Bacheloroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

Andreas Nygård Ljøterud
Lars Stormark Pedersen
Mads Olsen Nekkøy

Koios: Plattform for visualisering og analyse av data

Bacheloroppgave i Ingeniørfag, data
Veileder: Tom Røise

Mai 2021

NTNU
Kunnskap for en bedre verden

Andreas Nygård Ljøterud
Lars Stormark Pedersen
Mads Olsen Nekkøy

Koios: Plattform for visualisering og analyse av data

Bacheloroppgave i Ingeniørfag, data
Veileder: Tom Røise
Mai 2021

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

NTNU

Kunnskap for en bedre verden

Sammendrag av Bacheloroppgaven

Argos Solutions sitt hovedprodukt er automatisk visuell inspeksjon av treplater, og det ligger mye potensial i dataen fra denne inspeksjonen. Argos har allerede en eksisterende rapportløsning, men har konkludert med at denne ikke tilfredsstiller dagens behov. Argos har derfor et ønske om å utvikle en analyseplattform for å bedre utnytte samlet inspeksjonsdata, og med dette øke kundenes utbytte av deres tjenester. Plattformens hovedfunksjonalitet er visualisering av data gjennom dashbord og rapporter, samt simulert endring av historisk produksjon basert på graderingskriterier. Kjernen av plattformen består av et rammeverk som tilbyr et felles datagrunnlag til disse tjenestene. Dette rammeverket er fleksibelt og legger opp til enkel videreutvikling og vedlikehold.

Summary of the Bachelor Thesis

Argos Solutions' main product is automatic visual inspection of wooden boards, and there is a lot of potential in the data generated from this process. Argos already has an existing reporting service, but has determined that the existing solution is inadequate for todays requirements. Argos wants an analysis platform to make better use of the available inspection data. The main functionality of the platform consists of visualization of data through dashboards and reports, along with a tool to simulate change in historical production based on grading parameters. The core of this platform is a framework which offers a shared data source to these external tools. The framework is flexible, extendable and maintainable.

Forord

Vi vil først og fremst takke Tor Nordseth, Anders Hørtvedt og Ole Martin Ruud hos Argos for spesielt godt samarbeid gjennom hele prosjektløpet. Det har vært veldig lærerikt og motiverende å gjennomføre prosjektet med deres gjennomgående støtte og vilje til problemløsning.

Vi vil også takke vår veileder Tom Røise for god veiledning og et godt samarbeid, i tillegg til andre professorer på NTNU for god hjelp.

Videre vil vi takke familie og bekjente for korrekturlesing og moralsk støtte. I den sammenheng ønsker vi å rette en spesiell takk til våre medstudenter Herman, Andrea, Anders og Eirik for gode tilbakemeldinger på rapporten.

Til slutt ønsker vi å takke hverandre for god innsats og godt samarbeid.

Innhold

Figurer

Tabeller

Listings

Ordliste

brainstorming metode for kreativ problemløsning [1]. 42

dekorere innebærer å utvide en funksjon med mer funksjonalitet uten å endre på innholdet. 46

epic er en betegnelse på et større arbeid som brytes ned i mindre, mer konkrete oppgaver. 10

HTTPS Utvidelse av HTTP som krypterer forbindelsen. 79

issue er en konkret oppgave som skal gjennomføres. xi, 11

kohesjon eller cohesion på engelsk, beskriver en modell der all relatert funksjonalitet lever i samme atomiske enhet. 35

lean er en metodikk som omhandler effektivisering av prosesser for tid eller ressurser. 10

package manager system for å håndtere biblioteker, for eksempel pip eller apt. 74

product backlog er en betegnelse for en liste med issues som skal gjennomføres. 10, 11

proof of concept en realisering av en idé med hensikt å demonstrere muligheter. 13, 58

refactor er å skrive om kode for øke kodekvalitet. 14

skanner er en kortversjon av Argos Grading System scanner, som er maskinene som inspiserer platene. 1

sprint er en avgrenset repeterbar tidsperiode i Scrum med klare mål for utvikling. 9

sprint planning meeting er et planleggingsmøte som holdes initielt i hver sprint hvor arbeidet som skal gjennomføres defineres. 10, 11, 13

sprint retrospective er et møte som repeteres etter hver sprint, hvor formålet er å lære av den foregående perioden og dermed bli mer effektiv i den neste. 9, 10

sprint review meeting møte i slutten av Scrum sprinter for å få tilbakemelding fra produkteier om fremdriften. 10

SQL-injeksjon er et angrep som injiserer ondsinnede SQL-spørringer, for å angripe databasen. 26, 41

Akronymer

CIA Confidentiality, Integrity, Availability. 78

CRUD Create, Read, Update, Delete. 38

GraphQL Graph Query Language. 29

HTTP HyperText Transfer Protocol. 29

JSON JavaScript Object Notation. 29

JVM Java Virtual Machine. 29

KDE Kernel Density Estimation. 66

ORM Object Relational Mapping. 30

PR Pull Request. 11

REST REpresentational State Transfer. 29

SDLC Software Development Life Cycle. 9

SPA Single Page Application. 51

UCD User centered design. 82

VPN Virtual Private Network. 88

XML eXtensive Markup Language. 41

1 Introduksjon

1.1 Bakgrunn

> "Argos Solutions AS er et globalt høyteknologi-selskap som holder til på Kongsberg. Selskapet utvikler og produserer avanserte systemer for automatisk visuell inspeksjon til ledende treplateprodusenter verden over. Systemene benyttes i fabrikker for å automatisere og optimalisere produksjonsprosesser." [2]

Levering og drifting av det automatiserte inspeksjonsverktøyet Argos Grading System er del av Argos' kjernevirksomhet. Systemet analyserer forskjellige typer treplater og gir en vurdering av kvalitet basert på fabrikkens krav. Dataene om hver enkelt plate og informasjon om eventuelle defekter lagres i en database.

For å vise resultatet av denne prosessen for ikke-teknisk personell, finnes det i dag en løsning som genererer rapporter basert på forhåndsdefinerte maler. Over tid har det blitt oppdaget flere svakheter ved denne løsningen. De forhåndsdefinerte malene inneholder data som ofte ikke er interessant for kunden, og det er vanskelig å endre disse etter fabrikkenes individuelle behov. Det er dermed vanskelig for kundene å benytte seg av innsamlede data for å få innsikt i produksjonsprosessen.

Argos er derfor interessert i å utvikle en analyseplattform for å håndtere dette. Gjennom plattformen ønsker Argos å tilby sanntidsstatistikk, et mer fleksibelt rapporteringssystem og simuleringsmuligheter. Ambisjonen er at disse verktøyene skal gi kunden enklere tilgang til produksjonsdata og dermed øke deres utbytte av Argos' tjenester.

1.1.1 Hvordan brukes Argos Grading System?

Argos Grading System maskinene, som vi herfra og ut vil referere til som skannere, står helt i senter av vår oppgave. Disse maskinene samler inn all dataen vi jobber mot, og vi anser det derfor som viktig å gi en kort innføring i hvordan de fungerer og hvordan de brukes i praksis.

Ved hjelp av høyoppløselige kameraer og lyskilder i forskjellige vinkler, opparbeider skannerne (se figur 1) seg et helhetlig bilde av hver plate [3]. Denne informasjonen settes opp mot produktets krav, og brukes videre for å kategorisere platene etter kvalitet[1]. Kategoriseringen brukes deretter av eksterne systemer for å sortere ut plater (se figur 2) som må repareres eller vrakes.

Det er ofte flere skannere per fabrikk, og de er gjerne plassert på kritiske punkter

[1] Kvalitet beskriver her graden av samsvar med kravene satt av kunde.

Figur 1: Argos Grading System skanner [3].

Figur 2: Skanner med tilhørende sortering ved hjelp av sugekopper [3].

i produksjonslinjen - eksempelvis etter påføring av laminat eller pussemaskin. På bakgrunn av dette er det, gjennom data som samles inn av skannerne, mulig å danne et helhetlig bilde av produksjonen. Det er derfor et urealisert potensiale i dataen som blir samlet.

1.2 Oppgavebeskrivelse

Vi skal levere en analyseplattform som kan rulles ut internt i en fabrikk eller i skyen. Det skal i tillegg gjennomføres en generell undersøkelse av fordeler og

ulemper ved bruk av skytjenester, med spesielt fokus på faktorene ytelse, kostnad og risiko. Gjennom analyseplattformen skal det være mulig å:

- Se løpende statistikk over nåværende produksjon i sanntid gjennom et dashbord. Det bør også være enkelt å tilpasse innholdet i dashbordet til ulike fabrikker.
- Generere skreddersydde rapporter med oversikt over en gitt tidsperiode. Rapportene bør også enkelt kunne skrives ut i papirformat.
- Se utfallene av en justering på klassifiseringsalgoritmen, ved at man øker eller senker toleransen for feil.

1.2.1 Avgrensning

- Analyseplattformen skal aggregere og analysere data fra produksjon, men eventuelle endringer som skal gjøres som følge av dette vil ikke gjøres gjennom vårt produkt.
- Vi vil levere et produkt som støtter internasjonalisering, men vi tar ikke ansvar for implementasjon av språk og enheter utover det som er oppgitt i kravspesifikasjon.
- Dashbordets grafiske utforming tar utgangspunkt i en større skjermflate, og vi vil ikke ta hensyn til synlighet på mobile enheter.
- Sluttproduktet skal være fungerende, men ikke nødvendigvis klart for produksjon.

1.3 Prosjektmål

1.3.1 Resultatmål

Prosjektet kan deles i fire hovedområder:

1. Det skal være mulig å lage skreddersydde rapporter til hver enkelt fabrikk. Rapportene må enkelt kunne skrives ut i PDF-format.
2. Det skal være mulig å se løpende oversikt over produksjonen innenfor angitte tidsområder.
3. Det skal være mulig å simulere historisk produksjonsdata med forskjellige målekriterier.
4. Det skal gjøres en vurdering av fordeler og ulemper ved bruk av skyløsninger.

Ved prosjektets slutt ønsker Argos at de fire punktene over er operative, slik at det kan vises og eksperimenteres med.

1.3.2 Effektmål

Argos har et ønske om at analyseplattformen skal bidra til å:

- gi kunder enklere tilgang til produksjonsdata slik at feil oppdages raskere og svinn minimeres. Dette vil øke antall godkjente plater per skift.
- gi kundene økt utbytte av deres tjenester. Dette vil ha positiv effekt på selskapets fortjeneste og tilfredsstillelse blant kundene.
- redusere tid Argos bruker på vedlikehold og videreutvikling.

1.3.3 Læringsmål

Etter endt bachelor ønsker vi å ha mer kunnskap om:

- verktøy som er aktuelle i arbeidslivet, slik at vi er mer effektive i daglig arbeid.
- å jobbe sammen i team, for å bli bedre forberedt til arbeidslivet.
- hvordan det er å jobbe med en smidig utviklingsmetode.
- dokumentasjon, mer spesifikt hvordan dokumentere slik at programvaren lett kan videreføres.
- testing og kvalitetskontroll
  - rutiner for testing og bruk av ulike verktøy.
  - forstå viktigheten til testing.
  - effekten av god kvalitetskontroll.
- hvordan utforme og skrive en informativ rapport.

1.4 Rammer

- Tidsrammen for prosjektet er satt ved leveringsfrist 20. mai 2021.
- Løsningen skal utvikles med forbehold om at en potensiell migrering til sky kan forekomme i fremtiden. Dette innebærer at løsningen enkelt skal kunne migreres og kjøres i skyen dersom dette blir aktuelt.

- Gruppen vil gjennom prosjektløpet måtte ta hensyn til to eksisterende teknologier og versjoner som foreligger av disse:
  - Windows Server 2019
  - SQL Server 2012

1.5 Organisering

1.5.1 Gruppemedlemmer

Gruppen består av tre studenter ved NTNU i Gjøvik, hvor samtlige studerer dataingeniør. Dette gjorde at vi hadde et nokså likt utgangspunkt med tanke på faglig kompetanse, med unntak av forskjeller i valgemner.

Gjennom studieløpet har vi hatt flere emner som har gitt oss nyttige kunnskaper i møte med bacheloroppgaven. Systemutvikling er et av de mer fremtredende emnene, da planlegging, prosjektstyring og arbeidsmetodikk har ringvirkninger gjennom alle prosjektets faser. Til tross for vår manglende erfaring med prosjekter av denne størrelsesorden, har lærdom vi tilegnet oss i dette emnet vært uvurderlig.

I forhold til valgemner har vi stor spredning i gruppen, noe vi anser som en fordel. Den spredte kompetansen gjorde at vi kunne berike ulike deler av prosjektet, som også er noe vi tok hensyn til i fordelingen av roller og arbeidsoppgaver.

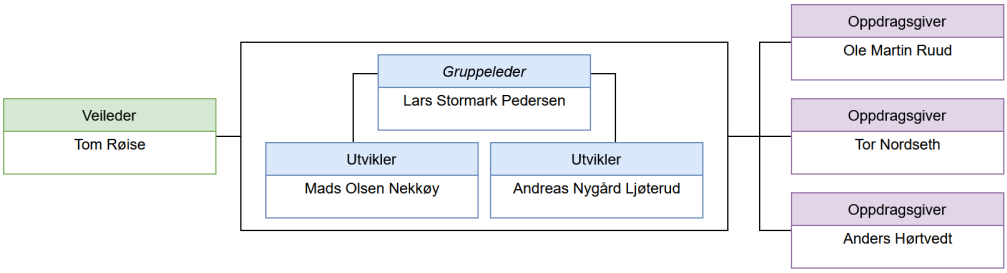
1.5.2 Roller

Vi bestemte tidlig at det var hensiktsmessig å dele oppgaven i ulike deler, og dermed delegere roller etter disse delene. Etter den initielle oppdelingen, ble det tidlig bestemt at dette skulle gjøres basert på hvilken av delene som tiltalte vedkommende mest, og ikke nødvendigvis der man hadde mest bakgrunnskunnskaper. Til tross for at dette tilsynelatende er et mindre effektivt alternativ, anså vi interesse og læringsutbytte som viktigere momenter enn ren effektivitet. I tillegg slo vi tidlig fast at disse rollene ikke skulle være absolutte, men heller fungere som et overordnet ansvarsområde. I praksis betydde dette at vi samarbeidet tett med alle deler av prosjektet, til tross for vår delegering av roller.

- **Gruppeleder, simulering og infrastruktur**
  Lars Stormark Pedersen hadde ansvar for infrastruktur og simuleringsdelen av oppgaven.

  I tillegg til å være utvikler, har Lars på bakgrunn av sin erfaring fra arbeidslivet blitt utnevnt gruppeleder. Dette har innebært et overordnet ansvar for fremdriften i prosjektet.

- **Utvikler, dashbord og design**
  Mads Olsen Nekkøy hadde hovedansvaret for dashbordet og alt design-relatert.
- **Utvikler, rapportgenerering og kvalitetsansvarlig**
  Andreas Nygård Ljøterud var hovedansvarlig for rapportgenerering, og hadde i tillegg hatt et overordnet ansvar for kvalitetsikring og dokumentasjon.
- **Veileder**
  Tom Røise, universitetslektor ved NTNU i Gjøvik.
- **Oppdragsgiver**
  Argos Solutions AS, representert ved Tor Nordseth, Anders Hørtvedt og Ole Martin Ruud.



Figur 3: Organisasjonskart.

1.6 Hvorfor valgte vi oppgaven

Vi fikk høre om Argos og muligheten for et potensielt samarbeid gjennom vår gode venn og tidligere medstudent Ole Martin Ruud, som nå er ansatt i bedriften. Det ble tidlig i prosessen opprettet dialog, hvor vi ble presentert med tre ulike oppgaver vi kunne velge mellom. Analyseplattformen vekket umiddelbar interesse blant samtlige i gruppen. Vår ambisjon var at oppgavens allsidighet ville introdusere faglige utfordringer på flere relevante fagområder, noe som var svært tiltalende for oss.

1.7 Målgrupper

Prosjektet i sin helhet inneholder leveranser til flere parter som hver har sine behov. Vi har derfor valgt å dele målgruppen i to deler: rapport og analyseplattform.

1.7.1 Rapport

Målgruppen for rapporten er i første rekke sensor, som skal vurdere prosjektet, og vår veileder, Tom Røise. På bakgrunn av dette er rapporten skrevet med utgangspunkt i at leser har teknisk innsikt, men ikke nødvendigvis kjennskap til spesifikke teknologier som er brukt.

1.7.2 Analyseplattform

For analyseplattformen i sin helhet, anser vi Argos og bedriftens kunder som vår primære målgruppe. Vi har derfor gjennom hele prosjektløpet hatt tett kommunikasjon med oppdragsgiver, for å best mulig forstå bedriftens og sluttbrukernes ønsker og behov.

1.8 Om rapporten

1.8.1 Navngivning

Navnet *Argos* kommer fra gresk mytologi, og blir gjerne brukt i sammenheng med *Sees med argosøyne*. Inspirert av dette, valgte vi å kalle prosjektet *Koios*. Koios er den greske guden for visdom og nysjerrighet [4]. Navnet oversettes i moderne tid til querying, questioning eller intelligence [5], noe vi synes var passende med tanke på bruksområdet til vårt system.

1.8.2 Rapportens struktur

Rapporten er strukturert sekvensielt på lignende måte som om prosjektarbeidet hadde fulgt en fossefallsmodell.

- **Utviklingsprosessen** redgjør for hvordan gruppen har gått frem for å systematisere arbeidet.
- **Kravspesifikasjon** beskriver kravene som har blitt satt til systemet.
- **Teknologier** gir en oversikt over hvilke teknologier som har blitt valgt.
- **Design** definerer hvordan systemet må utformes for å ivareta kravene.
- **Implementasjon** beskriver hvordan design- og teknologivalgene har blitt realisert.
- **Testing** viser hvordan implementasjonen har blitt kvalitetssikret.

