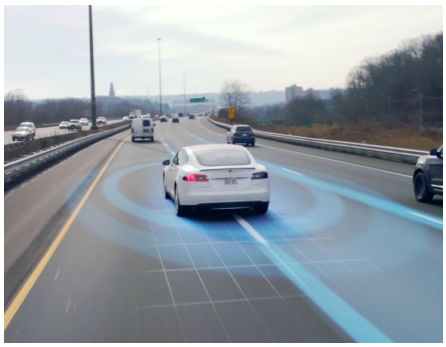
I tillegg vil vi også få en mulighet til å utvikle et enkelt grafisk grensesnitt for funksjonen, selv om det er ikke er et av kravene vil det være gunstig for oss, spesielt når vi skal vise prosjektet på presentasjoner og EXPO.

2.2 Litterær bakgrunn

Maskinlæring har blitt et stort tema de siste årene for alle selskaper. Aviser og nyhetsartikler skriver mye om maskinlæring og hvordan det vil være relevant både for oppgaver og for utvikling av AI. Per Kristian Bjørkeng skriver i Aftenposten "*Mener alle må ha disse to kursene i fremtiden*" at maskinlæring at det er et emne som vil komme til å påvirke arbeidshverdagen i absolutt alle bransjer [1]. Dette er for 3 år siden og vi kan se at flere og flere bedrifter har rettet seg mot dette.

IT selskapet Sopra Steria har begynt å tilby spesifikke tjenester innenfor implementering av maskinlæring hvor dem reklamerer for at dem skal hjelpe med monotone arbeidsoppgaver for at hverdagen blir så enkel som mulig [2].

Vi ser også at konsulentselskapet Bouvet tilbyr kunnskap og tjenester innenfor kunstig intelligens og maskinlæring for å bidra til økt lønnsomhet i virksomheten, blant annet ved å utføre rutineoppgaver slik at de ansatte kan utnytte kjernekompetansen sin bedre [3]. Begge disse selskapene har i løpet av et år etter artikkelen kom ut opprettet tjenester spesifikt for maskinlæring.



Figur 3 - Illustrasjon av Tesla Autopilot

Men kanskje hva mesteparten er overrasket er hva maskinlæring har gjort med videotjenester de siste årene. I 2017 ga Tesla ut det populære automatiske kjøreassistanse systemet som revolusjonerte maskinlæring for biler [4]. Med sine 12 kameraer klarer Tesla biler å gi en 360 graders oversikt med opptil 250 meters rekkevidde som kan automatisk styre bilen gjennom korte og lange kjøreturer [5].

Et annet eksempel er OpenAI Five. OpenAI Five er en selvlærende maskinlæringsalgoritme som startet å spille videospillet Dota 2 uten noen kjennskap til spillet [6]. Den klarte å revolusjonere maskinlæring innenfor videospill den 11. august 2017, ved å slå en av verdens beste spillere i den årlige konkurransen The International fra Valve Studios foran flere millioner seere [7].

Figur 4 - Logoen til OpenAI

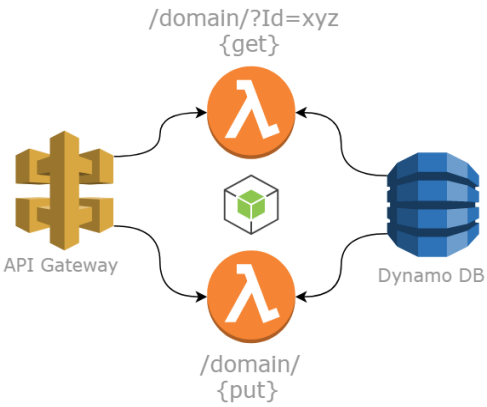
Kapittel 3 - PROSJEKTDESIGN

3.1 Mulige løsninger

3.1.1 Alternativ 1 - AWS applikasjon med .NET Core

Lage en backend applikasjon som ikke har noe UX-design. Dette vil tillate oss å gjøre alt med objektorientert programmering uten å bruke tid på å sette opp web-API som en nettside må ha. Applikasjonen vil fortsatt være satt opp ved hjelp av .NET Core sitt rammeverk for å skrive applikasjonen i C#. Ved å importere pakkene fra AWS sine nettsider og lage en applikasjon rundt dette vil vi få samme funksjonalitet som en applikasjon med grafisk grensesnitt på en nettside. Det negative med denne løsningen, er at det blir vanskelig å ha noe form for visuell funksjonalitet uten å lage et dedikert grafisk grensesnitt. Mye av arbeidet blir å foregå "bak kulissene" så det blir veldig lite visuell tilbakemelding fra applikasjonen.

3.1.2 Alternativ 2 - AWS applikasjon med Node.js

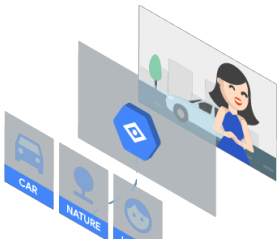


Figur 5 - Illustrasjon av AWS med Node.js

AWS SDK har også åpne muligheter for å lage applikasjoner med JavaScript. På denne måten kan man bruke JavaScript objekter for å yte AWS sine hjelpemidler for å få samme funksjonalitet som når man lager applikasjonen med .NET rammeverket. På denne måten vil man automatisk bruke RESTful API for å få objektene til å kommunisere med hverandre. Derfor vil man ha mer frihet når man lager en nettside ut fra koden ettersom alle responser og forespørsler vil skje direkte fra en nettside uten å gå igjennom en ekstern .NET Core server.

3.1.3 Alternativ 3 - Cloud Vision applikasjon med .NET Core

Google Cloud tilbyr også en API for å analysere filmer, nemlig Cloud Vision.[8] En mulighet er å bruke APIen til Google Cloud for å lage en applikasjon med samme funksjonalitet som AWS. Istedenfor å bruke Amazon sin database til å lagre metadataen vil Google Cloud tilby sin egne database alternativer for lagring av data. Applikasjonen



Figur 6 - Illustrasjon av Cloud Vision

vil fortsatt skrives i .NET Core rammeverket ved hjelp av programmeringsspråket C#.

3.1.4 Diskusjon om alternativene

Om det skulle oppstå noen tidsproblemer med prosjektet vil alternativ 1 være det mest realistiske alternativet å velge. Med å se bort fra et grafisk grensesnitt på nettet vil vi få mer tid til å utvikle en applikasjon med mer vekt på maskinlærings APIet til AWS.

Om vi bestemmer oss for å putte mer vekt på å lage en nettside igjennom utviklingsperioden vil alternativ 2 med Node.js være bedre å bruke. Ved å bare fokusere på HTML, CSS og JavaScript vil vi ha bedre kontroll på hvordan nettsiden skal fungere. Spesielt ettersom flere medlemmer i gruppen har erfaringer med dette fra tidligere praksisperioder og fag. Om man skal gå for dette alternativet må det skje tidlig i utviklingsprosessen ettersom det vil forandre på hele strukturen til prosjektet.

Om AWS skulle på et tidspunkt skape store tekniske problemer vil man alltid ha muligheter til å flytte over til alternativ 3 med Google Cloud sin alternativ til video tolkning. Men ved dette valget må gruppen tidlig skifte på hele modelleringen til å fungere med Cloud Vision sitt API. Men positivt med dette er at man vil da få bedre scene-gjenkjenning resultater ettersom Google Cloud er kjent for å gi bedre resultater på dette.

3.2 Spesifikasjon / valgt prosjektdesign

Designet vi har valgt å gå frem med innebærer bruk av teknologiene som bedriften er kjent med og benytter selv. Dette gjøres for å gjøre implementeringen av produktet lettere for Vimond i fremtiden. Hadde vi valgt å bruke Javascript løsningen ville applikasjonen vært vanskeligere å integrere i Vimonds Pipeline. I tillegg ville valg av Google Cloud ha skapt problemer ettersom Vimond allerede bruker AWS sine tjenester til sitt daglige arbeid. Applikasjonen vår vil derfor bli skrevet i .NET Core med kombinasjon av verktøy og APIer som AWS tilbyr etter anbefaling fra bedriften. Vi fikk også en mulighet for å designe et grafisk grensesnitt som kan vises på EXPO hvis vi får nok tid til dette. Verktøyene som vi kommer til å bruke er beskrevet i neste kapittel.

3.3 Valg av verktøy og programmeringsspråk

Rider

Rider er en IDE fra JetBrains for .NET/.NET Core-utvikling [9]. Den har en god kompilator for kompilering av kode og gode feilsøking-funksjoner også kalt debugging. Det er enkelt å refaktorere C# kode og Rider har god git-integrering.

Figur 7 - Logoen til Rider

AWS Rekognition og AWS Transcribe

Maskinlæring tjenester tilbudt av Amazon. AWS Rekognition gjør det enkelt å jobbe med bilde og video analyse til vår applikasjon. AWS Transcribe er en automatisk tale gjenkjennings tjeneste som gjør det enkelt å implementere "tale-til-tekst" til vår applikasjon.

Figur 8 - Logoen til AWS

Swagger/OpenAPI

OpenAPI er en API-beskrivelse format for REST API-er [10]. Swagger er sett med "open source" verktøy bygd rundt OpenAPI som gir hjelp når man skal designe, bygge, dokumentere og bruker REST API.

Figur 9 - Logoen til Swagger

.NET Core & C-sharp

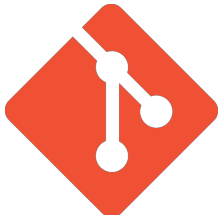
Kompilator og programmeringsspråk. .NET Core er en "open-source" utviklingsplattform laget av Microsoft [11]. .NET Core er cross-platform som gir støtte for flere operativsystemer.

C-sharp (C#) er programmeringsspråket vi skal benytte for å lage applikasjonen.

Figur 10 - Logoen til .NET Core

Git

Git er en versjonskontroll verktøy som blir brukt for kodedeling.[12]. Det er et verktøy som gir utviklere mulighet til å jobbe parallelt på samme prosjekt. Dette gjøres ofte med å dele opp koden i og jobbe i unike "branches". Git er nyttig for et prosjekt i den forstand at man gjerne ønsker å jobbe agilt og



Figur 11 - Logoen til Git

fleksibelt på koden. Du kan jobbe selvstendig helt uavhengig av andre, og senere kan du dele arbeidet du har gjort med resten av teamet.

Trello

Trello er et verktøy for å spore arbeidsoppgaver i det som blir kalt "issues" [13]. Trello er brukt for å spore problemer i programvare og prosjektoppgaver. Trello er fleksibelt og er lett å konfigurere. Det gjør at en prosjektleder kan lage flere typer arbeidsoppgaver. Alle oppgavene har forhåndsbestemte data og informasjon, som hvilke egenskaper og attributter som er tilhørende. Disse kan spores etter status, prioritet og versjon.

Figur 12 - Logoen til Trello

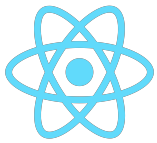
Slack

Slack er et kommunikasjonsverktøy for bedrifter [14]. Benyttes for kommunikasjon internt i Vimond. Her holder vi kontakt med Rune og tilhørende assistanse dersom de ikke er tilgjengelig i person.

Figur 13 - Logoen til Slack

React.js

React.js er et komponentbasert Javascript-rammeverk for bygging av brukerens grensesnitt for web-utvikling [15].



Figur 14 - Logoen til React.js

Node.js

Node.js er et Javascript miljø som lar datamaskinen kjøre kode uten nettleser [16].

Figur 15 - Logoen til Node.js

3.4 Prosjektets utviklingsmetode

3.4.1 Utviklingsmetode

Vi har benyttet oss av Scrum som utviklingsmetode, beskrevet i kapittel 2.1.3.2.

3.4.2 Prosjektplan

Vi har laget et GANTT diagram (Appendix 10.2) [24]. Vi har delt opp diagrammet i fire deler. Analyse, modellering, utvikling og EXPO. Her er forklaring av GANT-diagrammet.

Analyse: Det punktet handler om å analysere og forberede hva som er nødvendig for å gjennomføre. Her skal vi bli kjent med prosjektet. Vi utforsker nødvendige utviklingsverktøy, analysemetoder og tilhørende dokumentasjon. I tillegg er den obligatoriske M2 satt opp her.

Modellering: Her handler det om å forme en mulig struktur av prosjektet. Vi former arkitektur til prosjektet. Diskuterer og analyserer mulige løsninger for prosjektet. Vi dokumenterer dette i innleveringene og forbereder oss til muntlig presentasjoner.

Utvikling: I denne fasen skal vi finne endelig arkitektur og en endelig løsning. Deretter skal vi begynne å utvikle og om alt går etter planen vil produktet til slutt være ferdig. Utover hele perioden vil vi jobbe med bachelor rapporten og innhold til bloggen. Vi lever til slutt rapporten og refleksjonsnotat. Deretter begynner vi å gjøre oss klare til neste fase.

EXPO: Vi gjør alle nødvendige forberedelser til EXPO. Innebærer en presentasjon og hvilket innhold vi skal vise og hold på stand.

3.4.3 Risikohåndtering

Det er brukt en risikoanalyse (Appendix 10.1) for å avdekke hvilke risikoer prosjektet kan støtte på.

3.5 Evalueringsmetode

For å kunne evaluere resultatet av arbeidet har vi bestemt oss for å bruke en del evalueringsmetoder som vil tillate oss å vurdere hvor godt problemstillingen ble besvart. Disse metodene er beskrevet i kapitelene under.

3.5.1 Evaluering fra Scrum

Under utviklingsprosessen kommer vi til å benytte oss av Scrum. Denne utviklingsmetoden tillater oss å få kontinuerlig tilbakemeldinger under utvikling i forhold til problemstillingen. Vi kan så bruke disse til å evaluere arbeidet.

3.5.2 Kommunikasjon med veileder og bedriften

Kommunikasjon mellom oss og veilederen blir viktig for oss for å få kontinuerlig oppfølging både til dokumentasjon og utvikling. Vi vil kunne gjennomføre dette hovedsakelig gjennom møter hvor vi kan ta opp aktuelle saker med relevans til fremgang av prosjektet. Vi vil så kunne justere vår arbeidsflyt etter anbefalinger fra veilederen

Det samme vil gjelde vår kontaktperson i bedriften, samt ansatte som vil bidra med tilbakemeldinger hovedsakelig på møter og ukentlige standups.

3.5.3 Intervju

Mot slutten av prosjektarbeidet skal vi gjennomføre et intervju med produktsjefen. Denne vil gi oss en mulighet å få en konkret tilbakemelding om hvor bra problemstillingen ble besvart og hvilke krav som ble fulgt under utviklingen.

Under intervjuet skal disse spørsmålene besvares:

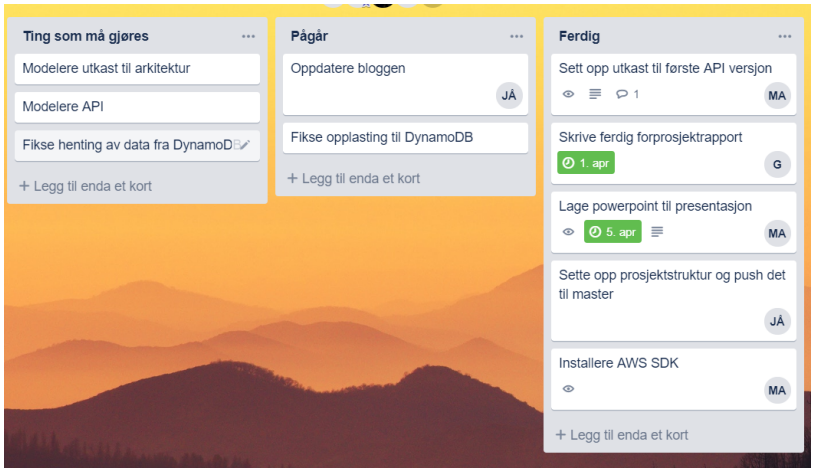
- Forventninger og resultat
- Prosessen om integrering i eksisterende systemer
- Nytteverdi av produktet
- Mulige endringer eller ekstra funksjonalitet

Kapittel 4 - PRODUKTDESIGN

I dette kapittelet vil vi beskrive hvordan produktet er designet. Vi vil gjennomgå planleggingsprosessen, arkitektur av diverse AWS komponenter og forklare nærmere hvordan applikasjonen er strukturert.

4.1 Planlegging og innledende design

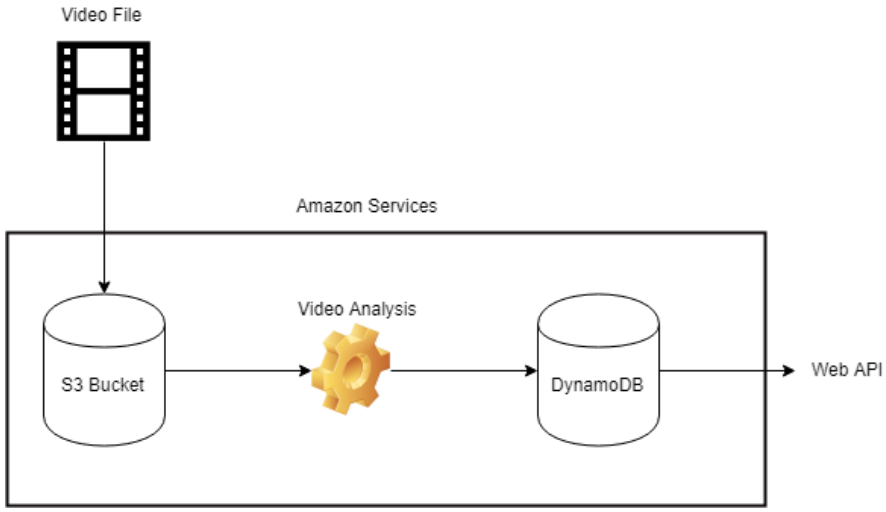
I begynnelsen av prosjektarbeidet fikk vi en konkret kravspesifikasjon (se kapittel 2.1.3) utviklet av Vimond. Denne beskrev både funksjonalitet og design av applikasjonen. Selv om disse kriteriene var fast definert, kunne metodikken vi brukte til å løse disse ha mer frihet. Vi bestemte da å følge et Scrum lignende tilnærming med bruk av tekniske hjelpeverktøy for å holde oversikt over oppgavene som skulle løses.



Figur 16 - Bilde fra Trello veggen vår

Fokuset i planleggingsfasen var på å få oversikt over hvilke teknologier vi skulle bruke og hvordan vi skulle strukturere applikasjonen i forhold til disse teknologiene. Mesteparten av AWS komponentene måtte ha spesifikke innstillinger når det kom til tilgang og bruk, så disse måtte fikses i første fase før vi kunne begynne på implementasjon av arkitekturen.

Etter arbeidet med å sette opp komponenter, gikk vi over til å designe en enkel, innledende oversikt over prosessen i den følgende figuren:



Figur 17 - Innledende designsketch

En videofil skulle først lastes opp til en Amazon S3 Bucket. Videre skulle den behandles av Rekognition og Transcribe algoritmene som skulle hente nødvendig data fra filen. Denne dataen skal da lagres i en DynamoDB database og vil kunne aksesseres ved bruk av en tilpasset Web API.

4.2 AWS Arkitektur

4.2.1 Oversikt over komponenter

Applikasjonen vi har utviklet benytter seg av en variasjon av AWS komponenter. Disse komponentene er:

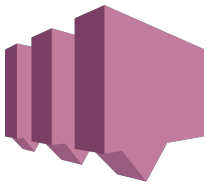
S3 Buckets

Amazon S3 er en enkel web interface som tillater brukere å lagre og hente data. Denne tjenesten gir utviklere en skalerbar og effektiv lagringstjeneste som er i tillegg utviklet for å jobbe sammen med mange andre AWS-komponenter. Applikasjonen vår benytter seg av S3 til å lagre videofilene som skal bli behandlet.

Figur 18 - Logoen til S3 Bucket

SNS (Simple Notification Service)

SNS er en sikker og fullt administrert meldingstjeneste som er hovedsakelig brukt for kommunikasjon mellom andre AWS-komponenter. Meldingene



Figur 19 - Logoen til SNS

som blir sendt gjennom SNS kan for eksempel bli brukt til å starte funksjonskall eller legge ut status på pågående operasjoner.

SQS (Simple Queue Service)

Figur 20 - Logoen til SQS

Simple Queue Service er en tjeneste som mottar og behandler innkommende meldinger sendt vanligvis gjennom SNS. Andre tjenester kan da få tilgang til køen og bruke disse meldingene selv.

Rekognition

Figur 21 - Logoen til Rekognition

AWS Rekognition er en tjeneste som bruker kraftige maskinlæring algoritmer utviklet av Amazon til å hente informasjon fra både bilde og videofiler [17]. Rekognition har en API som gjør det enkelt for utviklere å bruke det i kombinasjon med andre løsninger, og er derfor sentralt for oss.

Rekognition i vår løsning brukes til å hente to typer data:

- Objekt og scene informasjon i form av "labels"
- Ansiktsgjenkjenning

Transcribe

Figur 22 - Logoen til AWS ML

AWS Transcribe er en tjeneste som gjennomfører automatisk talegjenkjenning og er utviklet av Amazon. Den brukes for å hente "Tale til tekst". Ved bruk av Transcribe API kan man analysere lyd fra video filer og API vil returnere et JSON objekt av den transkriberte teksten [18].

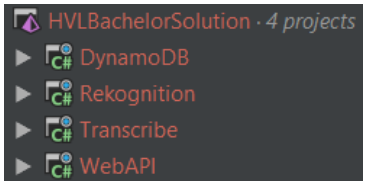
DynamoDB

DynamoDB er en serverless, robust NoSQL database som gjør det enkelt for oss å lagre og hente data som vi får fra videofilene. DynamoDB tabeller kan lagre og hente store mengder data under hvilken som helst mengde trafikk. De er skalerbare og gjør det enkelt å gjøre forandringer i databasen [19].

Figur 23 - Logoen til DynamoDB

4.2.2 Kodearkitektur

Applikasjonen vi har utviklet består av en .NET Core (C#) Solution fordelt i fire funksjonelle prosjekter hvor hver av dem håndterer en bestemt funksjon. Disse fire prosjektene kan også skilles opp i fire separate moduler som kan kjøres uavhengig, noe som øker gjenbrukbarheten til prosjektet (Se UML diagram i appendix 10.3).