- **Installasjon** omhandler relevante faktorer for produksjonssetting.
- **Sikkerhet** redgjør hvordan systemet ivaretar sikkerhetsutfordringer.
- **Drøfting** diskuterer resultatet av prosjektet, og hvordan utviklingsprosessen har vært.
- **Konklusjon** oppsummerer hele oppgaven og beskriver videre arbeid.

2 Utviklingsprosess

I dette kapittelet går vi gjennom hvordan vi planla å gjennomføre utviklingsarbeidet. Vi beskriver hvilke avveininger vi gjorde ved valg av utviklingsmodell, hvilke verktøy vi benyttet og en risikoanalyse av prosjektperioden.

2.1 Systemutviklingsmodell

For å planlegge og strukturere arbeidet, er det i et hvert utviklingsprosjekt essensielt å etablere en systemutviklingsmodell [6]. Det eksisterer mange typer modeller, som er tilpasset prosjekter med ulike omfang, struktur og omstendigheter. Det vanskelige blir derfor å velge modellen, eller de kombinasjonene av modeller, som passer best for vårt prosjekt. For å ta en best mulig beslutning, startet vi med å avdekke de faktorene i vårt prosjekt vi anser som de mest styrende for valg av utviklingsmodell:

- Gruppen består av tre utviklere og prosjektet skal gjennomføres på en relativt kort tidsperiode.
- Gruppen har lite erfaring med prosjekter av denne størrelsen.
- Utover de funksjonelle kravene i oppgavebeskrivelsen (vedlegg A), legges det ingen konkrete føringer til utforming og valg av teknologier.

På bakgrunn av de to sistnevnte faktorene, bestemte vi oss tidlig for å fokusere på smidige over plandrevne modeller. Plandrevne modeller gjennomfører utviklingsfasene i Software Development Life Cycle (SDLC) individuelt og i kronologisk rekkefølge. Modellene legger opp til at endringer ikke blir gjort i tidligere fullførte faser dersom problemer oppdages. I smidige modeller er det heller lagt opp til å gjennomføre fasene parallelt, som gjør det enklere å ta hensyn til nye krav eller revidere tidligere beslutninger med ny informasjon. Denne arbeidsflyten kombinert med vår begrensede erfaring og muligheten til å lære av tidligere utviklingsperioder, gjør at vi anser smidige modeller er mest aktuelle for vårt prosjekt.

Mer spesifikt valgte vi i all hovedsak å se på Scrum og Kanban, da vi hadde kjennskap til begge disse modellene fra tidligere i studieløpet. Gjennom Scrum sine sprinter kan vi planlegge hva utviklingsfokuset skal være for hver utviklingsperiode. Etter hver sprint legges det opp til å vurdere hvordan perioden gikk gjennom en sprint retrospective. Formålet med dette møtet er å lære av gruppens feil og erfaringer, slik at videre arbeid blir bedre. Samtidig var det også aspekter med Scrum vi anså som mindre optimale. Fundamentet i Scrum bygger på å definere etablerte roller. Med en gruppestørrelse på tre medlemmer

anså vi risikoen for at mye tid ville gå til prosesstyring som høy. Spesielt siden ingen på gruppen hadde erfaring med rollene fra tidligere, kunne dette ta verdifull tid bort fra utvikling.

Dette gjorde at vi undersøkte muligheten for å kombinere lean metodikker med Scrum. Her peker Kanban seg ut som et populært alternativ. Kanban ofrer noe av strukturen som Scrum tilbyr, i et forsøk på å maksimere effektivitet og flyt i utviklingen. Hovedfokus i Kanban ligger på å optimalisere arbeidsflyt gjennom visualisering i det som kalles en Kanban-tavle [7]. Tavlen fungerer ved å flytte oppgaver mellom predefinerte kolonner, med begrensinger for hvor mange oppgaver som kan være i hver kolonne [7]. Formålet med dette er å sikre at oppgaver ikke setter seg fast eller glemmes. Ulempene med Kanban er at det kan være for lite struktur for oppgaver som har en klart definert tidsramme og at tavlen fort blir utdatert på grunn av manglende oppfølging.

Scrum gir oss et godt grunnlag for å helhetlig styre prosjektet, mens Kanban er god på daglig drift. En kombinasjon av disse modellene, ofte kalt Scrumban, gir en utviklingsmodell som tilfredsstiller våre krav og rammer. Vi bestemte dermed oss for å gå for Scrumban som utviklingsmodell.

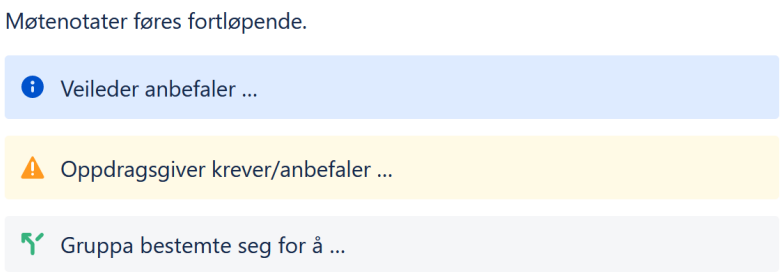
2.2 Gjennomføring

Vi valgte å gjennomføre sprinter på to uker, hvor vårt bi-ukentlige sprint review meeting med oppdragsgiver markerte slutten av hver sprint. Formålet med disse møtene var å presentere resultatet av den foregående sprinten og få tilbakemelding på det vi hadde gjort. På slutten av hver sprint gjennomførte vi et sprint retrospective møte for å reflektere over den forbigåtte sprinten.

De retrospektive møtene fungerte som en naturlig overgang til våre sprint planning meeting møter. Disse gjennomførte vi før hver sprint, der målet var å definere hvor fokusområdet lå for den kommende sprinten. Her definerte vi konkrete mål for perioden og hentet nye oppgaver fra product backloggen. Gjennom hver sprint hadde vi også ukentlige møter med veileder, hvor vi også mottok tilbakemelding på arbeidet vårt, men hvor fokus var mer rettet mot leveransen til NTNU. På alle eksterne møter ble det ført referat etter vår definerte stilguide (se figur 4). Dermed var det enkelt å tilegne seg informasjon fra tidligere referater, dersom dette var nødvendig. Eksempler på møtereferater kan sees i vedlegg H og I.

2.2.1 Arbeidsflyt

Da prosjektet i sin helhet består av separerte deler, er det hensiktsmessig å organisere relaterte arbeidsoppgaver i samlede bolker. For å gjøre dette valgte vi å bruke Jira sine epics. Vi lagde en epic for hver av hovedkategoriene i oppgaven

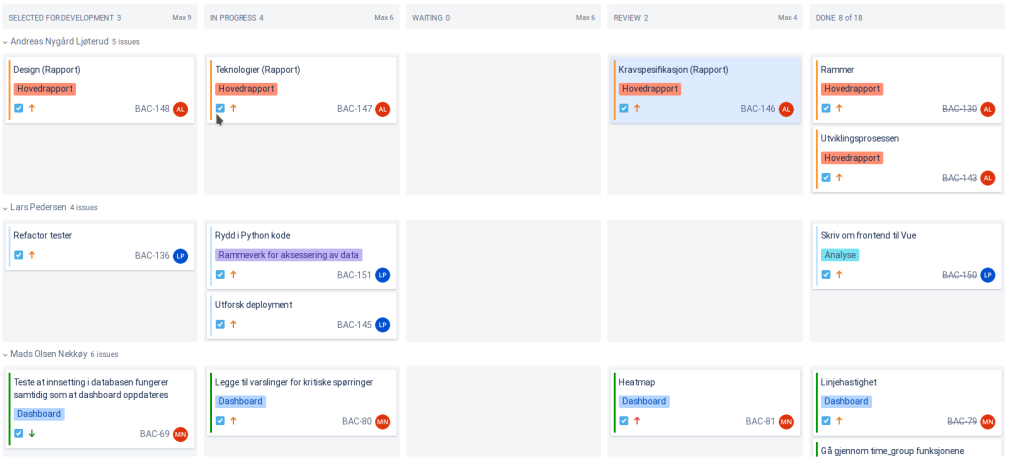


Figur 4: Stilguide for møtereferatene.

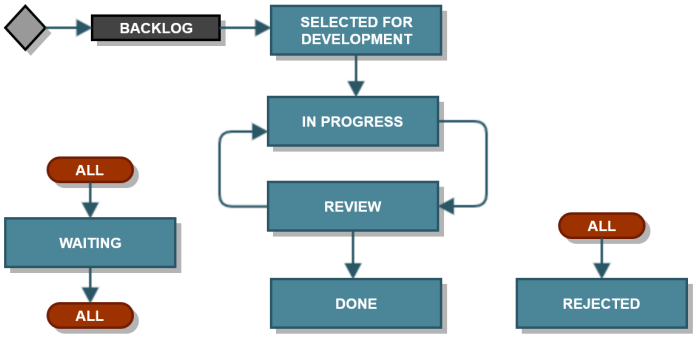
og delte disse inn i konkrete issues. Dette medførte at vi lettere kunne holde oversikt over progresjon i prosjektet. Ved hvert sprint planning meeting lagde vi oppgaver som videre ble koblet til tilhørende epics før de ble plassert i product backlog. Oppgavene ble etter beste evne delt opp i like store deler, og vi hadde som mål at gjennomføring skulle ta mellom tre og fem dagsverk per oppgave. For å opprettholde oversikten over oppgavene brukte vi Jira og dens tilhørende Kanban-tavle (se figur 5). Tavlen valgte vi å dele inn i fem kolonner:

- **Selected for development**
  Oppgaver som er hentet fra product backlog, men som enda ikke er påbegynt.
- **In progress**
  Påbegynte oppgaver.
- **Waiting**
  Oppgaver som er påbegynt, men som nå venter på en ekstern faktor før arbeid kan fortsette. Et eksempel på dette kan være at oppklaring til funksjonalitet kreves fra oppdragsgiver.
- **Review**
  Oppgaver som er ferdige, men som venter på gjennomgang av de andre gruppemedlemmene for kvalitetssikring. Før en oppgave skulle plasseres i denne kategorien, satt vi krav om at det skulle foreligge en Pull Request (PR).
- **Done**
  Ferdigstilt oppgaver. Vi anså en oppgave som ferdigstilt dersom den hadde blitt klarert av begge de andre medlemmene på gruppen. Dersom det oppsto behov for endringer, åpnet vi nye issues som igjen gikk gjennom den samme prosessen beskrevet over.

For hver av kolonnene begrenset vi antallet oppgaver som den kolonnen kunne inneha til enhver tid. Formålet med dette var å forhindre en oppbygning av halvferdige oppgaver og eventuelle flaskehalser. Denne grensen satt vi initielt til



Figur 5: Kanban-tavlen med inndeling etter medlem.



Figur 6: Visualisering av arbeidsflyt.

å være seks, slik at hvert medlem på gruppen kunne ha to aktive oppgaver i hver kategori.

Tallet seks ble i all hovedsak satt som et grovt utgangspunkt, der grensene kunne justeres gjennom prosessen. For å skape rom for et flertall mindre oppgaver, endte vi med å øke grensen for Selected for Development til ni i stedet for seks. På samme vis reduserte vi det maksimale antallet i Review, da vi anså det hensiktsmessig for arbeidsflyten å raskere gjennomgå ferdigstilte oppgaver.

2.2.2 Fremdrift

Vi delte utviklingsprosessen inn i seks milepæler:

1. **31. januar:** Ferdigstilt forprosjekt
2. **15. februar:** Ferdigstilt vurdering av skyløsning

3. **1. mars:** Første prototype
4. **22. april:** Konkludering av utvikling
   - Ingen ny funksjonalitet blir påbegynt, fokus på finpuss og testing
5. **6. mai:** Ferdigstilt produkt
   - Testing og koding ferdigstilles, fokuset går over på rapportskriving
6. **20. mai:** Ferdigstilt rapport

På hvert sprint planning meeting gjennom prosjektløpet, brukte vi milepælene som målestokk for vår overordnede progresjon. I hver sprint satt vi konkrete mål som jobbet mot de konkrete milepælene. Under er en overordnet oversikt over hva vi jobbet med i hver sprint. I denne oversikten har vi valgt å inkludere en sprint 0, selv om vi på dette tidspunktet i prosessen enda ikke hadde begynt å jobbe i sprinter. Arbeid gjort før starten av denne var i all hovedsak administrative oppgaver relatert til oppstart, og er ikke inkludert i oversikten.

- **Sprint 0:** 23/01 - 06/02
  - Planlegging
  - Forprosjekt
  - Start skyundersøkelse
- **Sprint 1:** 06/02 - 20/02
  - Ferdigstilt skyundersøkelse
  - Start rammeverk
  - Start Grafana
  - Undersøkte teknologier for rapportgenerering
- **Sprint 2:** 20/02 - 06/03
  - Rammeverk for aksessering av data
  - Grafana direkte mot databasen
  - Start simulering med resultater i Grafana
  - Grafana Reporter
- **Sprint 3:** 06/03 - 20/03
  - Start dataflyt gjennom API til Grafana
  - Intervju med salgsavdeling
  - Undersøk videreutvikling av nåværende rapportløsning
  - Start proof of concept rapport
  - Opprette bachelorrapport

  - Utvidelser simulering og rammeverk

- **Sprint 4:** 20/03 - 10/04
  - Ferdigstilt rapport proof of concept med dataflyt gjennom API
  - Ferdigstilt dataflyt gjennom API til Grafana
  - Start heatmap
  - Introduksjonsdelen av bachelorrapport
  - Start klassedelt simulering
  - Utvidelser rammeverk

- **Sprint 5:** 10/04 - 24/04
  - Start hoveddel bachelorrapport
  - Refactor dashbord-kode
  - Ferdigstilt klassedelt simulering
  - Forbedring av rammeverk

- **Sprint 6:** 24/04 - 08/05
  - refactor rapport-kode
  - Ferdigstilt hoveddel bachelorrapport
  - refactor tester
  - Start rapport drøfting

- **Sprint 7:** 08/05 - 20/05
  - Ferdigstilt bachelorrapport

2.2.3 Tidsbruk

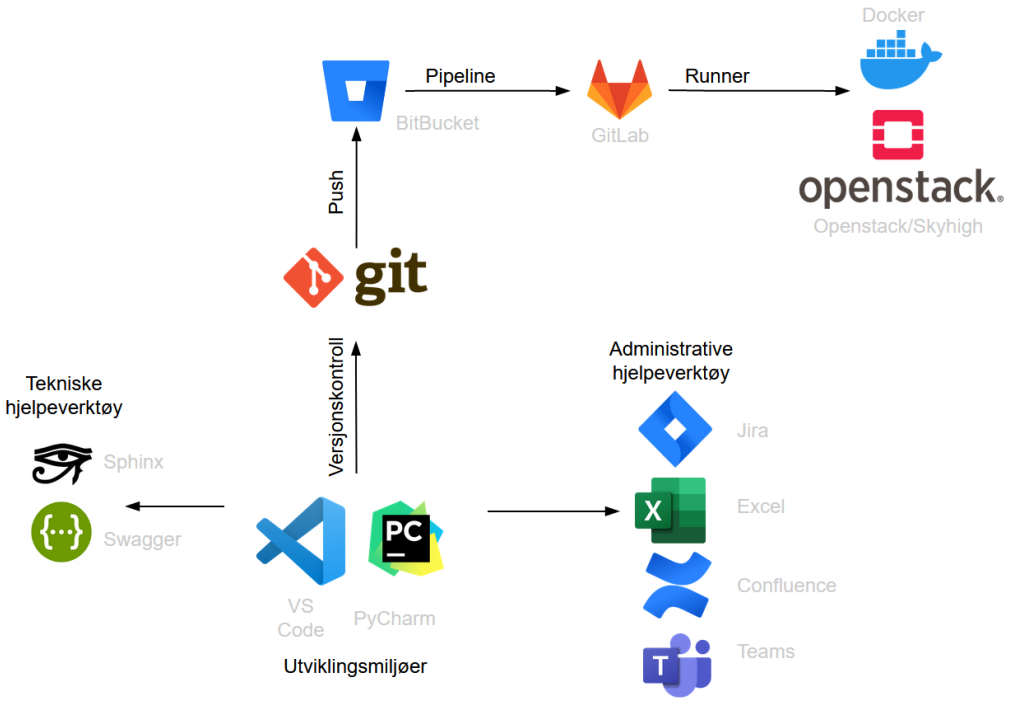
For å reflektere over hvordan tiden ble brukt gjennom prosjektløpet, har vi loggført alle arbeidstimer i Excel. Tiden ble ført per påbegynte halvtime og lunsjpauser ble ikke inkludert. I loggføringen valgte vi å skille mellom arbeid vi gjorde felles, og det som ble gjort på egenhånd. I tillegg brukte vi følgende fem kategorier for å ytterligere gruppere:

- **Administrativt**
  Administrativt arbeid som føring av referater, interne møter eller planlegging.
- **Litteratur**
  Tilegning av kunnskap gjennom litteratursøk. Eksempelvis lesing av akademiske tekster eller tidligere bacheloroppgaver.

- **Programmering**
  Utvikling av sluttproduktet og andre koderelaterte aktiviteter.
- **Rapport**
  Skriving og korrekturlesing av rapport.
- **Møte**
  Eksterne møter med veileder og oppdragsgiver.

2.3 Verktøy

Gjennom prosjektløpet har vi brukt en rekke verktøy. De mest sentrale verktøyene er inkludert i figur 7.



Figur 7: De mest sentrale verktøyene i prosjektet.

Utvikling og versjonskontroll

- **PyCharm** Utvikling i Python.
- **Visual Studio Code** Rapportskriving. og samarbeid.
- **Git** Versjonskontroll.
- **BitBucket** Lagring av kildekode.

- **GitLab** Pipeline.
- **Docker** Testing og pipeline.
- **OpenStack** Intern sky for testmiljø.

Administrative hjelpeverktøy

- **Jira** Styring og visualisering av arbeidsoppgaver.
- **Excel** Loggføring av arbeidstimer.
- **Confluence** Dokumentasjon av arbeid, blant annet møtereferater, løpende dokumentasjon og diagrammer.
- **Teams** Muntlig kommunikasjon mellom gruppen, veileder og oppdragsgiver.