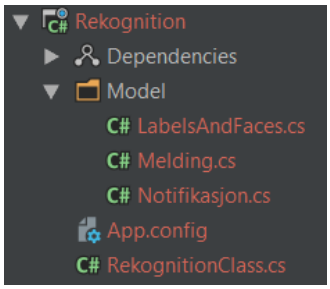


Figur 24 - Struktur av applikasjonen

Hvert prosjekt består av to typer klasser; funksjonelle klasser som utfører metodekall og forespørsler, og modell klasser som håndterer databehandling. På denne måten øker vi robustheten til programmet ved at prosjektet kan oppgraderes og forandres i framtiden uten store endringer. Man får også større overblikk over hvordan koden fungerer og hvordan modellene som brukes.

4.2.3 Teknisk funksjonalitet

Rekognition:

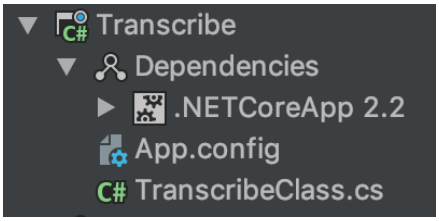


Figur 25 - Struktur av Rekognition prosjektet

I dette prosjektet har vi en klasse RekognitionClass som inneholder alle funksjonelle metoder relatert til videoanalysen. Data som SNS meldinger og resultatet sender blir lagret i klassen LabelsAndFaces i Model som beskrevet i 4.2.4.

Melding- og notifikasjonsklassen er en hjelpeklasse vi bruker for å konvertere JSON responsen vi får fra SQS til .NET objekter som kan brukes i prosjektet.

Transcribe:



Figur 26 – Struktur av Transcribe prosjektet

I dette prosjektet har vi en klasse TranscribeClass som inneholder funksjonalitet og metodene for å gjennomføre "tale til tekst" av samme videofil. Selve klassen returner en streng som inneholder JSON objektet som beskrevet i 4.2.3. Ettersom responsen fra AWS Transcribe er et JSON objekt så har vi muligheten til å direkte bruke denne uten noen konvertering. Om man skulle ha modifisert responsen måtte man laget hjelpeklasser som vi gjør med Rekognition.

DynamoDB:

DynamoDB prosjektet håndterer database operasjoner. Den består av to klasser: DynamoDBClass og JsonHandler.

I JsonHandler har vi en hjelpemetode som heter "jsonHandlerMethod".

```
public String jsonHandlerMethod(LabelsAndFaces rekog, String trans, String video)
{
    var transcribe = JsonConvert.DeserializeObject(trans);

    String json = JsonConvert.SerializeObject( value: new
    {
        video,
        rekog.faces,
        rekog.videoMetadata,
        rekog.labels,
        transcribe
    });
    return json;
}
```

Figur 27 - Hjelpemetoden "jsonHandlerMethod" i jsonHandler klassen.

Metoden er en viktig del av funksjonen til dette prosjektet. Den tar inn tre parametere. Listen med resultatene fra Rekognition, en streng som inneholder resultatet fra Transcribe og en streng som inneholder navnet på videofilen. Metoden samler objektene fra operasjonene fra begge analysene. Merk at strengen som er returnert fra Transcribe allerede er et "serialisert" objekt og må bli "deserialisert" før man setter de sammen med Rekognition og "serialisere" til et stort objekt som metoden returnerer. Metoden returnere da en streng som da blir det ferdig serialiserte objektet som DynamoDB støtter opp under, og klassen DynamoDBClass bruker denne strengen.

I DynamoDBClass starter man ved å initialiserer databasen. Hovedfunksjonen til denne klassen er å utføre en PUT operasjonen til DynamoDB tabellen vi opprettet, dypere detalj i kapittel 4.3.

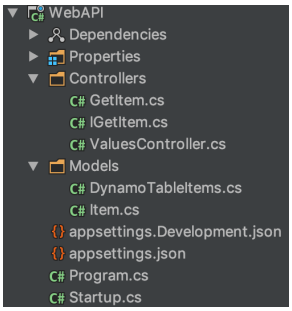
Metoden putItem lager en streng fra hjelpemetoden "jsonHandlerMethod", vi definerer hvilken tabell i DynamoDB vi skal utføre denne operasjonen mot, og deretter utfører den metoden "putItem" i DynamoDBClass klassen for å legge til informasjonen i databasen.

```
public void putItem(LabelsAndFaces labelsAndFaces, String trans, String video)
{
    String json = js.jsonHandlerMethod(labelsAndFaces, trans, video);
    Table table = Table.LoadTable(db, tableName);

    Document item = Document.FromJson(json.Trim());
    var response = table.PutItemAsync(item).GetAwaiter().GetResult();

    Console.WriteLine(response.Count.ToString());

}
```

Figur 28 - Metoden putItem i DynamoDBClass klassen

WebAPI:

WebAPI i .NET Core er strukturert på litt annen måte enn de andre prosjektene vi jobber med i produktet vårt som har vært konsoll applikasjoner/klasse biblioteker. Vi bruker den forhåndsdefinerte standarden til ASP.NET for å generere en RESTful applikasjon som kan brukes asynkront av flere enheter samtidig. APIet blir så koblet opp imot de 3 andre klassene i prosjektet ut ifra HTTP metodekallene.



Figur 29 - Struktur av WebAPI prosjektet

En RESTful applikasjon baserer seg på REST (Representational State Transfer) og er en rekke begrensninger for design av programvarearkitektur som gir effektive, pålitelige og skalerbare distribuerte systemer [22]. Et API blir da kalt RESTful når den oppfyller disse kravene [23]. REST sin arkitektoniske stil og begrensninger er basert på HTTP. HTTP er standard protokollen bak internett [24]. Noe som vi benytter i vår applikasjon.

Vi tar også i bruk Asp.Net MVC for å hjelpe oss sette opp arkitekturen til APIet vårt [26]. Men istedenfor den tradisjonelle MVC arkitekturen så bruker vi den heller på en måte hvor vi bruker modeller og innholdskontroller for å få et JSON resultat istedenfor et HTML resultat.

4.2.4 Struktur og funksjonalitet mellom AWS komponentene og Rekognition API

Prosessen av å hente data fra en lagret videofil utføres i 3 steg:

4.2.4.1 Starte video analysen med en startLabelDetection forespørsel (punkt 1 figur 32)

startLabelDetection forespørsel tar inn et JSON objekt som inneholder informasjon om videofilen, samt spesifikasjoner for analysen.

```
{
    "Video": {
        "S3Object": {
            "Bucket": "bucket",
            "Name": "video.mp4"
        }
    },
    "ClientRequestToken": "LabelDetectionToken",
    "MinConfidence": 50,
    "NotificationChannel": {
        "SNSTopicArn": "arn:aws:sns:us-east-1:nnnnnnnnnn:topic",
        "RoleArn": "arn:aws:iam::nnnnnnnnnn:role/roleopic"
    },
    "JobTag": "DetectingLabels"
}
```

Figur 30 - Eksempel på en StartLabelDetection request

S3Object refererer til videofilen som ligger på S3 Bucket. Videre oppgis SNS kanalen som skal motta meldingen om statusen på analysen. I responsen til denne forespørselen, mottas en JobId objekt som inneholder en unik ID som tilhører analyse operasjonen.

4.2.4.2 Motta status på analysen (punkt 2 og 3 figur 32)

Rekognition API sender en status melding til den registrerte SNS kanalen. Denne meldingen inneholder informasjon om operasjonen som dens Id og status.

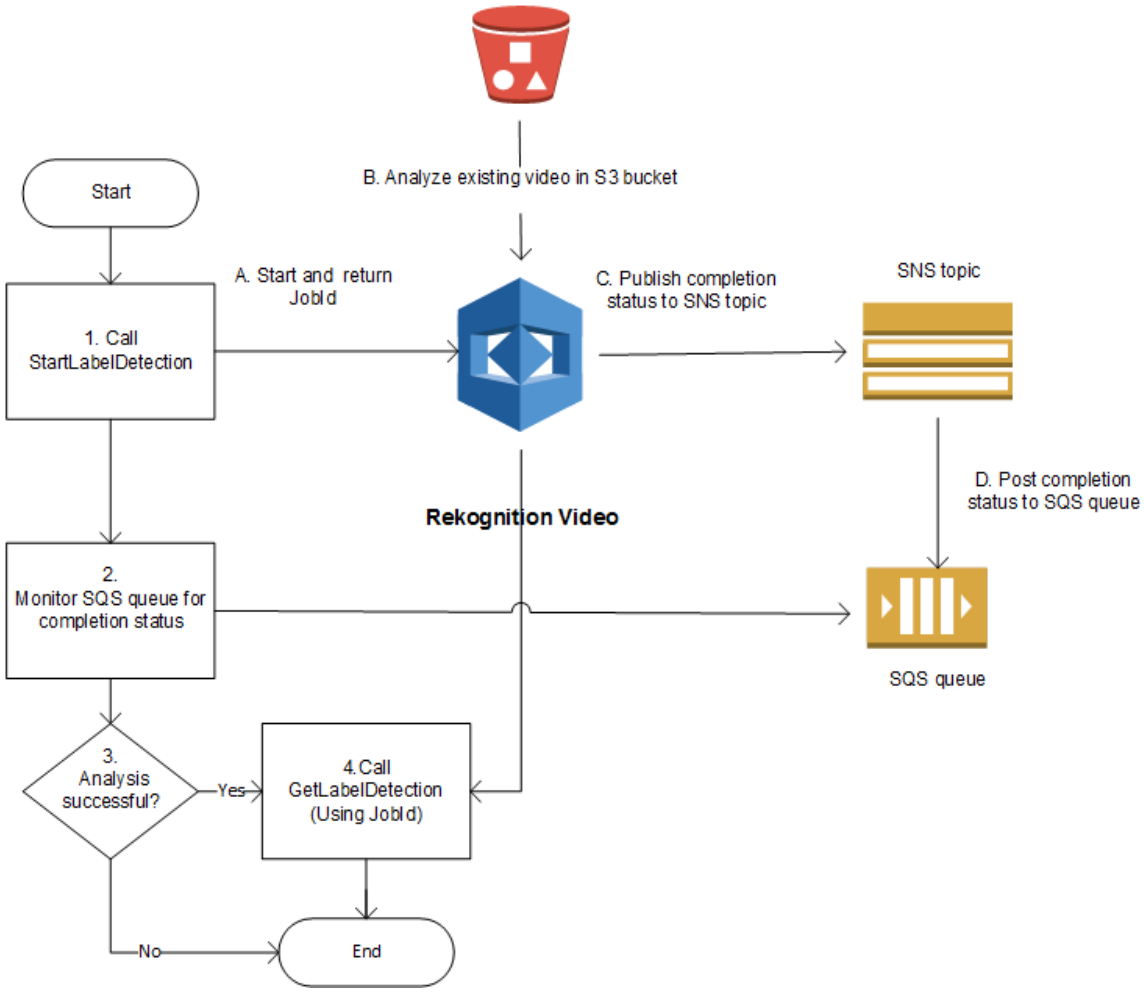
```
{
    "JobId": "270c1cc5e1d0ea2fbc59d97cb69a72a5495da75851976b14a1nnnnnnnnnnn",
    "Status": "SUCCEEDED",
    "API": "StartLabelDetection",
    "JobTag": "DetectingLabels",
    "Timestamp": 1510865364756,
    "Video": {
        "S3ObjectName": "video.mp4",
        "S3Bucket": "bucket"
    }
}
```

Figur 31 - Meldingen som SNS mottar ved en suksessfull utførsel av analysen

4.2.4.3 Hente resultatet av analysen fra SNS (punkt 4 figur 32)

For å hente resultatet av analysen må vi først forsikre oss om at den har lykkes ved å sjekke at Status parameteren er SUCCEEDED. Etter vi har sjekket det, kan vi sende en ny forespørsel getLabelDetection som tar inn en JobId til å finne den utførte analysen. Responsen på denne forespørselen vil da være en JSON objekt som inneholder all nødvendig data om videofilen.