Tekniske hjelpeverktøy

- **Sphinx** Automatisk generering av dokumentasjon av kildekode i HTML-format.
- **Swagger** Dokumentasjon og debugging av endepunktene.

2.4 Risikoanalyse

Under følger vår identifikasjon og analyse av situasjoner som kan oppstå i løpet av prosjektperioden. Disse situasjonene er beskrevet og deretter knyttet opp mot tilhørende risiko og konsekvens. Videre er tiltak formulert. Sannsynlighet og konsekvens er klassifisert etter skala lav, middels og høy.

1. **Tidsfrist overskrides**
   Uforutsette hendelser som tap av arbeid, alvorlig sykdom, tekniske problemer eller mangel på kompetanse kan føre til forsinkelser i prosjektets fremdrift. Sannsynlighet vurderes som middels og konsekvens vurderes som høy.
2. **Overestimert omfang**
   Gruppas manglende erfaring med prosjekter av denne størrelsen, gjør at tidsestimering er vanskelig. Sannsynlighet vurderes som lav og konsekvens vurderes som middels.
3. **Tap av arbeid ved systemkræsj eller uaktsomhet**
   Tap av arbeid i utviklingsprosessen kan i verste fall ha katastrofale ringvirkninger. Sannsynlighet vurderes som lav og konsekvens vurderes som høy.

4. **Mangel på kompetanse**
   Gruppa har lite erfaring med prosjekter av denne størrelsen, og vi anser det som høyst sannsynlig at vi vil støte på arbeidsoppgaver som krever at vi må tilegne oss ny kunnskap. Sannsynlighet vurderes som høy og konsekvens vurderes som middels.

5. **Tap av gruppemedlemmer**
   Alvorlig sykdom, ulykker eller at gruppemedlemmer slutter kan medføre at gruppenss kapasitet blir kraftig redusert. Sannsynlighet vurderes som lav og konsekvens vurderes som høy.

6. **Konflikter innad i gruppen**
   Uenigheter eller andre samarbeidslige faktorer underveis i prosjektløpet kan skape konflikter innad i gruppen. Sannsynlighet og konsekvens vurderes som lav.

7. **Data fra oppdragsgiver kommer på avveie**
   Dersom denne dataen kommer på avveie vil det kunne direkte påvirke kunden samt reflektere negativt på Argos. Sannsynlighet vurderes som lav og konsekvens vurderes som høy.

8. **Oppdragsgiver endrer krav underveis**
   Under utviklingsprosessen er det sannsynlig at enkelte krav endres, fjernes eller at det oppstår nye krav. Sannsynlighet vurderes som høy og konsekvens vurderes som lav.

9. **Mangel på datagrunnlag i simulering**
   Oppdragsgiver har ingen eksisterende løsning innen simulering, og det foreligger derfor mulighet for at datagrunnlaget for å gjøre etterspurte simuleringer ikke er tilstede. Sannsynlighet vurderes som lav og konsekvens vurderes som middels.

		Sannsynlighet		
		Lav	Middels	Høy
Konsekvens	Lav	6	2, 9	3, 5, 7
	Middels			1
	Høy	8	4	

Tabell 1: Risikovurdering for prosjektarbeidet.

Fra risikovurderingen (se tabell 1) peker risiko 1 og 4 seg ut som høy risiko, og krever derfor mitigerende tiltak. Disse tiltakene beskrives i tabell 2.

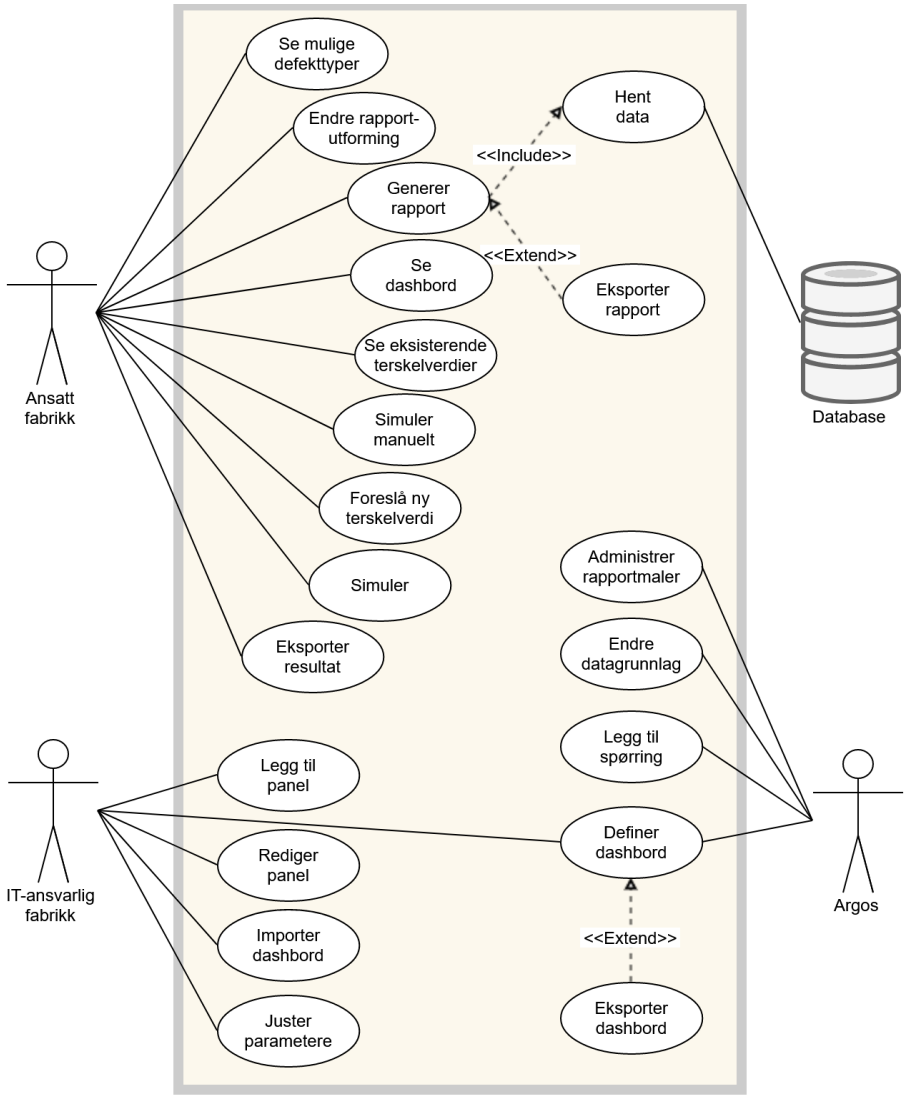
#	Risiko	Tiltak
1	Tidsfrist overskrides	Stort fokus på planlegging, både initielt, men også gjennom prosjektløpet ved retrospektive møter. Her vil satte milepæler bli brukt for å måle progresjon. Hyppige møter (2.2.2) med veileder og oppdragsgiver vil også gjennomføres for ekstern tilbakemelding.
4	Mangel på kompetanse	Gruppa vil samarbeide tett gjennom hele prosjektløpet. Der tilsvarende verktøy vurderes, vil vi også betrakte egen kompetanse og verktøyenes læringskurver.

Tabell 2: Tiltak med tilhørende beskrivelse.

3 Kravspesifikasjon

Gjennom intervjuer og tett kommunikasjon med oppdragsgiver, har vi i prosjektperioden kontinuerlig tilegnet oss informasjon som har formet kravene for vårt system. Det mest naturlige ville vært å presentere kravene på samme vis, men for å gjøre det enklere for leseren valgte vi her å presentere den endelige kravspesifikasjonen, som om vi hadde arbeidet etter fossefallsmetoden.

3.1 Use case



Figur 8: Use case diagram.

3.1.1 Høynivå use case

Tittel: Generer rapport **Aktør:** Ansatt fabrikk **Mål:** Generere rapporten **Beskrivelse:** Brukeren kan generere rapport med nødvendige parametere. Dersom parametere mangler eller er ugyldige, vil brukeren få beskjed om det.

Tabell 3: Use case: Generer rapport.

Tittel: Eksporter rapport **Aktør:** Ansatt fabrikk **Mål:** Eksportere rapporten i PDF-format. **Beskrivelse:** Brukeren kan velge å eksportere rapporten i PDF-format.

Tabell 4: Use case: Eksporter rapport.

Tittel: Hent data **Aktør:** Ansatt fabrikk, ≪Generer rapport≫ **Mål:** Hente nødvendige data fra databasen **Beskrivelse:** Dersom brukeren har valgt å generere en rapport, vil data hentes fra databasen. Hvilke data som hentes spesifiseres i rapporten.

Tabell 5: Use case: Hent data.

Tittel: Endre utformingen av rapport **Aktør:** Ansatt fabrikk **Mål:** Endre på den visuelle utformingen av rapporten **Beskrivelse:** Bruker kan endre på den visuelle utformingen av rapporten.

Tabell 6: Use case: Endre utformingen av rapport.

Tittel: Administrer rapportmaler **Aktør:** Argos, IT-ansvarlig fabrikk **Mål:** Lage og endre rapportmaler **Beskrivelse:** Administrator kan spesifisere nye rapporter og endre på nåværende dersom det skulle være ønskelig.

Tabell 7: Use case: Administrer rapportmaler.

Tittel: Endre datagrunnlag **Aktør:** Argos, IT-ansvarlig fabrikk **Mål:** Legge inn eller endre datagrunnlag **Beskrivelse:** IT-ansvarlig fabrikk kan legge inn nye datagrunnlag i eksisterende rapporter eller endre på nåværende.

Tabell 8: Use case: Endre datagrunnlag.

Navn: Se dashbord **Aktør:** Ansatt fabrikk **Mål:** Skaffe oversikt over produksjonen **Beskrivelse:** Fabrikkansatt velger et dashbord og får oversikt over produksjonen i sanntid.

Tabell 9: Use case: Se dashbord.

Navn: Rediger panel **Aktør:** IT-ansvarlig fabrikk **Mål:** Legge til, fjerne eller endre panel i dashbord **Beskrivelse:** I dashbordet skal paneler kunne modifiseres, for eksempel å endre på den visuelle fremvisningen og hvilke data som skal visualiseres. Det er også mulig å legge til eller fjerne paneler.

Tabell 10: Use case: Rediger panel.

Navn: Juster parametre **Aktør:** IT-ansvarlig fabrikk **Mål:** Få tilgang og mulighet til å justere parametre **Beskrivelse:** Det skal være mulig for IT-ansvarlig å endre på tekniske parametre. Hvilke skannere som skal brukes, hvilket tidsrom man ønsker å visualisere eller hvor ofte dashbordet skal oppdateres.

Tabell 11: Use case: Juster parametre.

Navn: Importer dashbord **Aktør:** IT-ansvarlig fabrikk **Mål:** Innføre nye maler til dashbord **Beskrivelse:** IT-ansvarlig i fabrikk skal kunne importere eksisterende dashbord.

Tabell 12: Use case: Importer dashbord.

Navn: Definer dashbord **Aktør:** Argos, IT-ansvalig fabrikk **Mål:** Definere nye dashbord **Beskrivelse:** Det skal være mulig å lage nye dashbord.

Tabell 13: Use case: Definer dashbord.

Navn: Eksporter dashbord **Aktør:** Argos, IT-ansvalig fabrikk **Mål:** Eksportere dashbord **Beskrivelse:** Når et dashbord er definert, så skal det være mulig å eksporteres for bruk i andre løsninger.

Tabell 14: Use case: Eksporter dashbord.

Navn: Legg til spørring **Aktør:** Argos **Mål:** Legge til ny spørring **Beskrivelse:** Argos legger til nye spørringer for å hente data fra databasen til dashbord.

Tabell 15: Use case: Legg til spørring.

Tittel: Se mulige defekttyper **Aktør:** Ansatt fabrikk **Mål:** Se hvilke defekttyper det er mulig å simulere på **Beskrivelse:** Brukeren skal kunne se hvilke defekt typer og subtyper som er aktuelle å simulere på

Tabell 16: Use case: Se mulige defekttyper.

Tittel: Se eksisterende terskelverdier **Aktør:** Ansatt fabrikk **Mål:** Se hvilke terskelverdier som er satt. **Beskrivelse:** For hver defekttype skal brukeren kunne se hvilke terskevlerdier som allerede er satt.

Tabell 17: Use case: Se eksisterende terskelverdier.

Tittel: Simuler manuelt **Aktør:** Ansatt fabrikk **Mål:** Se utfallet av simulering med andre terskelverdier. **Beskrivelse:** Brukeren skal kunne se hvordan produksjonen hadde endret seg dersom terskelverdien hadde vært endret. Brukeren skal se antall nye plater i hver klasse for hver defekttype. Brukeren bør kunne se en visuell representasjon av simuleringen gjennom en graf.

Tabell 18: Use case: Simuler manuelt.

Tittel: Foreslå ny terskelverdi **Aktør:** Ansatt fabrikk **Mål:** Gi brukeren et forslag til terskelverdi. **Beskrivelse:** Brukeren ønsker å få foreslått en terskelverdi som utgangspunkt for videre simulering.

Tabell 19: Use case: Foreslå ny terskelverdi.

Tittel: Simuler **Aktør:** Ansatt fabrikk **Mål:** Gjennomfør selve simuleringen **Beskrivelse:** Systemet gjennomfører selve simuleringen basert på inndata fra brukeren.

Tabell 20: Use case: Simuler.

Tittel: Eksporter resultater **Aktør:** Ansatt fabrikk **Mål:** Eksportere resultatet av simuleringen. **Beskrivelse:** Det bør være mulig å eksportere resultatet av simuleringen slik at det kan lagres for fremtidig bruk.

Tabell 21: Use case: Eksporter resultater.

3.2 Lavnivå use case

Tittel: Generer rapport
Aktør: Ansatt fabrikk
Mål: Generere rapporten
Forutsetning: Rapport som forsøkes generert eksisterer
Ettervirkning: Rapporten genereres
Normal flyt:

1. Bruker spesifiserer rapport-parametere.
2. Bruker velger å generere rapport.
3. Systemet henter data fra databasen.
4. Rapporten populeres med dataen.
5. Rapporten vises.

Alternativ flyt:
Normal flyt til og med punkt 2:

- Bruker velger ugyldige parametere. Bruker får beskjed om at ugyldige parametre har blitt oppgitt og gis dermed muligheten til å oppgi nye.

Feilsituasjoner:

- Systemet får ikke kontakt med databasen.

Tabell 22: Lavnivå use case for å generere rapport.

Tittel: Simuler
Aktør: Ansatt fabrikk
Mål: Simulere endring i utfallet av historisk produksjon
Forutsetning: Det finnes et datagrunnlag å simulere på i databasen, systemet er startet.
Ettervirkning: Brukeren ser resultatet av simuleringen
Normal flyt:

1. Bruker spesifiserer tidspunkt og skanner som skal benyttes.
2. Bruker velger hvilke defekter som skal simuleres på.
3. Bruker velger nye terskelverdier for defektene.
4. Systemet simulerer hvordan graderingene til platene endrer seg med de nye terskelverdiene. Systemet skal ikke regne med plater som ikke ville fått ny klasse dersom man ser bort ifra gammel klasse, da dette ikke vil gi en reell produksjonsendring.
5. Systemet returnerer en tabell over defekttypen, ny terskelverdi og produksjonsendring.

Alternativ flyt:

3b. Hvis systemet skal foreslå data, foreslår systemet heller ny terskelverdi basert på produksjonsendring.

4b. Dersom brukeren oppgir feil data, returneres en feilmelding.

Feilsituasjoner:

1. Systemet får ikke kontakt med databasen.
2. Det finnes ingen plater i angitt tidsrom

Tabell 23: Lavnivå use case for å simulering.

3.3 Ikke-funksjonelle krav

Ytelse

Systemet bør skrives på en slik måte at den ikke tar beslag på mye av kapasiteten til databasen.

- Resultat av simuleringen skal vises på skjerm innen 30 sekunder.
- Dashbordet skal oppdateres innen 10 sekunder.

Brukergrensesnitt

- Dashbord må utformes slik at brukeren skal kunne få en helhetlig oversikt over produksjonen innen 10 sekunder.
- Brukergrensesnittet bør utformes slik at det kan brukes av en person som ikke har detaljkunnskap om hvordan databasestrukturen er bygget opp.

Sikkerhet

- Bruker vil kunne oppgi data for å påvirke spørringer mot databasen. Systemet skal sikres mot angrep som SQL-injeksjoner.

Vedlikehold

- Kildekoden og konfigurasjonsfiler som leveres må dokumenteres på en måte slik at ansatte hos Argos kan få en innføring i systemet uten å manuelt grave i kildekoden. Dette innebærer at alle ikke-trivielle funksjoner skal dokumenteres og at logiske modulers inn- og utdatastruktur dokumenteres.
- Koden bør skrives på en slik måte at funksjonalitet kan gjenbrukes i fremtiden. Det bør også være enkelt å legge til ytterligere funksjonalitet ved et senere tidspunkt.

Plattform

- Systemet skal legge opp til å enkelt kunne flyttes mellom Windows og Linux.

3.4 Krav til innhold i dashbord

Tabell 9 beskriver at brukeren skal kunne se dashbord. For å imøtekomme dette kravet er det nødvendig med ytterlige informasjon om hva slags informasjon som må vises. Dette ble avdekket gjennom tett oppfølgning med oppdragsgiver og diskusjon internt i gruppen. Det ble i tillegg gjennomført et intervju med flere salgsrepresentanter fra selskapet, for å kartlegge kundenes behov. Resultatet av denne prosessen kan sees i tabell 24.