Hele prosessen kan illustreres med dette diagrammet [27]:



Figur 32 - Analyse av objekt/scene med Rekognition

JSON filen som man får ved en vellykket analyse vil typisk være strukturert på en bestemt måte avhengig av hvilken type analyse som ble utført. Ved objekt/scene gjenkjenning, vil filen inneholde en liste av "labels". Disse merkelappene består av informasjon om en spesifikk gjenstand/scene i videoen (figur 34).

Ved analyse av ansikt, vil JSON responsen bestå av en liste Face objekter, hvor hver Face tilsvarer et ansikt gjenkjent i videofilen.

Disse responsfilene er strukturert på lik måte uavhengig av videofilen. Dette gir oss en forutsigbar respons som kan lett bli behandlet for videre prosessering og lagring i database (figur 33).


```
"Timestamp": 0,
"Label": {
    "Instances": [
        {
            "BoundingBox": {
                "Width": 0.11109819263219833,
                "Top": 0.08098889887332916,
                "Left": 0.8881205320358276,
                "Height": 0.9073750972747803
            },
            "Confidence": 99.5831298828125
        },
        {
            "BoundingBox": {
                "Width": 0.1268676072359085,
                "Top": 0.14018426835536957,
                "Left": 0.0003282368124928324,
                "Height": 0.7993982434272766
            },
            "Confidence": 99.46029663085938
        }
    ],
    "Confidence": 99.53411102294922,
    "Parents": [],
    "Name": "Person"
}
```


Figur 34 - Struktur av en Label objekt


```
"Face": {
    "BoundingBox": {
        "Height": 0.23000000417232513,
        "Left": 0.42500001192092896,
        "Top": 0.16333332657814026,
        "Width": 0.12937499582767487
    },
    "Confidence": 99.97504425048828,
    "Landmarks": [
        {
            "Type": "eyeLeft",
            "X": 0.46415066719055176,
            "Y": 0.2572723925113678
        },
        {
            "Type": "eyeRight",
            "X": 0.5068183541297913,
            "Y": 0.23705792427062988
        },
        {
            "Type": "nose",
            "X": 0.49765899777412415,
            "Y": 0.28383663296699524
        },
        {
            "Type": "mouthLeft",
            "X": 0.487221896648407,
            "Y": 0.3452930748462677
        },
        {
            "Type": "mouthRight",
            "X": 0.5142884850502014,
            "Y": 0.33167609572410583
        }
    ],
    "Pose": {
        "Pitch": 15.966927528381348,
        "Roll": -15.547388076782227,
        "Yaw": 11.34195613861084
    },
    "Quality": {
        "Brightness": 44.80223083496094,
        "Sharpness": 99.95819854736328
    }
},
"Timestamp": 0
```


Figur 33 - Struktur av en Face objekt

4.2.5 Struktur og funksjonalitet mellom AWS komponentene og Transcribe API

Prosessen av å hente data fra en lagret videofil utføres i 3 steg:

4.2.5.1 Starte video analysen med en startTranscriptionJob forespørsel

Forespørselen startTranscriptionJob tar inn et JSON objekt som inneholder informasjon om videofilen, samt spesifikasjoner for analysen.

```
{
   "LanguageCode": "string",
   "Media": {
      "MediaFileUri": "string"
   },
   "MediaFormat": "string",
   "MediaSampleRateHertz": number,
   "OutputBucketName": "string",
   "Settings": {
      "ChannelIdentification": boolean,
      "MaxSpeakerLabels": number,
      "ShowSpeakerLabels": boolean,
      "VocabularyName": "string"
   },
   "TranscriptionJobName": "string"
}
```

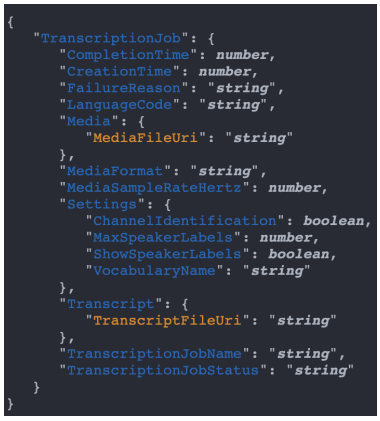
Figur 35 - Forespørsel syntax for startTranscriptionJob

MediaFileUri refererer til en streng som representerer stien til videofilen som ligger på S3 Bucket. LanguageCode er språket på lyden til videofilen som Transcribe trenger for å gjenkjenne riktig språk. Vi benytter oss av EnUS (Amerikansk engelsk), siden store deler av videofilene vi har tilgang til er på engelsk. MediaFormat spesifiserer hvilket format videofilen er, i vårt tilfelle er dette et .mp4 format. TranscripitionJobName har vi spesifisert til å være navnet på videoen vi skal analysere, for å gi et unikt navn på jobb navnet som Transcribe trenger. Dette jobb navnet lagres i en kø på AWS Transcribe sin tjeneste og kan brukes flere ganger om dette er ønsket.

4.2.5.2 Hente data fra getTranscriptionJob forespørsel etter utført analyse

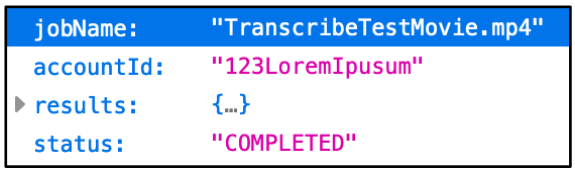
Responsen til getTranscriptionJob er et JSON objekt som inneholder informasjon fra transcription jobben vi startet. For å se status på jobben må man sjekke om TranscrptionJobStatus returner “COMPLETED”, på lik linje som Rekognition. I vårt tilfelle er det “TranscriptionFileUri” som inneholder en link til resultat vi er interessert i.

For å hente ut objektet vi ønsker benytter vi oss av et eksternt bibliotek til .NET Core som lar oss laste ned "TranscriptionFileUri" og definere det som en streng.



Figur 36 - Struktur av en Transcribe objekt

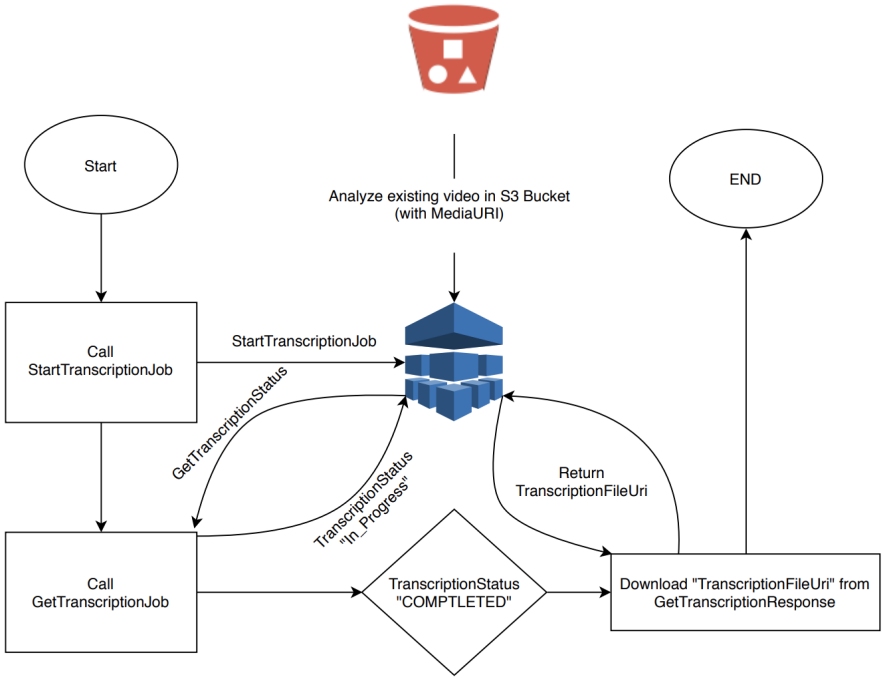
Denne strengen inneholder et JSON objekt som er strukturert på lik måte uavhengig av videofilen. Der jobName er som definert steg 1, videofilens navn. AccountId er identiteten til AWS kontoen benyttet for å starte jobben. Results inneholder en liste av



Figur 37 - Objekt fra Transcribtion URI

informasjon som faktisk er resultatet av jobben. Status er "COMPLETED" som indikerer at hele jobben gikk gjennom. Dette gir oss en forutsigbar respons som kan lett bli behandlet for videre prosessering og lagring i database.

Hele prosessen kan illustreres med dette diagrammenet:



Figur 38 - Analyse av video med Transcribe

4.3 Database arkitektur

Databasen er satt opp i Amazon sin DynamoDB. DynamoDB er en NoSQL database som har dynamiske skjemaer og er best egnet for en hierarkisk datastruktur. Dette formatet passer oppgaven vår veldig bra siden vi jobber med JSON objekter.

DynamoDB har en rekke med navneregler og datatyper som støtter under oppsettet til et "table" i databasen [20] De mest essensielle i vårt prosjekt er List, Map og String. Nedenfor beskrives disse mer i dybde.

List:

En liste attributt kan lagre en samling av verdier i en fast/ordnet rekkefølge. Listene er lukket med "square brackets: [...])". En liste er lik en JSON matrise. Det er ingen restriksjoner på hvilke datatyper som kan bli lagt i et List element og disse elementene trenger ikke å være av samme datatype.

```
FavoriteThings: ["Cookies", "Coffee", 3.14159]
```

Figur 39 - Eksempel på et List objekt

I vår database er det FacesDetection og LabelDetection resultatet fra Rekognition som lagres som liste. Det vil være dypere beskrivelse av oppsettet senere.

Map:

En Map attributt kan lagre en samling av navn-verdi par i en uordnet rekkefølge. Map er lukket med "curly braces: {...}". Map har likt oppsett som et JSON objekt. På lik linje med List attributtet, så er det ingen restriksjoner på hvilke datatyper som kan bli lagt i et Map element. Derfor er det godt egnet til å lagre JSON objekter som Map i DynamoDB.

```
{
    Day: "Monday",
    UnreadEmails: 42,
    ItemsOnMyDesk: [
        "Coffee Cup",
        "Telephone",
        {
            Pens: { Quantity : 3},
            Pencils: { Quantity : 2},
            Erasers: { Quantity : 1}
        }
    ]
}
```

Figur 40 - Eksempel på Map objekt

I vår database er det metadata fra videofilen og Transcription resultatet fra Transcribe som lagres som Map. Det vil være dypere beskrivelse av oppsettet senere.

String:

En de meste vanlige datatypene. Et String attributt er et element av ønsket verdi som tekst. Det er her vi har definert "Primary key" som er strengen til videofil navnet. Som er unik for hvert element i databasen.

4.3.1 Oppsett

Vi har laget et table i DynamoDB som vi benyttet til testing av produktet vårt. Det heter "BachelorTable" og her lagrer vi objektene som er essensielle som beskrevet ovenfor. Her ser vi at strengen "video" er primary-key og unik for hver video vi analyser med det ferdige programmet vårt. Faces og Labels lagres som List og Transcribe og videoMetadata lagres som Map.

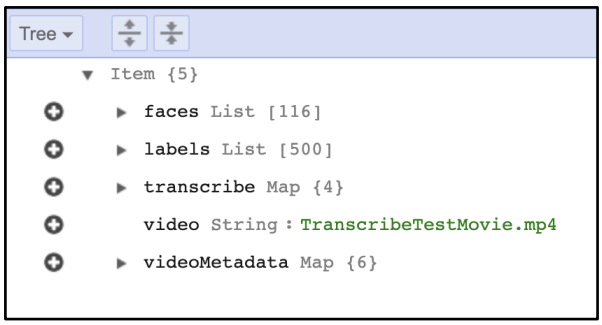
Oppsummert så lagres hele analysen av både Rekognition og Transcribe i denne tabellen.

Scan: [Table] BachelorTable: video — Viewing 1 to 4 items

video	faces	labels	transcribe	videoMetadata
TranscribeTestMovie.mp4	[{"M":{"Fa...	[{"M":{"L...	{"accountId":{"...	{"Codec":{"S":"h264"},"DurationMillis":{"N
ForBiggerEscapes2.mp4	[{"M":{"Fa...	[{"M":{"L...	{"accountId":{"...	{"Codec":{"S":"h264"},"DurationMillis":{"N
ForBiggerEscapes.mp4	[{"M":{"Fa...	[{"M":{"L...	{"accountId":{"...	{"Codec":{"S":"h264"},"DurationMillis":{"N
Boxing.mp4	[{"M":{"Fa...	[{"M":{"L...	{"accountId":{"...	{"Codec":{"S":"h264"},"DurationMillis":{"N

Figur 41 - Bilde fra DynamoDB

Her ser vi innholdet tilsvarende filmen "TranscribeTestMovie.mp4".



Figur 42 - Innholdet til en analysert film

4.3.2 Database operasjoner

I DynamoDB tilbyr fire operasjoner for CRUD funksjonalitet (Create, read, update og delete)

- PutItem – oppretter et "item" i tabellen..
- GetItem – leser et "item" i tabellen.
- UpdateItem – oppdaterer et "item" i tabellen.
- DeleteItem – sletter et "item" i tabellen.

I vår løsning benytter vi oss kun av operasjonene PutItem og GetItem. PutItem naturligvis for å få den behandlede dataen inn i databasen. GetItem for å få ut informasjon av databasen.

4.4 API design

Vårt API er bygd som et Web API med .NET Core 2.2 og har to spørringer til databasen.

getItems (GET)

Henter alle elementene i "BachelorTable" som beskrevet i delkapittelen ovenfor. Den vil returnere et JSON objekt som inneholder alle videoene i databasen per dags dato. Forespørselen ser slik ut:

```
https://localhost:5001/api/getItems
```

Om du ønsker å hente elementene tilhørt en spesifikk video, må du legge til en verdi i stien. Som ser slik ut:

```
https://localhost:5001/api/getItems?video=video
```

I dette eksemplet ønsker vi å hente data fra videofilen "ForBiggerEscapes.mp4" som allerede er analysert og ligger i databasen.

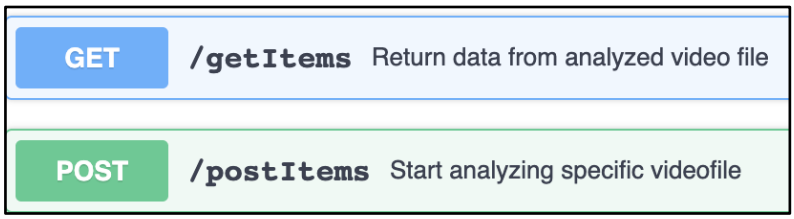
```
https://localhost:5001/api/getItems?video=ForBiggerEscapes.mp4
```

postItems (POST)

En POST request som setter i gang programmet og gjennomfører en analyse av ønsket videofil. Denne forespørselen starter analyse på filmen "ForBiggerEscapes.mp4". Det er viktig å notere seg at denne filmen må eksisterer i tilhørende S3 Bucket.

```
https://localhost:5001/api/postItems?video=ForBiggerEscapes.mp4
```

Her ser vi et skjermbilde fra Swagger API vi har definert. Her ser vi GET og POST forespørselen og tilhørende beskrivelse.



Figur 43 - Bilde av Swagger API

Kapittel 5 - EVALUERING

5.1 Evaluerings metode

Som beskrevet tidligere i kapittel 3, har vi valgt å gå frem med å evaluere resultatet hovedsakelig med bruk av kommunikasjonsmidler. Det vil si direkte tilbakemeldinger fra bedriften underveis i arbeidet og intervju med kontaktperson. Metoden av å evaluere resultatet med hjelp av Scrum har blitt lite aktuell for oss på slutten, siden vi fikk ikke benyttet prosjektrammeverket til sitt fulle.

5.1.1 Tilbakemeldinger under arbeidsperioden

I løpet av arbeidsperioden, har gruppen vært i kontakt med bedriften for å få tilbakemeldinger og veiledning. Dette var spesielt nyttig i begynnelsen av arbeidet, når det var uklart for oss hvordan vi skulle håndtere problemstillingen. Etter vi gikk over i utviklingsfasen, ble mengde kontakt med bedriften redusert, men selv om vi ikke hadde like aktiv kommunikasjon, tok fortsatt gruppen kontakt for veiledning når det trengtes. Ved sluttfasen av arbeidet, fikk utviklere fra bedriften prøve ut applikasjonen og samtidig gi tilbakemelding på temaer som kodekvalitet, utviklingsmetode og generelt inntrykk av arbeidet.

5.1.2 Intervju med kontaktperson

Mot slutten av prosjektarbeidet, har gruppen tatt kontakt med bedriftens tekniske kontaktperson for å evaluere resultatet av arbeidet og vurdere hvor godt problemstillingen ble besvart. Dette har vært den evalueringsmetoden som har gitt oss den mest verdifulle tilbakemeldingen, og har bidratt i en stor grad til analysen av sluttresultatet. Hele intervjuet kan leses i punkt 9.5 som vedlegg.

Hovedformålet med intervjuet var å besvare følgende tema:

- Bakgrunnen og motivasjonen til prosjektet
- Bakgrunn for bruk av maskinlæring i forhold til produktet
- Begrunnelse for valg av utviklingsmetodikk/tekniske verktøy
- Forventninger til resultatet av arbeidet
- Refleksjon over utført arbeid
- Planen videre

Vi kan konkludere følgende etter intervjuet ble gjennomført:

- Bakgrunnen for prosjektet var å innovere prosessen for ekstrahering av metadata. Som Rune nevner i intervjuet, må streaming tjenester manuelt legge inn metadata som de får fra leverandørene og eksterne kilder (Appendix 9.5, Spørsmål 3). En løsning som automatiserer denne prosessen ville spart kundene både tid og penger. Rune sier også i intervjuet at grunnen til at dette var et bachelorprosjekt var at Vimond ikke hadde prioriteringer på dette, men hadde veldig lyst å eksperimentere og danne et grunnlag for mulig videre arbeid. (Appendix 9.5, Spørsmål 16)
- Maskinlæring var ikke hovedfokuset, men åpner for nye muligheter når det gjelder videreutvikling av produktet. Ekstra funksjonalitet som f.eks faceblurring kunne blitt lagt til senere
- Prosjektet skulle utvikles i C# og med bruk av AWS verktøy for å simplifisere integrasjon i eksisterende løsninger. Mesteparten av Vimond sine produkter er utviklet med AWS tjenester, så valgt av programmeringsspråk og verktøy ble gjort hovedsakelig på basis av logistikk (Appendix 9.5, Spørsmål 7 og 9).
- Når det kommer til forventninger, nevner Rune dette i intervjuet: “Forventningene var å få et fornuftig strukturert program som snakker med alle komponentene (Rekognition og Transcribe) og få hentet ut resultat og lagre de.” (Appendix 9.5, Spørsmål 11). Dette ble reflektert i begynnelsen av arbeidet og har blitt fulgt opp.
- Resultat av arbeidet levde opp til Vimond sine forventninger. Rune nevner at disse ble’ ’absolutt oppfylt’’, og at måten vi har løst oppgaven på var’ ’ganske fornuftig’’ (Appendix 9.5, Spørsmål 12 og 13). Hvis produktet skal leve opp til Vimond sine standarder og kunne bli brukt til videre utvikling så må det forbedres først.
- Noen av de viktigste forbedringspunkt var feilhåndtering, enhets og integrasjonstester, kodestruktur og kvalitet.

5.2 Evaluerings resultat

Etter en gjennomført evalueringsprosess, har det blitt dannet et solid grunnlag for å kunne vurdere hvor godt problemstillingen ble besvart. Resultatet av evalueringen viser hvilke krav som ble oppfylt, og hvilke områder som fortsatt trenger mer arbeid.

Per dags dato, har produktet vi utviklet et grunnlag for videre utvikling og integrasjon. Kjernefunksjonalitet som vi har fått fra kravspesifikasjonene har blitt implementert og testet, og selv om noen av kravene har ikke blitt tilfredsstilt, er grunnlaget god nok til å kunne anses som en prototype som kan bli bygd videre på. Vi kan da konkludere flere punkter:

- Produktet sitt kjernefunksjonalitet av å ekstrahere informasjon fra videofiler ved hjelp av AWS tjenester har blitt implementert.
- Selv om løsningen er ikke bra nok for produksjon har det blitt dannet et solid grunnlag som kan bli brukt for videre utvikling.
- AWS Transcribe og AWS Rekognition fungerer godt nok til at dem kan forbedre dagens metoder.
- Partene i prosjektet har kommet til enighet at prosjektet når opp til forventet resultat og vært i fordel for alle involvert.

Kapittel 6 - DISKUSJON

6.1 Projektstart

Gruppen fikk oppgavebeskrivelse en måned før prosjektarbeidet skulle begynne. Dette tillot oss å bli litt kjent med problemstillingen på forhånd, og vi tenkte at det skulle gjøre prosessen av å sette seg inn i oppgaven lettere. Når prosjektarbeidet begynte offisielt, kunne vi gå rett på å sette opp en mer detaljert beskrivelse av oppgaven. Etter en del møter med veileder, fikk vi en klar oversikt over problemstillingen, kravene som skulle oppfylles og anbefalt approach vi kunne ta for å løse oppgaven med effektivt.