Produksjonsmål	**Beskrivelse**
Antall produserte plater	Se hvor mange plater som er produsert, inndelt etter gradering. Det må være mulig å se både øyeblikksbilde og trend. Dataen bør kunne vises i prosent.
Heatmap	Det skal være mulig å se et heatmap som viser defektenes lokasjon på platen. Det skal komme tydelig frem om visse områder har høy konsentrasjon av feil.
Forekomster av defekttyper	Vise forekomster av defekttype som har nedgradert plater. Det bør være mulig å begrense dette til eksempelvis topp 3. Det bør være mulig å se både øyeblikksbilde og trend.
Hente produktinformasjon	Hente informasjon om produktet, når ble det laget, tilhørende serie og annen relevant produktinformasjon.
Linjehastighet	Se hastigheten på produksjonen i dashbordet. Det bør tilrettelegges for å legge inn mål for hastighet.

Tabell 24: Krav til innhold i dashbord.

4 Teknologier

Oppgaven fra oppdragsgiver la ingen føringer for valg av teknologi eller arkitektur. Dette medførte at mye tid ble brukt på å finne de teknologiene som passet med kravene til systemet, avveiet av hvor mye vi kjente til de fra før og hvor enkle de var å anvende. Da dette prosjektet var stort og ukjent for oss, var det viktig for oss å velge teknologier som la til rette for for rask prototyping.

4.1 Programmeringsspråk

Et av de første valgene vi tok var hvilket programmeringsspråk vi skulle bruke. Valget la grunnlaget for andre viktige teknologivalg senere. Vi stilte følgende kriterier til språket:

- **Støtte for tredjepartsbiblioteker** For å tilrettelegge for rask prototyping var det viktig at språket hadde mange tredjepartsbiblioteker vi kunne benytte oss av.
- **Kompabilitet på tvers av platformer** Argos ønsker at systemet skal kunne flyttes over på andre plattformer uten større arbeid, for eksempel til sky.
- **Tidligere erfaringer** Dersom valget falt på et språk som gruppen ikke var kjent med, ville det medført ekstra tid til å lære språket. Det var også en fordel om Argos allerede hadde kjennskap til språket.
- **Sammenkobling med Argos' eksisterende teknologier** Det hadde vært en fordel om språket integrerte med Argos' eksisterende systemer.

Alternativene vi undersøkte var C++, Python og Java. Vi tok utgangspunkt i disse alternativene da gruppen har mer eller mindre kjennskap til disse fra før. Selv om det finnes andre programmeringsspråk som kunne vært relevante, for eksempel Go eller Rust, fant vi det ikke hensiktsmessig å lære oss disse med mindre det ga store fordeler for prosjektet vårt.

Fordelene med C++ var at Argos allerede hadde mye kjennskap til dette språket i bedriften. Dette gjør at systemet vårt enklere kunne integreres i Argos' kjerneplattform dersom dette skulle være ønskelig. Språket eksekverer raskt, som kan være en fordel dersom prosjektet vil innebære tunge algoritmer. Ulempene med språket var at gruppen har minst kjennskap på området av de tre overnevnte. C++ biblioteker kan variere mellom operativsystemer, noe som kan gjøre det vanskeligere å flytte systemet til en annen plattform ved behov. På grunn av denne kompleksiteten, er språket også lite egnet for rask prototyping.

Java er et språk som konsekvent scorer høyt på de fleste programmeringsindekser [8, 9, 10, 11]. Språket er modulært og gir gode muligheter for ekspansjon med

sin objektorienterte struktur. Det finnes mange utvidelser og rammeverk som gjør språket meget allsidig. Java er plattformuavhengig med sin Java Virtual Machine (JVM), som gjør systemet uavhengig av operativsystemet det kjøres på. Ulempene vi fant med Java var at gruppen hadde noe kunnskap om det, men at mer måtte læres for å bruke det effektivt. Språket brukes heller ikke av Argos idag, noe som vil medføre ekstra ressurser fra de ansattes side for å lære seg språket etter at bachelorprosjektet er over.

Det siste alternativet vi undersøkte var Python. Python er et språk som er velkjent for å være enkelt å lære seg og å forstå for personer uten tidligere erfaring [12]. Språket har bred støtte gjennom utvidelser og rammeverk som gir en bruker rask mulighet til å benytte språket for ellers komplisert funksjonalitet. Populære eksempler på dette er biblioteker som OpenCV for bildebehandling og PyTorch for maskinlæring. Språket er objektorientert i samme stil som Java, med mange av de samme fordelene. Det finnes Python tolkere for alle de store operativsystemene, som i praksis gjør den plattformuavhengig. I motsetning til C++ og Java som kompileres, fungerer Python heller ved at koden tolkes linje for linje når den eksekveres. Dette medfører betydelig redusert ytelse ved prosessorintensive algoritmer [13]. Python har ikke statiske variabeltyper som Java og C++, noe som åpner for flere problemer ved eksekvering som ellers kan detekteres av en kompilator på forhånd.

Basert på disse fordelene og ulempene valgte vi å gå for Python. Python og Java dekket alle de tekniske kriteriene vi satt for programmeringsspråk. På det rent tekniske så vi at Java muligens hadde vært det beste valget, men helhetlig falt vi til slutt på Python, da våre erfaringer og Argos' behov for vedlikehold i etterkant overveide fordelene til Java. Muligheten for rask prototyping var også en betydelig faktor.

4.2 API-arkitektur

Systemet vårt legger opp til å kjøre lokalt og i skyen. Vi så også umiddelbart at vi ville komme til å sende og motta data fra flere forskjellige eksterne systemer. Dette gjorde at en HyperText Transfer Protocol (HTTP)-basert arkitektur virket mest aktuell.

Det finnes to hovedalternativer for API-baserte løsninger for web: REpresentational State Transfer (REST) og Graph Query Language (GraphQL). REST er eldst, og dermed mest veletablert. REST kan regnes som en utvidelse av HTTP ved at den kan ta argumenter enten i URL-en direkte, gjennom http-form eller som et JavaScript Object Notation (JSON)-objekt. Det helhetlige API-et deles opp i mindre endepunkt som returnerer ønsket data. REST kan av denne grunn brukes av alle tjenester som støtter HTTP-oppslag. GraphQL er en nyere standard som forsøker å ha bedre ytelse enn REST [14], ved at man gjør spørringer (derav navnet Query Language) mot ett endepunkt som henter ut

nøyaktig den dataen man ønsker. Ettersom GraphQL er spørringsbasert vil det kreve at klienter har konkret støtte for det.

Vi valgte å fokusere på REST da dette er en enklere protokoll å benytte. Det ga større sannsynlighet for at flere eksterne systemer kunne bruke API-et, da langt flere systemer støtter HTTP-oppslag enn GraphQL-spørringer.

4.3 Python rammeverk

Som nevnt tidligere er bredden av tilgjengelige tredjepartsbiblioteker en av de største styrkene for Python. Her vil vi gå gjennom noen av de mest sentrale bibliotekene vi har utforsket og benyttet.

4.3.1 SQL rammeverk

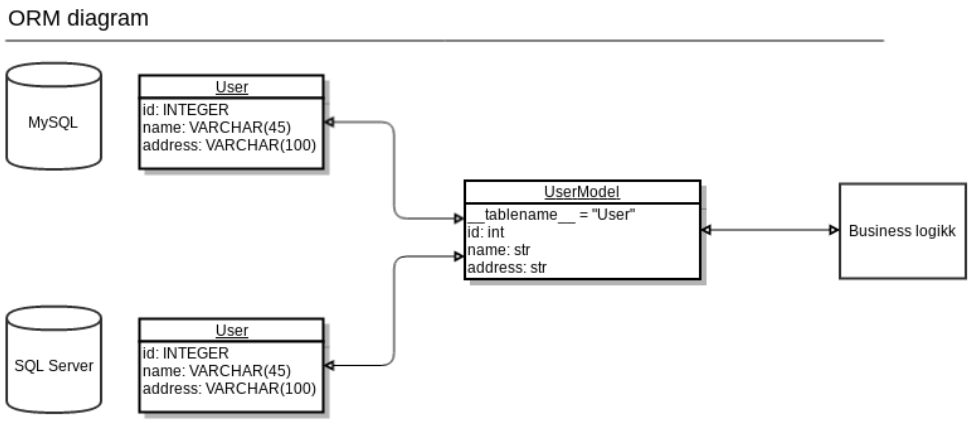
En ramme vi måtte forholde oss til var at eksisterende data ligger i en instans av SQL Server 2012. Det finnes to Python rammeverk for dette: pyodbc og pymssql [15]. Microsoft prioriterer å støtte pyodbc, så vi valgte å gå for dette alternativet.

På toppen av pyodbc var gruppen allerede kjent med en SQL-verktøykasse som heter SQLAlchemy [16]. SQLAlchemy tilbyr et standardisert grensesnitt for å generere spørringer, uavhengig av underliggende database. Denne modellen utheves spesielt i Object Relational Mapping (ORM) modulen. ORM er en implementasjon av data mapper designmodellen. Denne modellen oppretter et direkte forhold mellom en datakilde og objektorienterte klasser. Når disse klassene brukes i koden, fjernes avhengigheter mellom bruk av dataen og kilden. Denne abstraheringen gir oss en stor fordel ved at det for eksempel er enklere å bytte database i bakgrunnen, uten at større endringer kreves i koden. I figur 9 kan man se et eksempel på dette.

4.3.2 API-rammeverk

Det finnes i hovedsak tre alternativer for å implementere API-rammeverk i Python: Django, Flask og FastAPI. Vi vurderte disse rammeverkene utifra følgende hovedkriterier:

Funksjonalitet, eller hvor mye funksjonalitet er bygd inn/tilgjengelig fra tredjepartsbiblioteker. Her er Django sterkest, med sitt fokus på å tilby utviklere en rask måte å sette opp en fullstendig web-stack [17]. FastAPI kommer som nummer to med sine brede innebygde verktøy spesifikt for API utvikling. Eksempler her er Swagger, Redoc, OAuth 1/2 og OpenAPI [18]. Flask kommer sist da



Figur 9: Eksempel på ORM struktur.

hovedfokuset til flask er å tilby det absolutt minste som trengs for å kjøre en web-tjener [19], med tredjepartsbiblioteker for ytterligere funksjonalitet.

Ytelse, eller hvor raskt gjennomføres prosesseringen av spørringene. FastAPI kommer tydelig ut som best i denne kategorien. FastAPI bygger på Starlette, som igjen bygger på Uvicorn. Uvicorn er regnet av blant annet Techempower for å være en av de raskeste webtjenerene for Python. I samme test fikk FastAPI 1209 poeng, Flask fikk 468 og Django fikk 280 [20].

Kompleksitet, eller hvor enkelt eller vanskelig det er å benytte seg av funksjonaliteten, både på enkelt og avansert nivå. Her kommer FastAPI best ut hovedsakelig på grunn av sin kjernefilosofi om at bruk skal være enkel og intuitiv. Flask, derimot, er bygget på at utvikleren må implementere ekstrafunksjonalitet selv.

Utifra disse kriteriene bestemte vi oss for at FastAPI passer best for analyseplattformens formål. Vi trenger bare funksjonalitet for API-endepunkter, i tillegg til lett tjening av statiske filer. Django vil derfor tilby mye unødvendig funksjonalitet. FastAPI kommer også med funksjonalitet for dokumentasjon og typesjekking, som vi anså som nyttig. FastAPI regnes også som raskere enn alternativene, noe som er en fordel i skybaserte miljøer der ytelse direkte påvirker kostnad.

4.4 Dashbordvisualisering

Grafana ble valgt som verktøy for visualisering av data. Oppgavebeskrivelsen satt ikke konkrete krav til valg av teknologi, men valget ble heller gjort på bakgrunn

av de funksjonalle kravene.

Det var viktig for Argos at dashbordet kunne opereres av ikke-teknisk personell, og at det var enkelt å justere visualisering av data. Dette på grunn av deres tidligere erfaringer med variende preferanser hos kundene. På dette kriteriet stilte Grafana seg sterkt med sitt store utvalg av visualiseringsmetoder. Ettersom Grafana er open source, utvikles det også kontinuerlig nye metoder for visualisering, hvor brukere av systemet kan bidra. Grafana muliggjør hurtig tilpasning av dashbord for brukeren gjennom dra og slipp. Dette gjør det enkelt å tilpasse for ulike visningsformer, som for eksempel tv, dataskjermer eller mobile enheter. Det er enkelt å justere data som vises ved å klikke inn på det aktuelle panelet.

Grafana støtter en rekke datakilder som øker fleksibiliten om hvor dataen kan hentes fra. Det tilrettela for rask prototyping, da vi kunne koble direkte på SQL-databasen. I tillegg var Grafana open source med Apache 2.0 [21] lisens, som gjorde at programvaren kunne anvendes kommersielt.

Det var også mulig å lagre resultatet fra API-et i tidsseriedatabaser som Prometheus [22] eller Graphite [23], istedenfor å sende det direkte til dashbordløsningen. Fordelen med dette ville vært å kunne lagre den aggregerte dataen over lengre tid. Dette vil ikke være mulig med nåværende databasestruktur, da lagringsplass er begrenset. Gruppen kom etterhvert frem til at dette var unødvendig på bakgrunn av at vi ikke lengre skulle bruke Grafana til å lage rapporter. I tillegg var det ikke ønsket å innføre ytterlige kompleksitet i arktitekturen.

4.5 Rapportgenerering

Gjennom prosjektløpet vurderte gruppen flere verktøy for generering av rapport. Vi satt følgende krav til verktøy:

- Tilfredstille oppgavebeskrivelsen.
- Mulighet for å oppnå tilnærmet den samme funksjonaliteten som ved den eksisterende løsningen til Argos.
- Enkelt å gjøre endringer i visuell utforming av rapportene.

Videreutvikling av nåværende løsning

Eksisterende løsning hos Argos for å generere rapporter er fungerende, men problematikken ligger i at den bakomliggende strukturen er kompleks og utdatert, som gjør det vanskelig å gjøre endringer i eller legge til nye rapportmaler.

Da det eksisterende systemet tok i mot `.RDLC`-filer, var det naturlige første steget å forsøke å gjøre endringer gjennom det tilhørende verktøyet Microsoft Report

Builder. Gjennom Report Builder var det relativt enkelt å gjennomføre endringer i rapportmalene, men problematikken oppsto ved gjeninnføring av malene i systemet. Det nåværende systemet til Argos ble utviklet i 2008, og følger et utdatert XML-skjema fra samme år. Filstrukturen i rapportmalene som genereres av nåværende versjon av Report Builder (2019) er fra 2016 og var derfor ikke kompatibel med nåværende system.

Etter grundige undersøkelser og diskusjon med veileder, konkluderte vi med at det ikke ville være hensiktsmessig å bruke mer tid på dette i et forsøk på å rekonstruere gammel dokumentasjon.

Implementasjon av eget verktøy

Vi vurderte tidlig i prosessen muligheten for å implementere et egenlaget verktøy. På denne måten kunne vi skreddersy funksjonalitet etter behov, og dermed oppfylle alle kravene vi hadde satt. Dette alternativet ville gitt det høyeste læringsutbyttet, men vi anså tiden og kunnskapen dette ville kreve som usannsynlig. På grunnlag av dette la vi raskt fra oss denne ideen.

Grafana Reporter

Utviklingen av modulene foregikk parallelt, og Grafana ble tidlig i prosessen ansett som aktuelt for bruk som dashbordløsning. På bakgrunn av dette var det naturlig å undersøke om vi kunne bruke dette verktøyet for å tilfredstille flere deler av oppgaven.

Tanken var å lage et sekundært dashbord som kunne fungere som en historisk versjon av sanntidsstatistikken, og dermed generere en rapport ut fra dette. Grafana har allerede et verkøy inkludert i sin betalte løsning for å gjøre nettopp dette - Grafana Reporting. Dette verktøyet er fortsatt under utvikling, og for å aksessere og bruke verktøyet måtte Grafana Cloud benyttes [24]. Vår testing i Grafana foregikk ved bruk av Docker, og det var derfor ikke mulig for oss å benytte dette verktøyet - selv med tilgang til den utvidede funksjonaliteten.

For å likevel teste og undersøke muligheten for bruk av denne løsningen, fant vi at tilsvarende funksjonalitet var tilgjengelig gjennom open source verktøyet Grafana Reporter [25]. Grafana Reporter er en web-tjeneste som genererer rapporter i PDF-format ut fra dashbordene i Grafana, hvor grafer og tabeller inkluderes i rapportene som bilder (se vedlegg J). LaTeX lå til grunn for disse rapportene, noe som gruppen var kjent med fra tidligere. Endringene som kunne gjøres i LaTeX var begrenset, da store deler av rapporten var bilder av grafene fra dashbordet. Dette medførte at fleksibiliteten for å gjøre endringer i rapportstrukturen var lav. Dette anså vi som en stor ulempe, da det ved flere anledninger hadde blitt gjort tydelig fra oppdragsgiver at det var ønskelig at kunden skulle kunne endre på

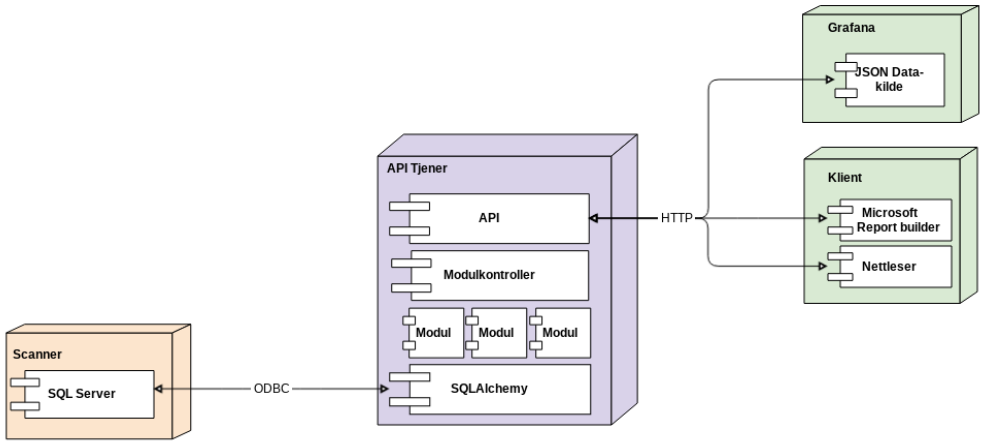
den generelle utformingen av rapporten, basert på preferansene til den enkelte. I tillegg var funksjonalitet som gruppering av rapporten ikke mulig med Grafana Reporter. Overordnet sett fungerte verktøyet godt for å eksportere grafer til bruk ved presentasjoner og lignende, men som et verktøy for å generere rapporter hadde det for store mangler til at vi vurderte det i større grad.

Microsoft Report Builder

Report Builder er et Microsoft verktøy som brukes for å definere og redigere rapportmaler. I malene er visuell utforming, hvilke data som hentes og datakilder spesifisert. Det var enkelt å gjøre endringer, og verktøyet legger også til rette for eksportering i en rekke formater. På bakgrunn av dette valgte vi å gå for Report Builder, da dette verktøyet tilfredstilte kravene vi hadde satt.

5 Design

Arkitekturen til systemet består i hovedtrekk av en sentral tjener som tar imot forespørsler fra eksterne systemer, og henter data fra angitte datakilder. Figur 10 viser hvordan den sentrale tjeneren er delt i en lagdelt modell, med en modulbasert arkitektur for definisjon av funksjonalitet.



Figur 10: Overordnet arkitektur.

5.1 Overordnet arkitektur

Oppgaven fra Argos er delt i tre forskjellige hoveddeler: Rapportgenerering, sanntidsdata og simuleringsverktøy. Vi innså raskt at det vil være lite hensiktsmessig å lage disse verktøyene fra bunn. Dermed fokuserte vi tidlig på å bruke eksisterende programvare for å utføre de mest kompliserte delene, samt bygge en egen plattform som kan binde disse systemene sammen. Argos' ønske for analyseplattformen var at den skulle fungere som et bindeledd mellom de eksisterende databasene og verktøy som benytter seg av disse. Dette gjorde at systemet måtte ha en arkitektur som la opp til enkel utvidelse i fremtiden, spesielt i form av nye eksterne verktøy.

Hovedmodellen i fellesplattformen baserte vi på en lagdelt modell. En lagdelt modell tilfredsstiller kravene til analyseplattformen på grunn av sine lave koblinger og høye kohesjon. I denne modellen kan hvert lag sømløst byttes ut, så lenge grensesnittet forblir det samme. Denne fleksibiliteten legger også opp til å enkelt introdusere ny funksjonalitet i hvert lag. Fleksibilitet i utrullingen var også en faktor som måtte tas i betraktning. Lave koblinger og høy kohesjon gir også en mer glidende overgang til distribuert arkitektur, der mindre deler av systemet kan flyttes til separate prosesser eller tjenere etter behov. Dette styrker analyseplattformens mulighet til å kjøre i et skybasert miljø.

5.2 Lagene i arkitekturen

Figur 10 viser hvordan fellesplattformen er delt inn i fire hovedlag: API, modulkontroller, moduler og databasetilgang. Her skal vi gå mer inn i hva disse lagene må innebære for å oppfylle kravene til plattformen.

5.2.1 API

API-laget håndterer hvordan eksterne tjenester skal strukturere spørringer til resten av systemet. I valg av teknologi (4.2) ble det tidlig bestemt at grensesnittet til eksterne tjenere skulle være REST-basert, da dette tilbyr størst fleksibilitet.

Grafana og Report Builder har forskjellige strukturer for hvordan de spør etter data, og hva de forventer at formatet på returnert data skal være. Grafana sender parametrene til sine spørringer gjennom JSON-objekter, og forventer svar i samme format. Report Builder derimot, benytter seg av `x-www-form-urlencoded` enkoding[2], og forventer XML-strukturert data som respons. Disse kravene gjør at API-laget må kunne tjene disse formatene samtidig.

Siden dette laget håndterer alle forespørsler til eksterne tjenester har den rollen som førstelinjeforsvar mot sikkerhetstrusler. API-laget håndterer både innkommende og returnerte data, og har derfor en spesielt viktig rolle i å validere dataen i begge retninger.

5.2.2 Modulkontroller

Det vil være lite formålstjenelig å prøve å lage generisk funksjonalitet som dekker alle behov, og det vil være umulig å lage all mulig ønsket funksjonalitet initielt. Disse rammene gjorde at et smidig, fleksibelt rammeverk som er enkelt å videreutvikle var nødvendig.

Av de fem mest brukte arkitekturmodellene for programvare [26] lener de overnevnte kravene seg mot en mikrokjernebasert modell. Denne type modell består av en kjerne med grunnleggende funksjonalitet, og selvstendige komponenter som enkelt kan legges til eller fjernes. Modellen er regnet som en smidig modell som gir gode vilkår for utrulling, testing og ytelse. Modellen er svakere dersom man ønsker å skalere opp hele tjenesten, og den gjør utvikling noe mer kompleks avhengig av hvor mange krav til kjerne man har [26].

I analyseplattformen vil kjernen bestå av nødvendig logikk for å tolke parametre som alle moduler vil trenge, oppslag av modul for å gi spørringen videre til riktig sted og håndtering av datastrukturen som returneres. I figur 10 utgjør API- og

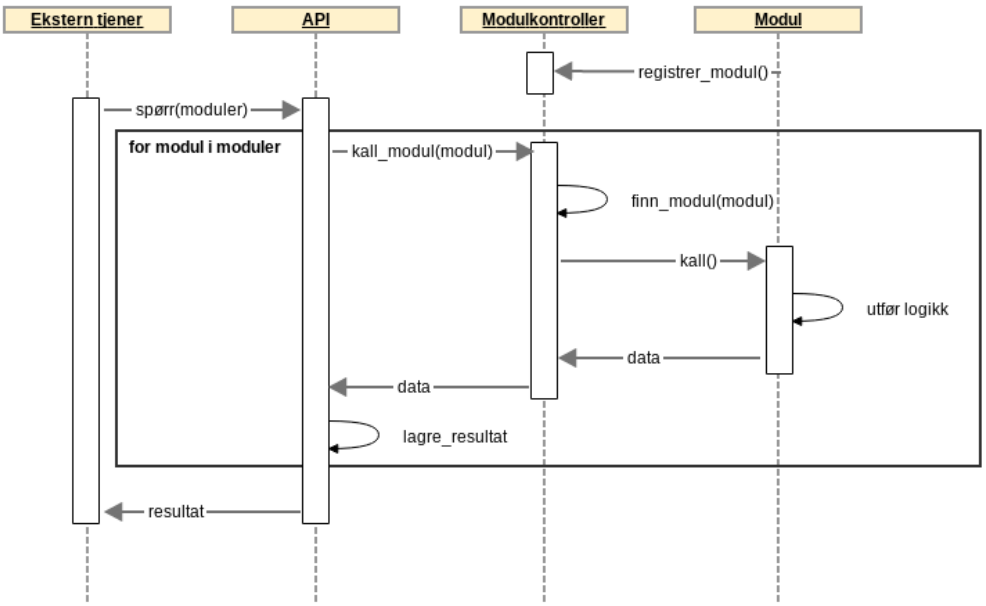
[2] Eksempelvis `param1=a&param2=b` mot JSON sitt `{'param1':'a', 'param2:'b'}`

modulkontrollerlagene kjernen, mens modulene er de selvstendige komponentene. Styrkene til modellen gjør at modulene selvstendig blir enklere å utvikle, da alle blir kalt med de et forhåndsdefinert sett med parametre. Det er også lett å velge ut nøyaktig hvilken funksjonalitet som rulles ut gjennom å velge hvilke moduler som inkluderes. Modellen åpner også for at moduler kan lastes fra forskjellige steder, for eksempel innebygd i kjerneplattformen eller distribuert over forskjellige systemer

Ved utvikling av en mikrokjernebasert modell er det viktig å tidlig definere hvordan grensesnittet mellom kjernen og modulene skal være [26]. Etter undersøkelser innen rapportgenerering og dashbord har vi kommet frem til at alle moduler vil trenge følgende parametre:

- **Datakilde**: Det vil være nødvendig for alle plugins å vite hvor de skal hente data fra.
- **Scannernavn**: For å kunne navngi den resulterende dataen trengs scannernavnet som sesjonen hører til.
- **Tidspunkter**: Alle data hører til innenfor et gitt tidsintervall.
- **Annen data**: Hver plugin kan ha behov for ytterligere data enn de andre oppgitte punktene. Derfor må det også være mulig å tilby dette.

Dataflyten gjennom kjernen er vist i figur 11.



Figur 11: Sekvensdiagram for kjernen.

5.2.3 Databasetilgang

Det nederste laget i figur 10 er datatilgangslaget. Et datatilgangslag bør inneholde funksjonalitet for tilkobling til databaser, sesjonshåndtering, Create, Read, Update, Delete (CRUD)-operasjoner og tjenestefrihet [27]. Ved at datatilgangslaget tar seg av dette vil modulene bare trenge å forholde seg til et predefinert grensesnitt, uten at de trenger å vite hvor dataen kommer fra selv.

I kapittelet om teknologi ble det bestemt at systemet skulle bruke SQLAlchemy for å gjøre spørringer mot databaser. SQLAlchemy kunne ivareta alle de overnevnte kravene til et databasetilgangslag, men hovedsakelig bare for individuelle tilkoblinger. Databasetilgangslaget i analyseplattformen består dermed av et rammeverk som utvider SQLAlchemy sin funksjonalitet gjennom å håndtere hvordan spørringene kjøres mot flere mulige databaser.

5.3 Simulering

Simuleringsdelen av analyseplattformen skal, i følge oppgavebeskrivelsen og kravene som tidligere definert, bestå av et verktøy for å simulere hvordan det historiske produksjonsresultatet hadde sett ut dersom utvalgte vurderingsparametre[3] hadde vært annerledes. Dette medfører at det vil være nødvendig for brukeren av verktøyet å kunne skrive inn hvilke parametre som ønskes å endres på, samt se utfallet av simuleringen for å optimalisere produksjonen.

Fra møtet med selgerene i Argos kom det frem at den aktuelle målgruppen for dette verktøyet ikke kjenner til hvordan Argos' interne databasestruktur er utformet. Vi måtte dermed ta hensyn til å bare benytte begreper som brukeren allerede kjenner til. Det ble også nevnt i møtet at brukerne selv med dette verktøyet mest sannsynlig vil prøve og feile seg frem til potensielle resultater, og dermed kjøre simuleringen flere ganger på kort tid for å sammenligne.

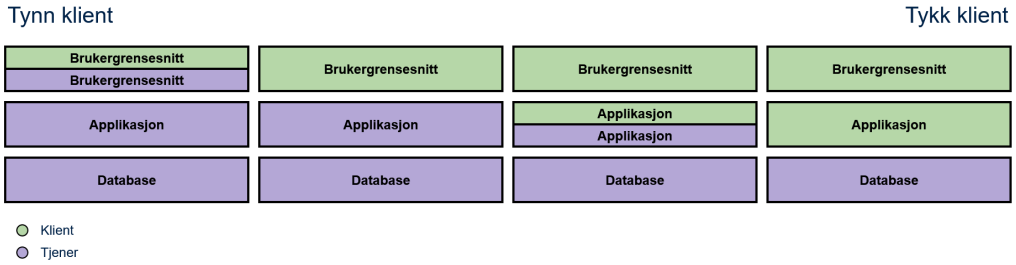
5.3.1 Valg av arkitektur for brukergrensesnitt

Som hovedplattform for brukergrensesnittet hadde vi valget mellom å gjenbruke en av de andre eksisterende teknologiene fra rapport- og dashbordløsningene, eller lage en dedikert løsning. Når vi vurderte de forskjellige alternativene la vi vekt på hvordan brukeren kan skrive inn data, hvilket format dataen kommer ut på og ytelsen til systemet. Fordelene vi så ved å benytte Grafana og Report Builder var at de var sterke på fremvisning av simuleringsresultatet. Den største ulempen med begge disse var at verktøyene ikke er tilrettelagt for å ta forskjellig input direkte fra brukeren, som for eksempel valg av defekttyper og

[3] Argos sin scanner vurderer plater utifra hvor mange av hver defekttype som er tilstede.

innsetting av nye terskelverdier for disse. Ettersom dette var en sentral del av simuleringsverktøyet ble det bestemt å gå bort fra disse løsningene.

En dedikert løsning ga rom til å skreddersy løsningen for nøyaktig det formålet den skulle ha. Dette gjorde det enklere å angi hvordan den skulle akseptere data, hvordan dataen skulle presenteres til brukeren, og hvordan verktøyet kobles sammen med API-løsningens eksisterende arkitektur.



Figur 12: Tykk/tynn klient-tjener modell

Ettersom kjernen av analyseplattformen er et API, utforsket vi klient-tjener som overordnet arkitekturmodell. Klient-tjener modellen (se figur 12) er delt inn i forskjellige trinn. Tykke klienter kan gjøre alt fra databehandling til presentasjon, mens API-et bare ville tilby rådata. For tynne klienter gjør API-et så mye prosessering som mulig, mens klienten bare håndterer presentasjon.

Hovedalternativene for valg av plattform for klienten var web-applikasjon, mobil-applikasjon og desktop applikasjon. Verktøyet skal brukes av forskjellige målgrupper på forskjellige lokasjoner, potensielt også på forskjellige enheter. Av den grunn kom vi frem til at en web-applikasjon var den mest fleksible løsningen for vårt verktøy. Erfaringsmessig er det i tillegg enklere å gjennomføre rask prototyping for web-applikasjoner enn desktop.

Web-applikasjoner er tynne klienter, da applikasjonen ikke må installeres på hver enhet det kjører på, men er tilgjengelig gjennom nettleseren. Verktøyets ytelse vil dermed påvirkes i liten grad av selve klienten, men heller internett forbindelsen og ytelsen til API-et [28]. Begrensningene som må vurderes ved implementasjon av denne modellen er hvor mye prosessering som kan gjøres på klientsiden, hvor kompleks grafikk som benyttes og hvor mye data som kan lagres på brukerens lokale enhet.

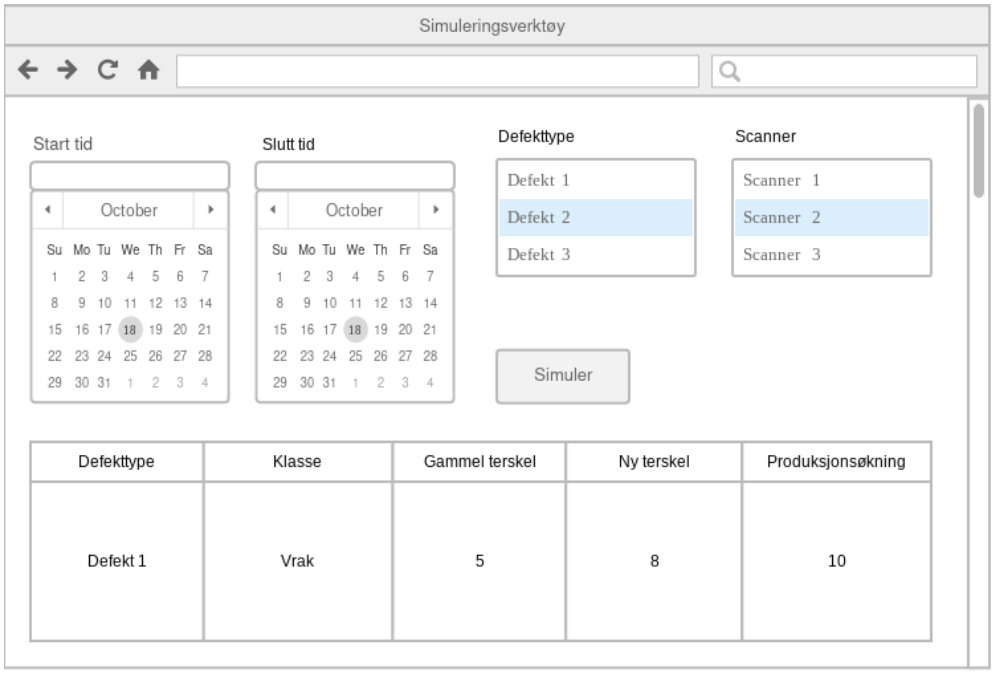
5.3.2 Innhold

For å oppfylle kravene til simuleringsverktøyet må det minst inneholde følgende elementer:

- Start og sluttidspunkt.

- Valg av defekttype.
- Valg av scanner.
- Mulighet for å spesifisere nye terskelverdier.
- Resultat av simulering.

Figur 13 viser en tidlig prototype av nettsiden. Her vil brukeren, etter å ha skrevet inn nødvendig informasjon, få en tabelloversikt over hvordan hver valgte defekter påvirker klassifiseringen til forskjellige plater.



Figur 13: Modell av de viktigste komponentene i simuleringsverktøyet.

5.4 Dataflyt for generering av rapport

Gruppen diskuterte tidlig i prosessen tre hovedmuligheter for hvordan vi skulle aggregere data fra databasen til Report Builder:

Direkte mot databasen

Gjennom Report Builder kunne vi enkelt koble verktøyet direkte opp mot databasen, og deretter gjøre spørringer for å hente data. Dette åpner for enkel implementasjon, men legger mer av ansvaret på brukeren av rapportverktøyet. Grunnlaget for dette er at dataen som kommer inn vil da være rådata, noe som i vårt tilfelle ville medføre at ytterligere manipulasjon var nødvendig for å skape meningsfull statistikk. Dette måtte blitt gjort for hver rapport i Report Builder, noe som ville vært langtekkelig og lite brukervennlig. Dette var heller ikke et alternativ Argos kunne tatt i bruk i dag, da nåværende versjon av Report Builder (2019) ikke støtter direkte tilkobling til SQL Server 2012. Da mange av spørringene som gjerne er interessante i en rapport allerede var laget for dashbordet, var det ønskelig å gjenbruke disse, noe som ikke var mulig med denne løsningen. I tillegg var sikkerhetsaspektet også en faktor, da manuell innskriving av spørringer direkte mot databasen åpner for sikkerhetssvakheter som SQL-injeksjoner.