De første fasene i prosjektarbeidet ble dedikert hovedsakelig til planlegging av arbeidet. Vi måtte finne ut når og hvordan vi skulle jobbe. Siden to av medlemmene i gruppen hadde deltidsjobb ved siden av, måtte vi ta hensyn til det og tenke hvordan vi skulle få mest mulig tid på oppgaven. Vi ble da enige om å møtes ved alle anledninger hvor alle medlemmene av gruppen hadde tid til det, og samtidig fortsette å jobbe uavhengig av hverandre hvis en situasjon hvor et medlem ikke kunne jobbe skulle oppstå. Til slutt fikk vi en oversikt over hvilke verktøy vi skulle bruke og hvilke nye teknologier vi måtte lære oss.

6.2 Prosjektgjennomføring

6.2.1 Oppstartsfasen

Etter vi fikk en oversikt over hvilke komponenter vi skulle bruke begynte vi rett på å lære hvordan vi skulle bruke dem. For å oppnå ønsket resultat, måtte vi ta i bruk en kombinasjon av AWS komponenter, og det er i oppsettet av disse vi møtte på vår første utfordring. For å bruke Amazon sine tjenester, må administratoren av kontoen først konfigurere tilgang. Siden vi fikk helt nye brukere, måtte alle tilgangene konfigureres helt fra scratch. Dette ble ekstra utfordrende siden flere komponenter krevde spesielle innstillinger for å jobbe med hverandre, og alle måtte konfigureres av en administrator fra Vimond. Dette bidro til en del forsinkelser i oppstartsfasen, siden vi ikke kunne bruke komponenter vi ikke hadde tilgang til.

En annen utfordring som vi møtte på i denne fasen var den bratte læringskurven på flere av disse tjenestene. Ingen av medlemmene i gruppen hadde erfaring med Amazon sine tjenester, spesielt med bruk av disse i .NET. Vi ble derfor avhengig av eksterne ressurser

for å lære oss å beherske verktøyene vi skulle bruke. Flere av de tilgjengelige ressursene var også ikke tilrettelagt det programmeringsspråket vi skulle bruke, som gjør det ekstra utfordrende å få informasjon. Alt dette kombinert reduserte effektiviteten i oppstartsfasen kraftig, og mye tid ble brukt på å sette seg inn i diverse tema for å forstå hvordan vi skulle løse problemstillingen.

6.2.2 Utviklingsfasen

Etter å ha fått en litt bedre forståelse for teknologiene vi skulle bruke kunne vi starte med utviklingsprosessen. I utgangspunktet skulle vi holde oss til en Scrum lignende approach, men pga. begrensninger i kunnskap og dokumentasjon fokuserte vi heller på å prøve oss frem med forskjellige typer løsninger. Selv om det var ikke den mest strukturerte måten å løse problemstillingen på, klarte vi fortsatt å produsere kode jevnt og fokuserte på å implementere funksjonalitet gradvis etter kravspesifikasjonene.

Mesteparten av arbeidet gikk inn i å få til Rekognition API ’et til å fungere på en god måte. Dokumentasjonen refererte til flere måter å implementere dette på, så vi måtte gjennom flere iterasjoner for å finne noe som ville passet til vår problemstilling. Etter det var på plass, gjentok vi samme prosess med database integrering ved å prøve ut flere løsninger og velge den mest aktuelle for vårt produkt. Arbeidet med WebAPI ’et var den delen av utviklingen som var mest strukturert og tok kortest tid. Vi var kjent med teknologien og brukte verktøy som SwaggerAPI for å kartlegge hvordan API ‘et skulle se ut før implementasjon.

Ved slutten av utviklingen, hadde vi klart å produsere en applikasjon som oppfylte mesteparten av kravene, men manglet en del ekstra funksjonalitet som tester og grafisk grensesnitt. De største utfordringene under denne fasen var mangel på teknisk kunnskap (beskrevet i kapittel 6.2.1) og tidsbegrensninger relatert til innleveringer og personlige forhold. Samtidig som vi jobbet med utviklingen av produktet, hadde vi flere skolerelaterte oppgaver som skulle leveres inn på bestemte frister. Konsekvensen av dette var at vi måtte til tider bytte vårt fokus fra utvikling og heller bruke tiden på å skrive rapport/dokumentasjon.

6.3 Konsekvenser og videre arbeid

Som beskrevet i kapittel 6.2.1 og 6.2.2, var de største utfordringene mangel på kunnskap relatert til teknologi og tidsbegrensninger. Konsekvenser av disse utfordringene kan fordeles i tre kategorier:

- Forandring i arbeidsflyt
- Forsinkelser
- Mangel på ekstra funksjonalitet i sluttproduktet

Vår arbeidsflyt og prosjektmetodikk måtte justeres for å klare å opprettholde tidsfrister. Dette førte til en mindre organisert arbeidsstil og gikk ut over resultatet på det endelige produktet. Hvis vi holdt oss til en mer testet arbeidsmetode som Scrum, ville vi sikkert klart å håndtere utfordringene i utviklingsfasen på en bedre måte.

Når det gjelder forsinkelser, har det ikke påvirket oss i like stor grad som forandringer i prosjektmetodikk. Vi holdte de fleste fristene fleksible og klarte alltid å kompensere for eventuelle tidstap. Likevel har vi merket at det har bidratt til en mer stressende arbeidssituasjon, og vi kunne unngått det ved å fokusere mer på å sette klarere frister for våre mål.

Alle disse faktorene akkumulerte til slutt i et produkt som tilfredsstiller mesteparten av de funksjonelle kravene, men er samtidig manglende på noen fronter. Mye av utviklingsprosessen var lite organisert i forhold til prosjektmetodikken, og med kombinasjon med forsinkelser måtte vi prioritere kjernefunksjonalitet foran alt annet.

Kapittel 7 - KONKLUSJON OG FREMTIDIG ARBEID

7.1 Gjennomføring av målene

Vårt primære mål var å bygge en selvstendig tjeneste som kan ekstrahere metadata fra video. Denne tjenesten skal i en senere kunne kobles opp mot en AWS Step Function basert workflow.

Prosjektet har nådd målet ved å kunne hente analyse utført av AWS sine maskinlæringsalgoritmer. Den dataen vi henter får vi puttet i en database og hentet tilbake igjen gjennom et API. Vi kan da påstå at målet som vi ble satt har blitt nådd.

Vårt andre delmål inkluderer integrasjon av tjenesten i Vimond sine eksisterende systemer og utvikling av et grafisk grensesnitt for applikasjonen som skal bli brukt til å vise den fram på Expo. Produktet vi har utviklet kan potensielt integreres i en AWS Step Function basert workflow om det er ønskelig. Delmålet med et grafisk grensesnitt ble ikke oppnådd. Dette vil nære neste prioritet for å få noe som vi kan vise frem på Expo.

7.2 Fremtidig arbeid

Selv om prosjektet nådde alle målene som ble gitt av bedriften, så er det fortsatt store muligheter for forbedringer. Prosjektet ligger langt fra kode standarden til å kunne rulles ut til Vimonds kunder. De mest åpenbare forbedringene er som følger:

7.2.3 Feilhåndtering

Prosjektet vårt har ingen feilhåndteringer per dags dato. Det vil si at den sjekker ikke om tidligere arbeidet er gjort. Om man prøver å analysere den samme filmen to ganger vil applikasjonen enten gjøre hele arbeidet på nytt eller kræsje. Samme problemet oppstår hvis prosjektet plutselig skulle kræsje under et av arbeidene. Ved å implementere flere "killswitch"er i koden som vil stoppe programmet ved feil eller å dobbeltsjekke input fra brukeren ville vært en god start å få programmet mer fleksibelt og stabilt.

7.2.4 Samtidig kjøring

Programmet vårt tar ikke hensyn til å kjøre noe asynkront. Den venter på respons fra det ene APIet til AWS før den videre fortsetter med med neste API-kall. Selv om AWS kjøres stateless og åpner muligheten for at flere kall kan skje samtidig fra flere enheter, så tar vi ikke hensyn til dette. Ved implementering av samtidig kjøring ville vi triplet effektiviteten

av prosjektet. Dette kunne ha blitt gjort ved å splitte hovedtråden til programmet opp i tre tråder hvor enhver tråd kaller på APIet til AWS.

7.2.5 Filtrering av unødvendig data

Når prosjektet er ferdig med å analysere en film får vi flere tusen linjer med data. Problemet er at all denne dataen ikke er nødvendig å ha med når det skal lagres i databasen. Om disse dataene ble filtrert ville det vært lettere og raskere å håndtere responsen man får i etterkant.

7.2.6 Fjerne unødvendige meldinger

For hver film vi analyser produseres det en rekke med meldinger i SQS. Disse meldingene forblir i køen i 90 dager om dem ikke slettes etter hver analyse. Dette vil føre til at effektiviteten svekkes for hver jobb som kjøres ettersom prosjektet vårt går gjennom alle meldingene i køen fram til den ser arbeidet er ferdig. Istedenfor å manuelt slette meldingene som ligger i køen kunne man ha implementert en slettefunksjon som forsikrer at meldingene fjernes ved en vellykket analyse.

7.2.7 Legge til kommentar i koden

Koden vi har skrevet har ikke noe forklarende kommentar om hvordan koden fungerer. Vi burde fått inn noen kommentar på funksjonalitet til hver modul. Det burde også ha blitt laget en instruksjon på hvordan koden fungerer og hva som må gjøres for å få kjørt koden.