Direkte med stored procedures

For å automatisere selve spørringsprosessen og flytte dette ut av selve rapportverktøyet, er det mulig å bruke stored procedures. Stored procedures er en betegnelse på en gruppe ferdiglagde spørringer som lagres i databasen. Fordelen med dette er at spørringene kan benyttes på tvers av rapporter, som simplifiserer prosessen med å legge inn eller endre data i rapportmalene. Denne løsningen lider også av flere av de samme svakhetene som den forutnevnte, selv om modulariteten var noe forbedret.

Gjennom API-et

Report Builder støtter også aggregering gjennom andre datakilder, blant annet eXtensive Markup Language (XML). Dette gir oss muligheten til å benytte API-et for å hente data fra databasen, konverte til XML, og bruke dette som datakilde. Dette løste flere av problemene nevnt over. På denne måten trengte spørringer kun og skrives én gang, da spørringene ville være felles for både dashbord, simulering og rapport. Dataene som kom inn ville da også være ferdig behandlet, noe som ville tilsi at det ville være enklere for alle typer brukere å gjøre endringer

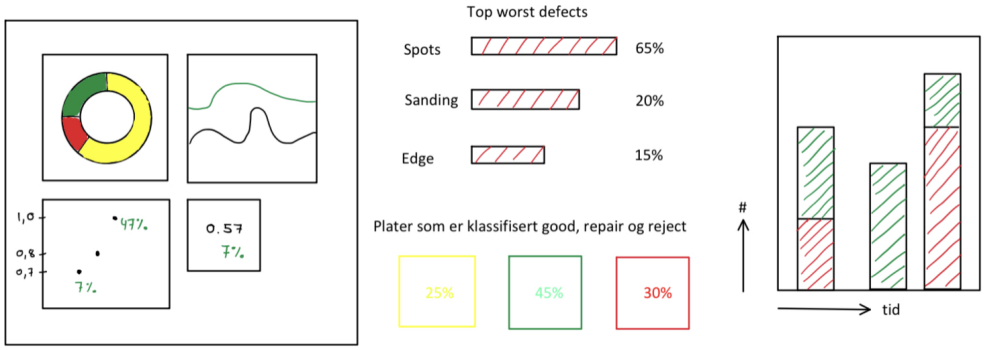
i rapportmalene. Ved å gå gjennom API-et vil rapportløsningen være helt uavhengig av underliggende datakilde. Dette innebærer at det er mulig å benytte nyeste versjon av Report Builder med eksisterende versjon av SQL Server 2012. Samtidig vil også sikkerheten ivaretas, da de underliggende spørringene ikke eksponeres for bruker. Nedsiden med denne måten å løse det på, er at XML er typeløst uten et skjema, noe som medfører at datasettene som sendes til Report Builder vil bestå kun av tekst. Dette medfører at dersom dataen skal behandles på noe vis som krever matematiske operasjoner, vil det være nødvendig å konvertere disse til faktiske tall. Dette er mulig gjennom Report Builder, men vil medføre noe ekstra arbeid.

På bakgrunn av disse avveiningene bestemte vi oss for å gå for den API-baserte modellen. Dette ga oss fleksibilitet til å gjenbruke funksjonalitet, samtidig som at det gir muligheten til å utvide med flere verktøy i fremtiden.

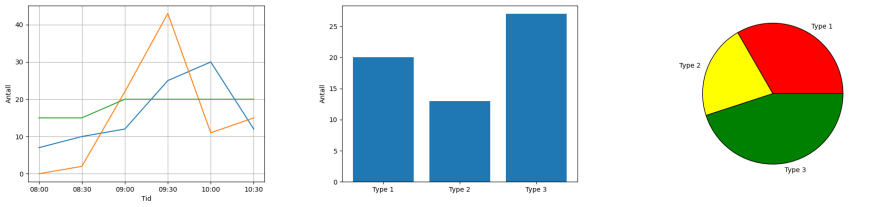
5.5 Dashbord

Et dashbord var ønsket av Argos da det var ønskelig å visualisere produksjonsdata i sanntid. Dette innebærte å visualisere forskjellige produksjonmål, som for eksempel antall plater og antall defekter. Argos kom ikke med spesifikke krav til innholdet i dashbordet, da deres primære mål var å se potensialet i løsningen.

Dermed var det essensielt å fastslå hvilke data som var interessante for kunden, og hvordan disse kunne fremstilles for best mulig utbytte. Denne informasjonen tilegnet vi oss gjennom dialog med oppdragsgiver, intervju med salgsrepresentanter i selskapet og brainstorming innad i gruppen. I dette kapittelet vil vi ta for oss prosessen bak utformingen av hvert panel.



Figur 14: Resultat fra brainstorming.



(a) Linjediagram **(b)** Søylediagram **(c)** Sektordiagram

Figur 15: Eksempel på mulige visualiseringer.

5.5.1 Utforming av paneler

For å kunne implementere kravene til innhold i dashbordet, måtte vi bestemme hvilke visualiseringsformer som var aktuelle. For å lage et tidlig utkast, begynte vi med brainstorming. Figur 14 viser resultatet av denne prosessen. Deretter gikk vi gjennom de ønskede produksjonemålene for å avdekke potensielle visualiseringsformer.

Antall produserte plater

For å vise et historisk bilde av plater som ble produsert med ulik gradering, kom vi frem til at et linjediagram [29] kunne vært et godt alternativ. Et linjediagram passer generelt godt til å vise utvikling over tid.

Ved å bruke en linje per gradering, er det enkelt mulig å tolke hvor mange plater i hver gradering som blir produsert over tid. Figur 15a viser hvordan dette kunne sett ut. Et linjediagram passer derimot dårlig for å vise data i sanntid, da det er vanskeligere å sammenligne verdiene opp mot hverandre. Et søylediagram [30] vil av den grunn være bedre egnet, ettersom man kan vise graderingene som individuelle søyler på x-aksen. På denne måten er det enkelt å se hvor mange plater av hver gradering som blir produsert. Dersom det er ønskelig å se hvordan andelene utgjør helheten, vil et sektordiagram [31] som vist i figur 15c være bedre egnet.

Heatmap

Et heatmap representerer hvor det finnes konsentrasjoner av defekter. For å visualisere dette tok vi utgangspunkt i dimensjonen til platene, for å deretter fremheve hvor defektene inntreffer. En fargegradient kan benyttes for å utheve hvor mange defekter som er på samme sted. På denne måten vil områder med få defekter være lyse, mens områder med mange defekter være mørkere.

Forekomster av defekttyper

Et sentralt produksjonsmål for trevirkeindustrien er fordelingen av defektene som inntreffer. Målet med dette er å identifisere hvilke defekter som påvirker graderingen. Det er dermed viktig å få en rask oversikt over hvilke defekter som

inntreffer oftest, og om denne trenden er ny. På samme måte som antall plater per gradering vil et linjediagram kunne vise historisk trend. Det er derimot viktig å være oppmerksom på antall defekttyper i linjediagrammet, da det raskt blir vanskelig å separere linjene. Søyle- og sektordiagrammer passer også godt for å vise fordelingen av defekttyper i sanntid.

Hente produktinformasjon
Produktinformasjon består av produktnavn, serie, dato, kunde, presstype, pressplate ID, og så videre. På grunn av fokuset på tekst og tall, vil en tabellstruktur passe godt for dette. En tabell gir oversikt over dataen mellom kolonner og rader [32]. Kolonnene i tabellen består av valgt informasjon, og radene representerer hver plate. På den måten kan man enkelt se informasjon om hver plate som er blitt produsert, eller aggregere dette for en mer helhetlig oversikt.

Linjehastighet
Enkelte fabrikker har ofte produksjonsmål, og må derav ha en viss hastighet på linjen. En hastighetsmåler, eller speedometer, vil gi en enkel og oversiktlig fremvisning av linjehastigheten. Her kan man vise hastigheten i kontekst, og på den måten se om den nåværende hastigheten er lavere enn målet som er satt. Et alternativ kan være å vise hastigheten som en ren tallverdi, men da kreves det at brukeren har innsikt om hastigheten for å få utbytte av den.

6 Implementasjon

I dette kapittelet viser vi hvordan teknologivalgene, kravene og designet har blitt brukt for å implementere de funksjonalitetene som er ønsket. Kapittelet er delt inn i tilsvarende måter som tidligere, der vi først viser hvordan fellesplattformen har blitt utviklet, for så å beskrive hvordan de eksterne tjenestene benytter seg av denne.

6.1 Rammeverk for aksessering av data

Rammeverket for aksessering av data er bindeleddet mellom de eksterne tjenestene og databasene til Argos. Plattformen er bygd for å være fleksibel, men samtidig oppfylle alle kravene som satt i kravspesifikasjon og design. Rammeverket implementerer FastAPI for å ta imot spørringer, og sender dette videre til en egenlaget modulkontroller som finner modulen med ønsket data. Disse modulene er skreddersydde, og det er lagt opp til at det er enkelt for både Argos' ansatte og deres kunder utvide systemet med ekstra moduler dersom det er ønsket. Modulene får automatisk tilgang til relevante SQLAlchemy sesjoner, slik at det er enkelt å aksessere data fra nåværende databaseløsning.

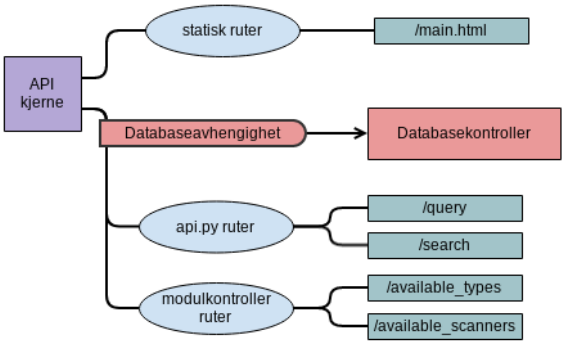
6.1.1 Rammeverkets grunnstruktur

Listing 1 viser hvordan rammeverkets grunnstruktur er bygd som en tradisjonell Python pakke, der hvert lag er implementert i sin egen fil. Dette medfører at det er mulig å kjøre applikasjonen som en helhet, men det er også mulig å bare kjøre utvalgte deler av dette dersom det er ønskelig.

```
ArgosAnalysePlatform
    html
    plugins
        __init__.py
        boards.py
        defects.py
        heatmap.py
        regfile_reader.py
        xml_converter.py
    __init__.py
    __main__.py
    api.py
    database.py
    plugin_manager.py
    util.py
```

Listing 1: ArgosAnalysePlatform filstruktur.

Dataflyten når spørringer gjennomføres kan sees i figur 11.



Figur 16: Fordeling over FastAPI ruter, endepunkter og avhengigheter.

6.1.2 API

API-laget har ansvar for eksponering av tilgjengelige endepunkter, samt validering av innkommende og utgående datastrukturer. Dette gjorde vi ved å benytte oss av FastAPI. Kjernen til FastAPI består av å dekorere Python funksjoner som mottaker av `GET`, `POST`, `PUT` eller `DELETE` operasjoner. For å kunne skalere ved større systemer er det i tillegg mulig å benytte FastAPI `Router` klasser, som deretter kan importeres av kjerneklassen for å få tilgang til de aktuelle endepunktene. Figur 16 viser hvordan vi har implementert denne fordelingen.

Basert på designet og kravene til API-et, kom vi frem til at de fleste spørringene er avhengige av tilgang til databasen. For å gi denne tilgangen benyttet vi oss av FastAPI sitt innebygde system for å dynamisk legge ved ekstra data i innkommende spørringer. I listing 2 blir `/query` endepunktet satt opp gjennom `api.py` ruteren fra figur 16. Dette gjorde det enkelt å legge ved databasen til bruk i de forskjellige endepunktene. Denne modellen medfører at tilgangen til databasene bare spesifiseres én gang, og gjør det enkelt å bytte ut denne ved testing.

```
1  from ArgosAnalysePlatform.api import router as api_router
2
3  class APIApp(FastAPI):
4      def __init__(self):
5          self.include_router(api_router)
```

```
6  router = APIRouter()
7
8  class JSONQueryInputType(BaseModel):
9      range: TimeRangeType
10     intervalMs: int
11     scopedVars: Dict[str, ScopedVarsModel]
12     targets: List[TargetsModel]
13
14 @router.post(
15     "/query",
```

```
16      response_model=Union[JSONQueryTimeseriesResponseModel, (...)]
17  )
18  def query(request: Request, inp: JSONQueryInputType):
19      ...
```

Listing 2: Forkortet utdrag fra analyseplattformen.

En sentral del av API-laget, som nevnt i design (4.2), er valideringen av innkommende og utgående datastruktur. FastAPI benytter seg av Pydantic for serialisering og deserialisering av data. Pydantic kombinerer Python sitt innebygde typesystem med egne klasser for mer komplekse modeller. Disse klassene kan arve av hverandre for å lage ytterligere komplekse strukturer. Dersom Pydantic får felter den ikke kjenner igjen, vil den enten ignorere disse eller avbryte med en feilmelding. I listing 2 defineres inputmodellen ved `inp: JSONQueryInputType` og output gjennom `response_model=`. Fullt kodeeksempel av dette utdraget kan finnes i vedlegg K.

6.1.3 Modulkontrolleren

Modulkontrollerens ansvar er å holde en oversikt over systemets registrerte moduler, og gi et grensesnitt for kall til bruk i API-et. Vi har implementert dette ved hjelp av en egenskrevet klasse. For å registrere en modul kreves navnet som skal brukes til oppslag, samt hvilken kategori modulen skal høre til. Dette kan ses i første del av listing 3. Når moduler har blitt registrert er det også mulig å søke blant modulene dersom man ikke vet det nøyaktige navnet og kategorien for modulen man ønsker.

For å kalle modulene vil modulkontrolleren returnere det registrerte funksjonsobjektet, basert på oppgitt navn og kategori. Dette funksjonsobjektet kan deretter benyttes som en ellers ordinær Python funksjon. Denne implementasjonen av mikrokjernearkitekturen gir en fordel ved at det er mulig å gruppere modulene basert på hvilken ekstern tjeneste de er ment å benyttes med. Disse faktorene mitigerer også vanskeligheten med grensesnittet mellom kjernen og modulene, som er en svakhet med mikrokjernemodellen. Modulkontrolleren i seg selv trenger ikke å vite noe om hvilke parametre som blir benyttet med det returnerte funksjonsobjektet. Et eksempel på dette kan sees i andre halvdel av listing 3.

```
1  plugmanager = PluginManager()
2
3  @plugmanager.register_plugin(
4      'plugin_name',
5      category='report_builder'
6  )
7  def plugin_callable(txt: str):
8      print(txt)
```

```
9  # Kall modul gjennom modulkontrolleren.
```

```
pluginmanager_returned_callable = plugmanager.call_plugin(
    'plugin_name',
    category='report_builder'
)
pluginmanager_returned_callable('sample text')

# Kall modul som en vanlig funksjon.
plugin_callable('Sample text')
```

Listing 3: Registrering og kall av modul

6.1.4 Håndtering av databasetilkoblinger

I kravspesifikasjonen og designet ble det avdekket at det var ønskelig å enkelt kunne benytte seg av data fra flere databaser. Dette gjorde at databasetilgangslaget måtte ta hånd om kommunikasjon med enkeltdatabasene, samt tilby et grensesnitt for hvordan spørringene kunne gjøres mot flere databaser av gangen. Dette har vi implementert ved at SQLAlchemy tar seg av spørringer til enkeltdatabasene, mens en egenskrevet klasse gir et grensesnitt for hvordan disse spørringene kjøres mot flere databaser.

For å kunne gjøre spørringer med SQLAlchemy må det oppgis hvor databasen er. Dette gjør SQLAlchemy i sitt `engine` objekt. Dette objektet trenger informasjon om vertsnavn, brukernavn, passord, port og databasenavn for å koble seg til. For å sikre brukervennligheten og vedlikeholdbarheten til systemet var det viktig at det er enkelt for brukerene å legge til eller endre disse parameterene. Vi valgte å lagre denne dataen i en YAML-fil, da dette formatet er mer brukervennlig enn JSON for konfigurasjon [33]. Strukturen av denne filen kan sees i listing 4. Her lagres parameterene med et egendefinert nøkkelord som navn, og nødvendige detaljer som innhold.

```
shared:
    dbname: DatabaseName
    port: 1433

connections:
    wood_grain:
        ip: 1.1.1.1
        password: pass
    wood_smooth:
        ip: 2.2.2.2
        password: pass2
```

Listing 4: Spesifisering av tilkoblingsdetaljer for database i YAML.

6.1.5 Håndtering av databaseobjekter

Når tilkoblingsdetaljene er spesifisert benyttes de for å lage `Sessionmaker`-objekter. Disse objektene benyttes for å generere sesjoner, som brukes av modulene for å gjøre spørringene mot databasen. De nødvendige stegene for å sette opp en enkelt tilkobling kan sees i listing 5.

```
1 engine = create_engine('mssql+pyodbc://username:password@ip/
      tablename')
2 sess_maker = sessionmaker(bind=engine)
3 session = sess_maker()
4 return session.query(...)
```

Listing 5: Minimal kode for å gjøre spørringer med SQLAlchemy.

Når flere databaser blir spesifisert kan det bli komplisert å holde styr på disse individuelle objektene. Dette løste vi ved hjelp av en egenlaget klasse som tok hånd om lesing av tilkoblingsdetaljer, og oppsett av `Engine`- og `Sessionmaker`-objektene til bruk.

Modulene vet hvilke databaser de ønsker data fra, og trenger et sesjonsobjekt for hver av disse databasene for å gjøre spørringene. Med klassen vi laget for å håndtere databasetilkoblingene, var det enkelt å legge til rette for dette. Listing 6 viser hvordan samme spørring kjøres mot flere databaser samtidig gjennom vårt grensesnitt.

```
1 db = DatabaseConnections(...)
2 for name, session in db.sessioniter(['wood_grain', 'wood_smooth']):
3     return session.query(...)
```

Listing 6: Bruk av egenskrevet databaseklasse.

6.1.6 Knytte alt sammen

Endepunktene `/query` og `/xml/query` knytter alle disse lagene sammen for å gi de eksterne systemene tilgang til etterspurt data fra databasen. Begge disse endepunktene benytter seg av dataflyten som vist i sekvensdiagrammet i figur 11.

Endepunktene tar de samme parametrene: start- og sluttidspunkt, tidsbolkoppløsning, ønskede skannere og hvilke moduler som skal kalles. Endepunktene prosesserer disse parametrene, før den benytter seg av databasekontrolleren for å finne sesjonsobjekter, og modulkontrolleren for å finne modulene. Listing 7 en forkortet versjon av hvordan dette er implementert i `/query` endepunktet.

```python
@router.post("/query")
def query(request: Request):
    # Forbered tidsparametre her.
    scanners = ...
    modules = ...

    response = []
    for module_name in modules
        for db_name, session in request.database.sessioniter(
            scanners):
            response.extend(
                plugmanager.call_plugin(module_name, category="")(
                    session,
                    db_name,
                    ...
                )
            )

    return response
```

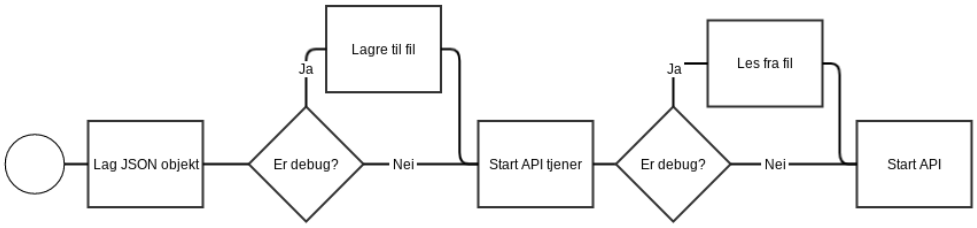
Listing 7: Sammenkobling av lagene i `/query`.

6.1.7 Oppstart

Før oppstart av API-et, trenger den informasjon om miljøet den skal kjøre i. Dette angis gjennom spesifiserte kommandolinjeargumenter som tolkes av Click [34]. Dette er et rammeverk som gjør det enkelt å spesifisere og validere kommandolinjeargumenter i Python. Click validerer typene til parametrene, samt legger til rette for en beskrivende hjelpetekst via `--help`, som vist i listing 8.

```
  Run the Argos Analysis platform.

  Will launch the API with specified options.

Options:
  --external-plugins TEXT  Path to external plugins
  --debug                  Enable debug mode. This will show
                           extra debug output and enable
                           automatic reload when source files
                           change.

  --no-reflect             Do not reflect the database tables
                           if set.

  --port INTEGER           The port to run the server on
  --host TEXT              The host to listen on
  --help                   Show this message and exit.
```

Listing 8: `--help` for rammeverket.



Figur 17: Prosessering av kommandolinjeargumenter .

Flyten i programmet varierer basert på om brukeren ønsker å kjøre FastAPI i sin `--debug` modus eller ikke. Dersom debug modus er satt, vil FastAPI starte API-tjeneren i en annen kontekst enn den som fikk kommandolinjeargumentene. For å komme oss rundt dette ble parametrene gitt til hovedinstansen lagret i JSON, som leses i den nye Python konteksten. Figur 17 viser et flytskjema av denne prosessen.

6.2 Analyse

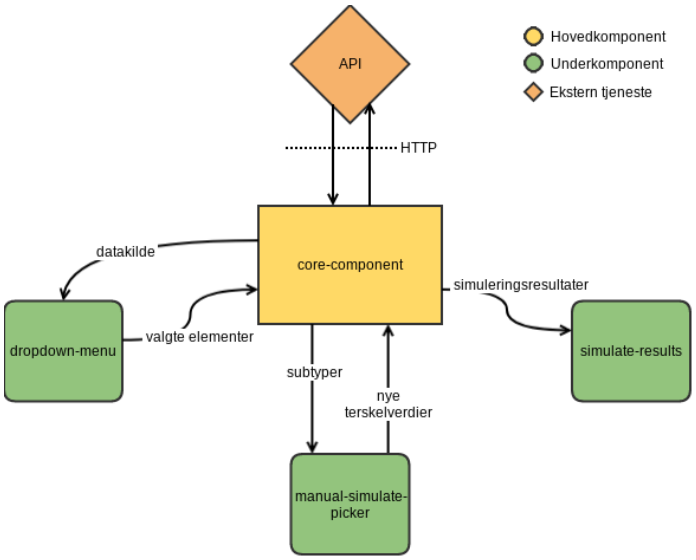
Simuleringsverktøyet består av en webapplikasjon som har tatt inspirasjon av Single Page Application (SPA)-tankegangen. Kommunikasjon og annen data som trengs fra API-et hentes asynkront ved hjelp av jQuery. Kjernen av webapplikasjonen er Vue. Dette verktøyet gjør det enkelt å oppdatere applikasjonens utseende basert på datastrukturen i bakgrunnen. Simuleringsalgoritmen i API-et henter defekttypene som er angitt av brukeren, og returnerer en oversikt over hvordan endringer i vurderingen av disse defekttypene påvirker produksjonen. Det endelige brukergrensesnittet for verktøyet har blitt lagt ved som vedlegg M.

6.2.1 Nettsidens grunnstruktur

Nettsiden er bygd med Vue. Dette er et rammeverk som benyttes ved utvikling av SPA-webapplikasjoner. Kjernen til Vue består av spesialbygde JavaScript-objekter, eller komponenter, som igjen er koblet til HTML-elementer. Når dataen som er lagret i komponenten endres, vil Vue selv håndtere hvordan HTML-strukturen oppdateres automatisk. Vue har en rekke forhåndsdefinerte HTML-tags som beskriver hvordan dataen i JavaScript-objektet skal reflekteres i utseendet. Hver komponent er selvstendige enheter og de må definere klare grensesnitt om hvordan data skal sendes mellom hverandre.

Den overordnede strukturen i simuleringsverktøyet som er vist i figur 18 består av tre typer underkomponenter `dropdown_menu`, `manual-simulate-picker`, `simulate_results` , og en hovedkomponent som håndterer kommunikasjonen mellom komponentene. Denne strukturen har gjort det enkelt å gjenbruke komponenter, samt enkelt

kontrollere hvilke komponenter som vises når, og hvordan.

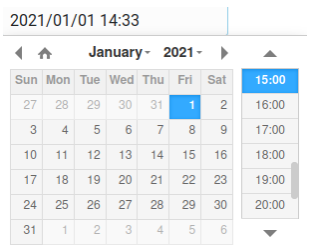


Figur 18: Overordnet Vue komponentstruktur .

6.2.2 Valg av parametre

For å gjennomføre simuleringen må brukeren oppgi start- og sluttidspunkt, ønskede defekttyper og hvilken skanner dataen skal hentes fra. For å slippe å skrive inn datoen for hånd har en klikkbar kalender blitt implementert. Figur 19 viser jQuery modulen jquery-datetimepicker, som ga oss et rent brukergrensesnitt for valg av både dato og klokkeslett.

Valg av defekttyper og skanner besto av å la bruker velge blant eksisterende verdier. Denne datastrukturen passer bra med nedtrekkslister, men vi trengte i tillegg en tabell over valgte alternativer. Selve nedtrekkslisten ble implementert med jQuery modulen Select2. Denne modulen gjorde det enkelt å legge inn alternativene basert på data direkte fra API-et, og la modulen håndtere hvordan brukeren samhandler med dette. Tabellen av valgte alternativer implementerte



Figur 19: Valg av dato og klokkeslett.

vi ved å la Vue håndtere en HTML-tabell basert på innholdet i en intern Javascript liste. Listing 9 viser hvordan vi implementerte dette gjennom definisjonen av HTML-tabellen og Javascript koden som krevdes for å oppdatere denne. Her benyttes Vue sin `v-for` attributt for å iterere over innholdet i `rows` listen.

```
<table>
    <template v-for="(row, i) in rows">
        <tr>
            <td>
                <button v-on:click="remove(i)">Remove</button>
            </td>
            <td>{{row}}</td>
        </tr>
    </template>
</table>
```

```
// Store the 'this' for Vue as it will be overriden by Select2s 'this'
// inside the event handler
var external_this = this;
dropdown_element.on("select2:select", function (e) {
    external_this.add(e.params.data.id);
});
```

Listing 9: Insetting av nytt valg i tabell over valgte alternativer.

Nedtrekkslisten og tabellen over valgte alternativer ligger i en egen komponent `dropdown-element`. Ved å samle disse, kan innholdet enkelt gjenbrukes av både defekttype- og scannerparameterene. Listing 10 viser hvordan dette enkelt gjøres ved å benytte komponenten som en HTML tag, med tilhørende attributter.

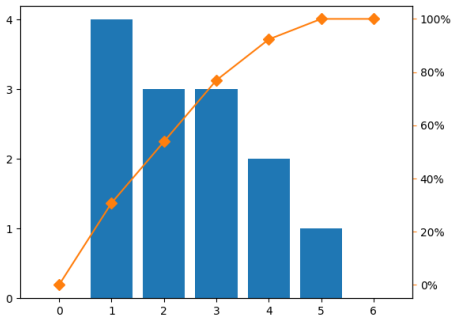
```
<div id="vueApplication">
    <dropdown-menu
            dropdown_label="Scannere"
    ></dropdown-menu>
    <dropdown-menu
            dropdown_label="Typer / Subtyper"
    ></dropdown-menu>
</div>
```

Listing 10: Gjenbruk av komponenter.

6.2.3 Simuleringsprosessen

Når de tidligere nevnte parametrene har blitt definert kan selve simuleringen gjennomføres. Dette startes av web-applikasjonen gjennom API-endepunktet `/simulate_images`.

Når Argos sin skanner har gjenkjent alle defektene på en plate, vil den kategorisere defektene ved å sammenligne antallet defekter av en type mot dens forhåndsdefinerte terskelverdier. Målet med simuleringen er å dermed å finne ut



Figur 20: Paretodiagram for fordelingen av hakk .

hvordan en endring av disse terskelverdiene vil påvirke produksjonen. Et eksempel på dette er dersom den gamle terskelverdien var to, men det ønskes å se hvor mange flere plater som hadde blitt godkjent ved å øke til fire.

For å visualisere dette har vi valgt å representere fordelingen av antall defekter (i dette tilfellet hakk) i et paretodiagram. Dette er et søylediagram som beskriver feilsituasjoner på x-aksen, og antall forekomster på y-aksen. På toppen av søylene er et linjediagram som viser kumulativ opptelling av antall feil. Figur 20 viser en fordeling av situasjonen over, der feil 1 inntraff fire ganger. Den kumulative fordelingen (her i oransje), viser at dette tilsvarer omtrent 30% av totale antall feil.

Simuleringsalgoritmen i analyseplatformen vil generere et paretodiagram for hver defekttype. Y-aksen angir antall plater som ble nedgradert spesifikt av denne defekttypen, og x-aksen angir antall feil av defekttypen som forekom på platene. I eksempelet over er det fire plater som hadde en forekomst av defekttypen. Simulert produksjonsendring kan med dette regnes ut ved å summere søylene mellom ny og gammel grense (x-aksen). I figur 20 ville produksjonen økt med 8 plater dersom gammel grense er en og ny grense er fire (inklusiv).

For å konvertere rådataen fra databasen til et format som kan vises gjennom et paretodiagram må følgende steg utføres:

1. Finne plater som har angitte defekttyper som årsak for nedgradering.
2. Filtrere bort plater med andre defekter som ellers ville ført til nedgradering.
3. Telle antall defekter per defekttype for hver plate.
4. Telle antall plater ved å gruppere defekttype og antall defekter.

Siden brukeren potensielt ønsker å spørre om flere defekttyper om gangen, var det viktig å tenke på ytelse i algoritmen. De tre første stegene ble dermed implementert gjennom en større spørring til SQL, mens steg fire ble gjort i Python.

Her benyttet vi oss av SQLAlchemy sin modulære struktur for å dele opp koden inn i mindre, mer forståelige deler.

Listing 11 viser hvordan steg 1 ble gjennomført ved å skrive en `subquery`, eller delspørring. Gjennom listen `where_argosnames` og filteret `Defects.c.IsWorstDefect == 1` blir `Defects` tabellen filtrert slik at den bare inneholder plater som er aktuelle.

Steg en ble gjennomført ved å hente alle platene som har angitte defekttyper som grunnlag for nedgradering. Dette ble gjort som en subquery (se listing 11), som gjør det mulig å referere til denne i både steg to og tre.

```
worst_subq = (
    session.query(Defects.c.BoardID, Defects.c.ArgosName)
    .join(Boards)
    .filter(Boards.c.Time.between(start_time, end_time))
    .filter(Defects.c.IsWorstDefect == 1)
    .filter(Defects.c.ArgosName.in_(where_argosnames))
    .subquery()
)
```

Listing 11: Subquery for filtrering av Defects tabell.

Steg to ble implementert i en separat delspørring der ny og gammel klasse defineres ved variablene i listing 12. Her hentes den gamle verdien som den høyeste klassen av alle defekter for en plate. Den nye klassen blir hentet på samme vis, men ser bort ifra defekttypen som nedgraderte platen. Til slutt vil delspørringen filtrere bort alle plater som har lik ny og gammel klasse.

```
old_class = func.max(Defects.c.Class)
# Get the highest class that is not worst defect.
new_class = func.max(
    case(
        [(Defects.c.ArgosName != worst_subq.c.ArgosName, Defects.c.
            Class)], else_=0
    )
)
```

Listing 12: Definisjon av gammel og ny klasse.

Når filtreringen i steg en og to har blitt gjennomført kan steg tre telle opp antall defekter per defekttype på platene som gjenstår. Dette vil gi et resultat tilsvarende tabell 25. Steg fire kan deretter telle opp antall `plate id` for hver defekttype, samt hver nye og gamle klasse. Det endelige resultatet av denne prosessen er vist i tabell 26. For å plotte denne dataen i et paretodiagram blir kolonnen `antall defekter` representert av x-aksen, mens `antall plater` blir representert av y-aksen. Funksjonen som utgjør steg en til fire kan i sin helhet sees i vedlegg L.

Plate ID	Defekttype	Antall defekter	Gammel klasse	Ny klasse
1	Hull	5	Reparer	Godkjent
2	Bark	3	Reparer	Godkjent
3	Hull	5	Reparer	Godkjent
4	Bark	3	Reparer	Godkjent
5	Bark	3	Reparer	Godkjent
6	Hull	5	Vrak	Reparer

Tabell 25: Foreløpig resultat etter de første tre stegene.

Defekttype	Antall defekter	Ny klasse	Gammel klasse	Antall plater
Hull	5	Godkjent	Reparer	2
Bark	3	Godkjent	Reparer	3
Hull	5	Reparer	Vrak	1

Tabell 26: Endelig resultat etter prosessering.

6.2.4 Bestemme terskelverdier

Når datagrunnlaget har blitt laget ved hjelp av simuleringsprosessen kan dataen analyseres ved å angi hva den gamle terskelverdi var og hva ny verdien skal være. De eksisterende terskelverdiene ligger lagret i skannerenes Windows registre. Lesingen av disse verdiene ble gjort gjennom en egen klasse `ArgosRegistry`. For å holde systemet fleksibelt kan klassen lese registeret gjennom Windows `.reg` filformatet. Klassen eksponerer deretter et definert grensesnitt for å hente aktuelle terskelverdier basert på defekttype.

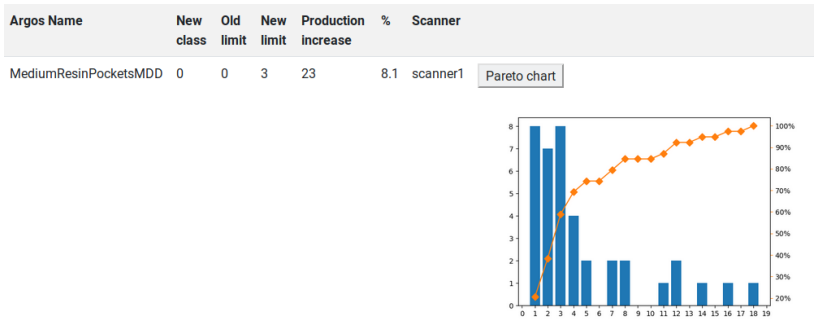
Den nye terskelverdien kan bestemmes på to måter, enten manuelt eller ved et automatisk forslag. Manuell spesifisering av tersklene gjøres ved at brukeren får opp et ekstra panel som gir muligheten å spesifisere terskelverdier for hver defekttype. Dette panelet er implementert med Vue-komponenten `manual-simulate-picker` som vist i figur 18.

Det automatiske forslaget regnes ut etter datagrunnlaget har blitt definert. Det automatiske forslaget forsøker å balansere hvor mange flere plater som blir oppgradert mot hvor mange flere defekter som tillates. Dette har blitt implementert ved å begynne på den gamle terskelverdien å regne ut et løpende standardavvik av plater for hver x-verdi. Dersom dette standardavviket blir større enn en gitt terskel, vil algoritmen foreslå siste x-verdi som ny terskelverdi. Svakhetene ved denne implementasjonen er at den potensielt vil overskyte dersom antall plater er relativt stabilt. Den vil også raskt stoppe både dersom forskjellen i antall plater er både positivt og negativt, da standardavvik er et avstandsmål uten fortegn. Den første svakheten har vi løst ved å legge inn en maksgrense for hvor langt algoritmen vil gå, uavhengig av standardavviket, på denne måten unngår vi at algoritmen foreslår en uendelig grense. På bakgrunn av datasettet vi har hatt tilgang til, har vi bedømt at den andre svakheten inntreffer såpass sjeldent

at det ikke vil være hensiktsmessig å mitigere dette.

6.2.5 Presentasjon av resultat

Når ny terskelverdi er valgt kan sluttresultatet av simuleringen presenteres. I tillegg til å se numerisk resultat, er det også mulig å se det i paretodiagrammet. Et utdrag av selve visningen i nettleseren kan sees i figur 21.