Kapittel 8 - REFERANSER

1. Bjørkeng, P. (2016). *Her er de to kursene alle trenger i fremtiden*. https://www.aftenposten.no/okonomi/i/lXQxG/Mener-alle-ma-ha-disse-to-kursene-i-fremtiden. Åpnet 25. mars 2019.
2. Løvlie, L. (2017). *Kunstig intelligens og maskinlæring*. https://www.soprasteria.no/tema/kunstig-intelligens-og-maskinlæring. Åpnet 25. mars 2019.
3. Buzzi, H. (2017). *AI og Maskinlæring*. https://www.bouvet.no/kurs/kategorier/maskinlaering. Åpnet 25. mars 2019.
4. Korosec, K. (2018). *Tesla is rolling out its Navigate on Autopilot feature* https://techcrunch.com/2018/10/26/tesla-is-rolling-out-its-navigate-on-autopilot-feature/?guccounter=1&guce_referrer_us=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_cs=nUzh5vFxQzgkktV4FCpEnA. Åpnet 25. mars 2019.
5. Tesla.com. (2019). *Autopilot*. https://www.tesla.com/no_NO/autopilot. Åpnet 26. mars 2019.
6. OpenAI. (2018). *OpenAI Five*. https://openai.com/five/. Åpnet 26 Mar. 2019.
7. Liquipedia Dota2 Wiki. (2018). *The International*. https://liquipedia.net/dota2/The_International. Åpnet 26. mars 2019.
8. "Vision - Google Cloud." https://cloud.google.com/vision/. Åpnet 20 mai. 2019.
9. "Features - Rider - JetBrains." https://www.jetbrains.com/rider/features/. Åpnet 20 mai. 2019.
10. "OpenAPI Initiative: Home." https://www.openapis.org/. Åpnet 20 mai. 2019.
11. "NET Core Guide - Microsoft Docs." 31 jul.. 2018, https://docs.microsoft.com/en-us/dotnet/core/. Åpnet 20 mai. 2019.
12. "About - Git." https://git-scm.com/about. Åpnet 20 mai. 2019.
13. "Trello." https://trello.com/. Åpnet 20 mai. 2019.
14. "What is Slack? – Slack Help Center." https://get.slack.help/hc/en-us/articles/115004071768-What-is-Slack-. Åpnet 20 mai. 2019.
15. "GitHub - facebook/react: A declarative, efficient, and flexible JavaScript ...." https://github.com/facebook/react. Åpnet 20 mai. 2019.
16. "Node.js." https://nodejs.org/. Åpnet 20 mai. 2019.
17. "Amazon Rekognition – Video and Image ...." https://aws.amazon.com/rekognition/. Åpnet 20 mai. 2019.
18. "Amazon Transcribe – Automatic Speech ...." https://aws.amazon.com/transcribe/. Åpnet 20 mai. 2019.
19. "Amazon DynamoDB - Overview - AWS ...." https://aws.amazon.com/dynamodb/. Åpnet 20 mai. 2019.
20. "Naming Rules and Data Types - Amazon DynamoDB." 15 feb.. 2016, https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.NamingRulesDataTypes.html. Åpnet 20 mai. 2019.
21. "What is Scrum? | Agile Alliance." https://www.agilealliance.org/glossary/scrum/. Åpnet 20 mai. 2019.
22. "REST - MDN Web Docs Glossary: Definitions of Web-related ... - Mozilla." 23 mar.. 2019, https://developer.mozilla.org/en-US/docs/Glossary/REST. Åpnet 20 mai. 2019.
23. "What is RESTful API? - SearchMicroservices - TechTarget." https://searchmicroservices.techtarget.com/definition/RESTful-API. Åpnet 20 mai. 2019.
24. "Hva er et Gantt-diagram?." https://www.gantt.com/no/. Åpnet 21 mai. 2019.
25. "HTTP | MDN." https://developer.mozilla.org/en-US/docs/Web/HTTP. Åpnet 21 mai. 2019.
26. "ASP.NET MVC Pattern | .NET." https://dotnet.microsoft.com/apps/aspnet/mvc. Åpnet 21 mai. 2019.
27. "Working with Stored Videos - Amazon Rekognition - AWS ...." https://docs.aws.amazon.com/rekognition/latest/dg/video.html. Åpnet 29 mai. 2019.
28. "Introduction to Test Driven Development (TDD) - Agile Data." http://agiledata.org/essays/tdd.html. Åpnet 31 mai. 2019.

Kapittel 9 - APPENDIX

9.1 Risiko liste (risikoanalyse)

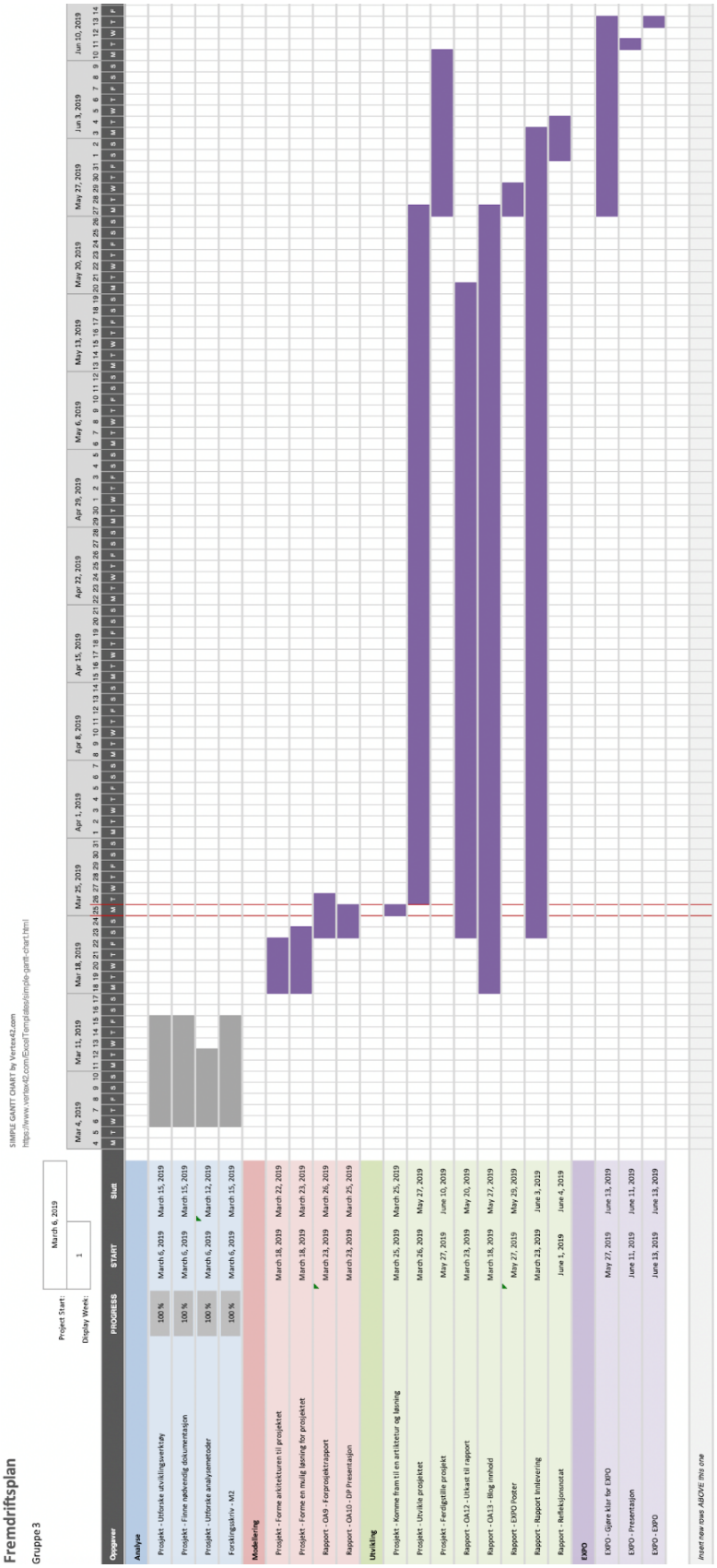
Suksessfaktor	S	K	RF	Tiltak
Forsinkelser ved tidsfrister	5	4	20	Sette klare tidsfrister tidlig og passe på at dem blir opprettholdt.
Tekniske problemer ved utvikling	4	6	24	Holde oss til anbefalte verktøy og løsninger.
Produktet kan ikke brukes	3	6	18	Opprettholde kommunikasjon med bedriften for å forsikre at applikasjonen er innenfor bedriftens standarder.
Produktet er ikke testet godt nok	4	5	20	Gjennomføre tester regelmessig under utvikling og forsikre at ingen av testene mislykkes.
Produktet blir ikke ferdig	6	6	32	Bruke fremdriftsplanen godt for å nå alle fristene i tide.
Juridiske problemer	2	7	14	Lese igjennom alle kontrakter godt og passe på at ingen av punktene brytes under prosjektet.
Dårlig kommunikasjon og arbeidsflyt	2	7	14	Passe på å holde aktiv kommunikasjon med veileder og bedriften samt dokumentere møter.

S = Sannsynlighet (Fra 1 til 10)

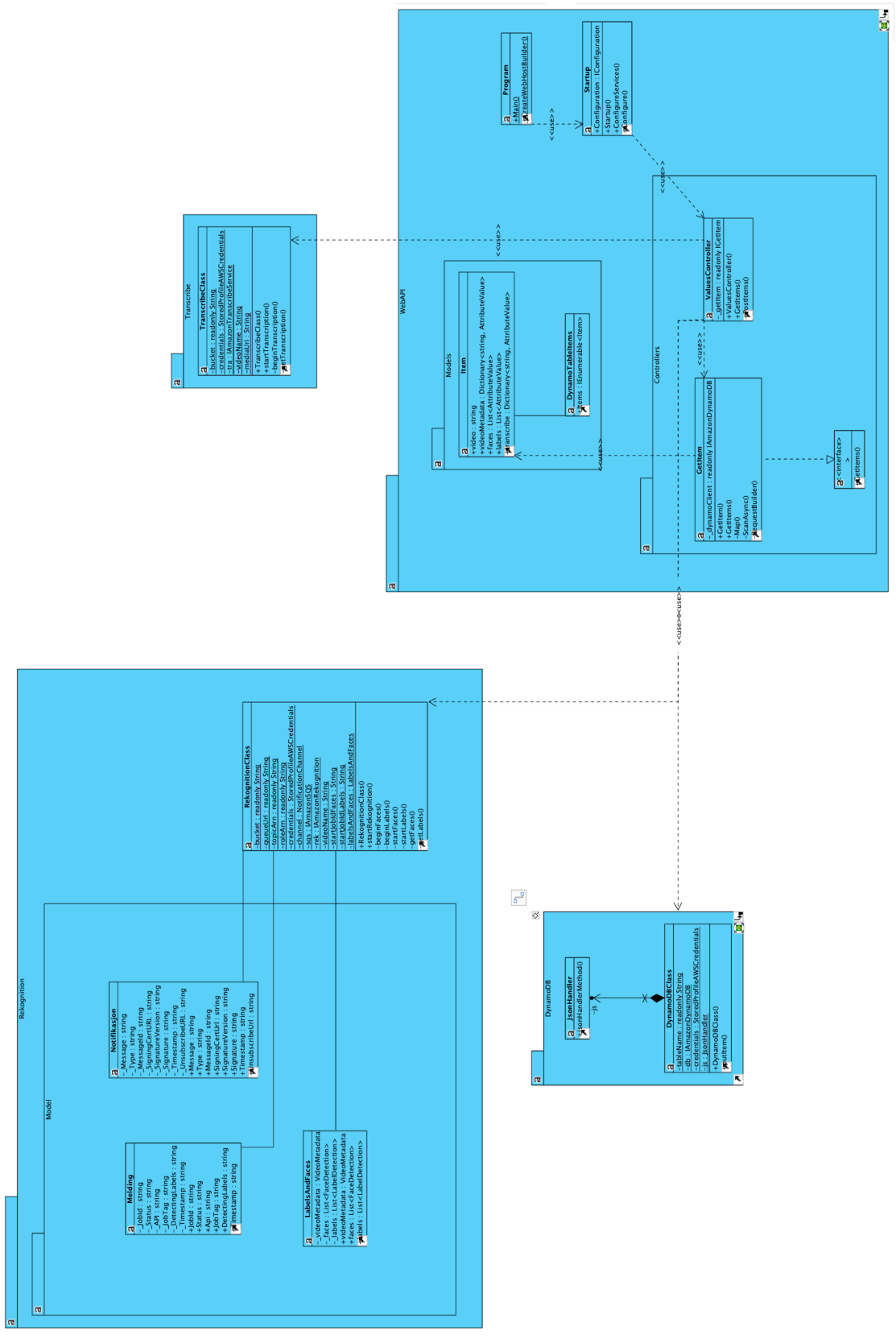
K = Konsekvens (Fra 1 til 10)

RF = Risikofaktor (S x K)

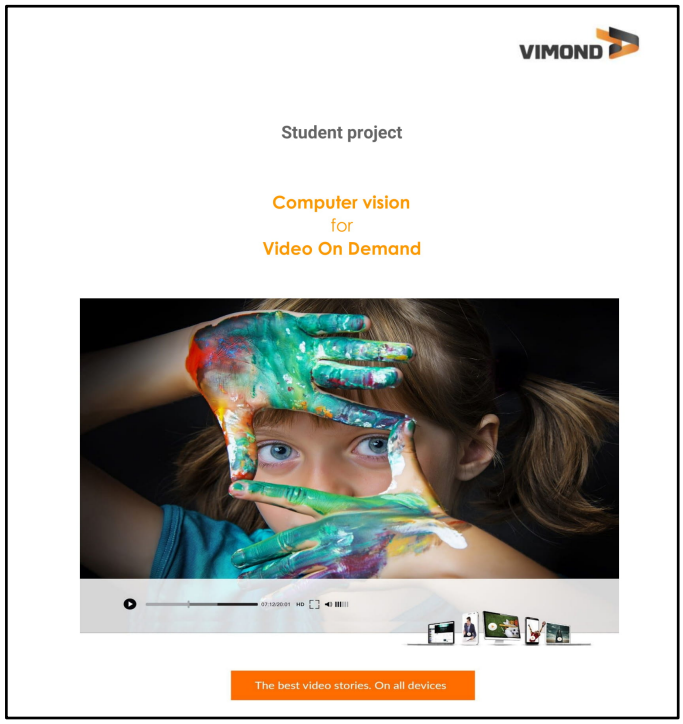
9.2 GANTT diagram



9.3 UML Diagram



9.4 Oppgavebeskrivelse



VIMOND

Objective

The high level objective of this project is to extract information from video files using machine learning.

Project background

In recent years, video streaming platforms have moved from novelty to commodity. To retain a cutting edge in the VOD market, Vimond is looking to extend its services with machine learning capabilities.

Tasks

1. Build a stand-alone service that can - at a later stage - plug in to an AWS Step Functions based workflow seamlessly.
2. Integrate the service with AWS Recognition and AWS Transcribe and use it to extract information from video files.
3. The information extracted must be persisted to a data store like Dynamodb or Elasticsearch.
4. Information that is to be extracted from videos include:
    a. Object, scene, and activity detection
    b. Facial recognition
    c. Speech-to-text

9.5 Intervju med Rune Jordal

Dato: 23. mai 2019
Sted: Vimonds lokaler

Info

Mikail Andreassen (omtalt som "**M**") er en av bachelor gruppemedlemmene og intervjuer.
Rune Jordal (omtalt som "**R**") er prosjektets eier og intervjuobjekt.

Spørsmål 1

M: Er det greit vi kan bruke det du sier som sitat i rapporten?
R: Absolutt!

Spørsmål 2

M: Hvilken del av prosjektet som kan regnes som konfidensielt?
R: Er ingen spesielle deler her som er veldig Vimond konkrete sånn sett. Så det blir det hovedsak, hvordan det skal brukes. Men selve løsningen er vel ikke veldig.

Spørsmål 3

M: Hvorfor trenger dere en slik løsning i systemet deres?
R: Det er jo flere bruksområder til dette. Typisk sånn det fungere i dag. La oss ta sånn VoD generelt, typisk streaming tjenester. Så får de gjerne levert videofiler fra leverandører, feks Hollywood Studios eller hva det måtte være. Da får de gjerne levert med en del metadata, altså regissører og litt sånt som det. Så finnes det løsninger som IMDB eller og tilsvarende som leverer ekstra metadata. Så driver gjerne å hente ann ut og legger den inn manuelt, gjøre noen integrasjoner med noen slike tjenester. Men i det hele så er det ikke veldig bra. I tillegg så koster IMDB fryktelig mye penger. Tross for at startet som et åpen prosjekt. På et tidspunkt så har det blitt den største databasen for film. Så det er jo det ene: hvis man kan klare å få inn sin egen metadata, så reduserer du kostnader fra IMDB. Du reduserer kostnadene for at noen manuelt må føre det inn.
R: Det andre er jo på live innhold som kommer inn, som klippes for eksempel TV2 de klipper "God Morgen Norge", nyheter, sport og på alt dette må noen manuelt gå inn i skrive inn denne metadataen. F.eks dato, hva skjedde på dette klippet og hvem var med i klippet osv. Masse ting som skal ligge der, så man kan legge dette i arkivet. Hvis alt dette kan gjøres automatisk vil man spare masse tid, etterhvert så vil antageligvis dette være en løsning som kan benyttes.

Spørsmål 4

M: Hvorfor trenger dere akkurat en maskinlærings basert løsning?
R: Det er jo egentlig ikke maskinlæring spesifikt som er behovet, det er vel bare en løsning på det at vi vil ha metadata. Dette var en enkel løsning på å få masse data. Men selvfølgelig med maskinlæring så tenker vi på andre områder som kan være interessante i fremtiden som f.eks automatisk "faceblurring". Akkurat nå så bruker de mye tid på å blurre ansikter manuelt. Så å kunne få ansiktene gjenkjent automatisk vil spare mye tid.

Spørsmål 5

M: Hvordan kunne effektiviteten på en slik løsning bli testet? (med tanke på kundene deres).
R: Da vil jeg tro at man kan sammenligne de dataen man klar å trekke ut i der med andre data fra studioets metadata og for eksempel IMDB. At man sammenligner hva klarer denne her maskinlæring algoritmen istedenfor studioet. Det vil jo være ganske åpenbart.
Også hastighetene, hvor raskt fra kundene våre kan publisere ut til sine portaler. Hvis det manuelt skal legges inn metadata så vil det forsinke prosessen. Om man da kunne kjøre gjennom video med maskinlæring så kan det hende man får et konkurransefortrinn foran konkurrenten.

Spørsmål 6

M: Hvor lenge har dere tenkt på å implementere maskinlæring i deres eksisterende løsninger?
R: Ikke videoteam, men på "IO-team" er det veldig aktuelt. Kommer ikke på noen annet. Vi er med på å se på de store tingene som prosessering av de store tingene som infrastruktur, styre hvor mye kapasitet vi trenger, kostnaden er ved å flytte infrastrukturen til en annen region som er billigere. Redusere denne kostnaden. Det kunne vært interessant. Det er ikke rett rundt svingen enda da.

Spørsmål 7

M: Hvorfor ville dere akkurat ha denne løsningen utviklet i C#?
R: Bakgrunnen er at videoteam jobber i C#, det er den enkleste veien til mål. Det ingen spesiell grunn det, vi er ganske fleksible på programmeringsspråk generelt og hoveddelen av Vimond jobber i Java og NodeJS. Det er videoteam spesifikt som bruker C#. Det er i bakgrunn av at Windows har vært mye bedre på videoprogramvare. Fra gammelt av var Windows definitivt best, det er så enkelt som det.

Spørsmål 8

M: Har dere tenkt på noen andre alternativer som for eksempel Google Cloud Vision eller IBM Watson? Hvorfor gikk dere for AWS?
R: Det er lett å svare på. Vi bruker allerede mye av Amazon sine tjenester. Hvis du skal ta en videofil å flytte den over til Google eller Azure (Microsoft) eller hva det måtte være, så man fysisk flytte den. Da kommer kostnader om flyttingen av filen, altså tidskostnader. Det vil forsinke vår videoprosessering betydelig, så derfor gjør bruker vi Amazon. Vi har hatt side prosjektet på de andre plattformene og interne prosjekter hvor vi har først på de andre mulige tjenestene. Det handler mer om logistikken fremfor effektiviteten av selve tjenesten (eller algoritmene tilbudt).

Spørsmål 9

M: Bruker dere noen andre AWS tjenester i deres systemer?
R: Det gjør vi, vi bruker noe til DocketRegistry, vi bruker Step Functions til å kontrollere arbeidsflyt, vi bruker S3 til lagring og playout storage, alle systemene de tilbyr.. Servere selvfølgelig, alle mulige servere instanser enten fast server eller serverless med muligheter som Lambda og Batch. Vi bruker nesten alt de har tror eg. Ganske stor kunde hos Amazon kan du si.

Spørsmål 10

M: På hvilken måte skal denne løsningen integreres i deres eksisterende systemer?
R: Denne her vil bli integrert i Orchestrator hos oss. Som er et sett av mikrotjenester som tar inn videofiler og transkoder filen og gjør filen klar til streaming. Så løsningen vil da bli som et steg i det systemet. Samtidig så som videofilen gjør seg klar til streaming så kan vi hente ut metadata, i vårt CMS. Det kan passe veldig fint der, men vi må også sørge for at kundene selv vil ha denne tjenesten. Det vil jo ha en kostnad å kjøre den gjennom Rekognition og Transcribe.

Spørsmål 11

M: Hvilke forventninger hadde dere til løsningen (arbeid vårt)?
R: Forventningene var å få et fornuftig strukturert program som snakker med alle komponentene (Rekogntion og Transcribe) og få hentet ut resultat og lagre de. Det har jo dere fått til ganske greit og fått ut den informasjonen vi vil ha.

Spørsmål 12

M: I hvilken grad ble disse oppfylt?
R: Oppfylt, absolutt.

Spørsmål 13

M: Hva synes dere om måten vi løste oppgaven på? (Med tanke på struktur av komponentene)
R: Dere har jo delt opp prosjektet ganske fornuftig og ja det ser greit ut. Ser ut som dere har tenkt en del gjennom. Vi har ikke noe å utsette på når det gjelder strukturen.

Spørsmål 14

M: Kan løsningen vi har utviklet bli brukt til videre utvikling?
R: Ja det kan den absolutt. Vi har jo noen standarder som vi må følge, men vi jobber ganske hardt med å øke kvaliteten på alt vi gjør så derfor krever vi enhetstester og integrasjonstester og at man følger gitte design patterns og kodeformat osv. Så vi er strengere og strengere for hver dag som går for å bli et selskap som autodeployer alt. Så da må du ha ganske høy standard på det

Spørsmål 15

M: Hva må gjøres hvis vi skal oppnå produksjonskvalitet etter deres standarder?
R: Det har påpekt noe av det selv. Feilhåndtering, den er innlysende det må på plass. Enhetstester for å verifisere endringer og integrasjonstester som ser at dette funker når det kommer til produksjonsmiljø. Når man har endringer så er det viktig å ha tester som sier at det fortsatt fungerer. Det er jo ofte kunder som allerede bruker tjenesten og plutselig så må man vite at alt er på plass.
Vi ville nok også godt gjennom kodestrukturen, men jeg synes det er veldig bra for å første forsøk. Det må jeg si.

Spørsmål 16

M: Hvorfor var dette et bachelorprosjekt? (Og ikke internprosjekt for eksempel)
R: Vi har jo hatt parallelle prosjekter, og det kunne fint ha vært et internt prosjekt men akkurat nå så er det ikke et fokusområde for oss, men det er mange av kundene våre som har dette som fokus. Så derfor er dette en fin mulighet for oss å eksperimentere med dette og ta første lærdommen for å få til grunnlag hvis vi skal gå videre med det, så vet vi akkurat hva som skal gjøres. Så det handler om prioriteringer, vi har ikke prioriteringer på dette, men vi har veldig lyst.