Figur 21: Simureringsresultat med paretodiagram.

6.3 Rapportgenerering

Ettersom vi i rapportdelen endte opp med å lage en proof of concept, valgte vi å gjenskape en rapport fra Argos for å vise potensialet i løsningen vår. Vi fikk tidlig i prosessen tilsendt eksisterende rapporter som inspirasjon til hvilke data det var interessant å vise, og vi valgte den mest brukte rapporten av disse som grunnlag (se vedlegg N). Vi anså det ikke som nødvendig å vise nøyaktig de samme dataene i vår forbedret versjon, da formålet i all hovedsak dreiet seg om å vise hva slags muligheter verktøyet kunne tilby. Den resulterende konseptrealiseringen kan ses i vedlegg P. For å vise at dette valget ikke ble tatt på bakgrunn av begrensinger i vår løsning, har vi i tillegg gjenskapt rapporten basert på data Argos syntes var viktigst. Resultatet av gjenskapningen kan ses i vedlegg O.

6.3.1 Dataflyt

For å illustrere dataflyten i rapportgenereringen, har vi valgt å følge et datasett fra konsept til realisering i rapport. Denne prosessen vil være identisk for alle datasett som skal innføres, og består av følgende steg:

0. Hvilke data vi er ute etter
1. Henting av data
2. Konvertering og registrering
3. Definering av data
4. Visualisering av data

Datasettet vi skal se nærmere på i dette tilfellet er nedgraderinger per defekttype (se figur 24), da dette er en av de mer sentrale produksjonsmålene. Det er i flere av stegene inkludert deler av spørringene for å beskrive det som diskuteres. Den fullstendige spørringen finnes i vedlegg Q.

Steg 0: Hvilke data vi er ute etter

Før første steg i prosessen kan begynne, er det essensielt å vite hvilke data som er nødvendig for gjennomføring. For å kreere grafen over er vi avhengig av tre kolonner i databasen:

- **Type**: Typen defekt.

- **Class**: Klassifisering av defekten. Klassifiseringen sier noe om hvilken klasse defekten tilhører. Er defekten av klasse 0 er den ubetydelig og trenger ingen ytterligere behandling. Klasse 1 og 2 vil henholdsvis si at defekten trenger reparasjon eller at den er så alvorlig at platen må vrakes.
- **BoardID**: Den unike ID-en til hver plate, som her brukes for å telle antall forekomster av defekter.

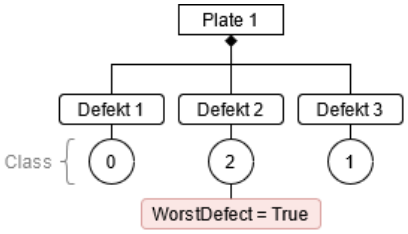
Steg 1: Henting av data

Når vi har bestemt hvilke data vi er ute etter, er neste steg å hente de fra databasen.

```
1 data = (
2     session.query(Defects.c.Type, func.count(Boards.c.ID), Defects.
          c.Class)
3     .join(Defects)
4     .filter(Boards.c.Time.between(start_time, end_time))
5     .filter(Defects.c.IsWorstDefect == "1")
6 )
```

Listing 13: Spørring for å hente type, klasse og antallet unike forekomster.

I listing 13 henter vi de tidligere nevnte dataene fra databasen og filtrerer på start- og slutt-tidspunkt. Deretter filtrerer vi på om defekten er markert som WorstDefect.



Figur 22: Hvordan WorstDefect status tildeles.

Som navnet impliserer, forteller dette oss om feiltypen var grunnlaget for nedgraderingen av platen. Som nevnt tidligere, og beskrevet i figur 22, blir den feiltypen med høyest klasse for hver plate markert som `WorstDefect`. Grunnen til at vi her filtrerer på dette parameteret, er at vi i vårt tilfelle kun er interessert i de feiltypene som faktisk nedgraderte platene.

Steg 2: Konvertering og registrering

Ettersom vi i designfasen bestemte at Report Builder skulle aggregere data fra rammeverket, måtte vi benytte XML som serialiseringsmetode. Dataene som returneres av spørringen består av resulterende SQLAlchemy-rader.

```
for item in dataset:
    # Set row child-node for each individual result in dataset
    row = ET.Element("Row")

for i, part in enumerate(item):
    # Set each part of every result as individual value child-nodes
    ET.SubElement(row, column_names[i]).text = str(part)

    yield row
```

Listing 14: Konvertering av data til XML.

I selve konverteringen 14 aksesseres hvert individuelle objekt i datasettet ved hjelp av dens tilhørende indeks, og for hvert av disse genereres et XML-element. Resulterende trestruktur kan ses i listing 15.

```
<root>
    <Row>
        <DefectType></DefectType>
        <Count></Count>
        <Class></Class>
    </Row>
</root>
```

Listing 15: Grunnleggende XML-struktur.

Selve behandlingen av dataen er nå ferdigstilt, og den er nå klar til og sendes til Report Builder. Første steg i denne prosessen gjøres i listing 16, der spørringen registreres som en modul. På denne måten vil spørringen være tilgjengelig gjennom `/xml/query`-endepunktet, og kan dermed hentes gjennom Report Builder.

```
@plugmanager.register_plugin(
    "Downgrades",
    category="report_builder"
)
def xml_get_downgrades(...)
```

Listing 16: Registrering av modul.

Steg 3: Definering av data

Det første som må gjøres for å få dataen inn i Report Builder er å legge inn datakilden. For å oppnå denne tilkoblingen trenger vi kun å oppgi tilkoblingsdetaljene, som består av IP, port og navnet på endepunktet: `http://localhost:8000/xml/query`.

Parameter	Parameterverdi
start_time	Brukerinput
end_time	Brukerinput
resolution	30 minutter
targets	scanner1
action	Downgrades

Tabell 27: Parametere med tilhørende verdier.

Når datakilden er definert vil neste steg bestå i å spesifisere hvilke datasett[4] vi er ute etter. Defineringen av datasett foregår over to steg:

Først må det spesifiseres hvilke elementer i det innkommende datasettet vi er interesserte i. Dette gjøres ved å opprette en spørring i *XML Query Syntax*, noe som syntaktisk vil gjøres på lignende måte som ved det mer kjente Xpath [35]. Spørringen i listing 17 spesifiserer den såkalte `ElementPathen`, eller elementstien, til dataene vi er ute etter. Oppbyggingen av stien vil samsvare med den resulterende trestrukturen fra steg to (se figur 15).

```
1 <Query>
2     <ElementPath>root/Row{DefectType, Class, Count}</ElementPath>
3 </Query>
```

Listing 17: XML Query Language spørring.

Videre krever API-et fem faste parametere for å gjennomføre spørringer mot databasen, og disse spesifiseres også i datasettet. Start- og sluttidspunkt har vi valgt å definere som globale parametere for rapporten, mens resterende verdier legges direkte i datasettet (se tabell 27). Ved å definere globale parametere på denne måten, kan brukeren selv spesifisere disse ved generering av rapporten.

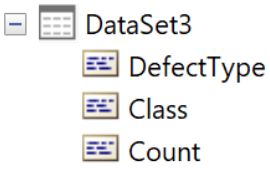
Steg 4: Visualisering av data

Ved ferdigstilt oppsett av datasettet (se figur 23), gjenstår bare visualisering i rapport. I Report Builder gjøres dette simpelten ved å velge diagramtype og deretter plassere data inn i diagrammet. I vårt tilfelle har vi, for dette datasettet, valgt å representere dataen gjennom et liggende stolpediagram (se figur 24).

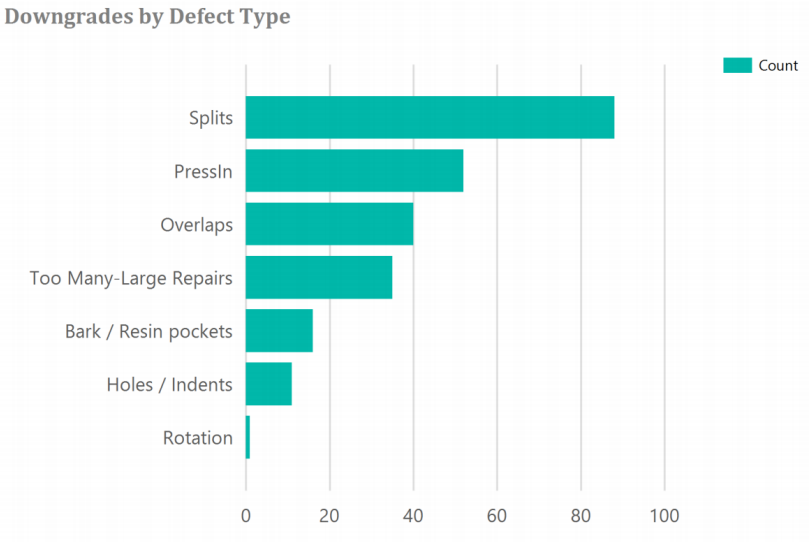
6.4 Gruppering av innhold

I den opprinnelige produksjonsrapporten (se vedlegg N) er innholdet gruppert, noe vi anså som en sentral del av eksisterende funksjonalitet. Dette valgte vi å

[4] Datasett representerer et utvalg av data fra en datakilde.



Figur 23: Datasett i Report Builder.



Figur 24: Antall nedgraderinger per defekttype.

implementere i vår løsning ved å implementere to ytterligere datasett parametere `gbcolumn_name` og `gbcolumn_value`. Disse beskriver henholdsvis navnet på kolonnen i databasen det skal sorteres på og verdien til denne kolonnen.

Ettersom enhver rapport ikke nødvendigvis skal grupperes, valgte vi å implementere disse som valgfrie. Dette var enkelt å implementere, da det gjennom API-et allerede var lagt til rette for ved at tilleggsinformasjon til spørringene kunne innføres gjennom `extra_data`. På denne måten kunne vi supplere spørringene med ytterligere data slik at resultatet ble gruppert. Dette ble implementert ved å filtrere dataen før den sendes til endepunktet, dersom ytterligere parametere er oppgitt (se listing 18).

```
1 # If group-by column is included in call, filter data by it
2 if "gbcolumn_name" in extra_data:
3     data = data.filter(
4         getattr(Boards.c, extra_data["gbcolumn_name"])
5         == extra_data["gbcolumn_value"]
6     )
```

Listing 18: Gruppering i spørring.

I Report Builder legges det til rette for tilsvarende funksjonalitet gjennom subre-

ports. En subreport fungerer som en referanse til en annen frittstående rapport, og på den måten er det mulig å generere den basert på andre parametere i hovedrapporten. Dermed kunne vi supplere subreporten med hvilket parameter den skulle grupperes på, og videre hente data basert på dette parameteret gjennom `gb_column`-parametrene.

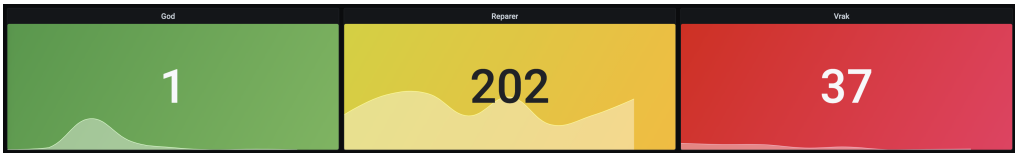
6.5 Dashbord

Dashbordløsningen har blitt implementert i Grafana. Her har vi implementert panelene som ble utformet i designfasen for å vise et dashbord med statistikk over sanntidsproduksjon. Panelene bruker hovedsaklig Grafana sin innebygde funksjonalitet, mens heatmap har blitt implementert ved å i tillegg benytte eksterne verktøy.

Nødvendig data for disse panelene blir hentet gjennom fellesplattformens API. Grafana sitt fleksible grensesnitt gjør det enkelt å skreddersy dashbordene til brukerenes spesifikke behov.

6.5.1 Antall produserte plater

Et av de mest sentrale målene i produksjonen er å vise hvor mange plater som er produsert, kategorisert etter gradering. Denne graderingen består av godkjent, må repareres og vrak. Her kreves både en oversikt over hvor mange plater som har blitt produsert i sanntid, men også over en lengre periode. Figur 25 viser hvordan sannttidsdataen kan se ut.



Figur 25: Antall produserte plater i sanntid.

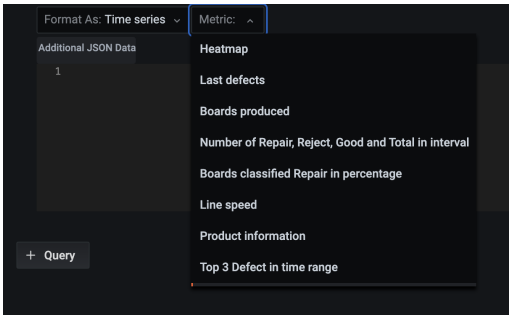
Hente data fra databasen

For å strukturere dataen som er nødvendig for å vise antall produserte plater for hver klasse, må følgende fire steg gjennomføres:

1. Registrere modul og utføre spørring
2. Hente tidsbolker og klasse mellom start- og sluttid

3. For hver tidsbolk, gruppere etter tid og klasse, og telle antall plater
4. Formatere data

Før Grafana kan hente data gjennom API-et, må en modul registreres i modulkontrolleren. Grafana henter de registrerte moduler, og viser disse i en nedtrekksliste i panelene, som vist i figur 26.



Figur 26: Nedtrekksliste i Grafana.

For å gi et mer detaljert bilde av produksjonen over tid, er det nødvendig å dele opp hele tidsintervallet i mindre intervall. Størrelsen disse intervallene oppgis i Grafana gjennom `resolution` parameteret.

I steg 1 hentes alle klassene, og tidsintervallene innenfor start- og sluttiden. Disse tidsintervallene genereres ved å runde opp tidspunktene til hver plate, til nærmeste angitte `resolution`. Et eksempel på dette er hvordan `08:05:00` rundes opp til `08:30:00` ved 30 minutters intervaller.

Dette medfører at det bare genereres intervaller der det finnes minst 1 plate. Grafana forventer derimot at alle intervall-klasse par er tilstede i returnert data, uavhengig om antall plater er null i intervallet eller ikke. Dette gjør at alle mulige kombinasjoner av avdekkede klasser og tidsintervaller må genereres, se figur 27. Denne operasjonen kalles en cross join. Med alle radene fra steg 1, kan vi i steg 2 telle opp antall plater innenfor intervall-klasse parene[5]. Forskjellen i returnert resultat fra denne tellingen med og uten cross join kan sees i tabell 28.

Klasse	Tid	Antall plater
Godkjent	08:00	26
Reparer	08:00	0
Vrak	08:00	47

(a) Cross join

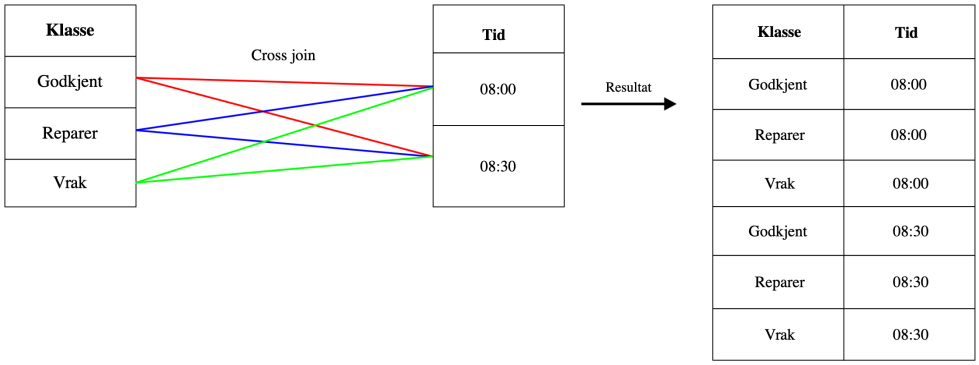
Klasse	Tid	Antall plater
Godkjent	08:00	26
Vrak	08:00	47

(b) Uten cross join

Tabell 28: Resultat med og uten cross join.

I steg 3 formateres dataen fra steg 2 til et tidsserieformat, som aksepteres av

[5]En unik kombinasjon av klasse og tid, se figur 27.



Figur 27: Cross join legger sammen alle ulike kombinasjoner mellom tabeller.

Grafana. Tidsseriedata består av en liste over tidspunkt med tilhørende tallverdi, og et navn for å beskrive disse. Dette formatet er beskrevet i listing 19.

```
{
    "navn": "str",
     "datapoints": [
        (verdi, tid),
        (verdi, tid)
    ]
}
```

Listing 19: Format til Grafana.

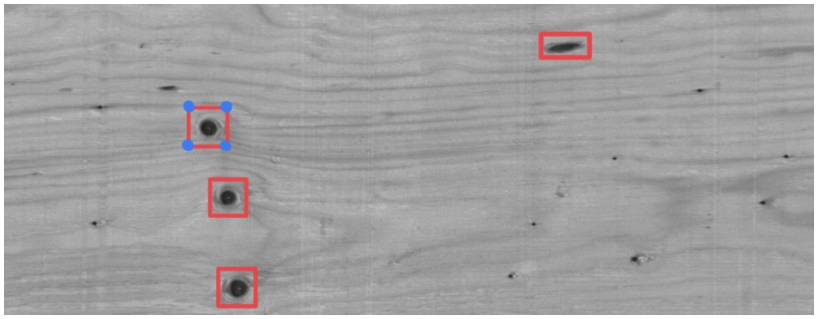
Dataen fra steg 2 konverteres til tidsseriedata ved å først samle alle rader med samme klasse. Deretter blir listen over datapunkter laget ved å hente ut tidspunktet for intervallet, og antall plater. Denne prosessen gjentas for hver returnerte klasse for å danne sluttresultatet.

6.5.2 Heatmap

Metoden for implementasjon beskrevet over er felles for resterende spørringer i Grafana, med unntak av heatmappet. Grafana tilbød ingen eksisterende paneltyper som kunne brukes for å implementere funksjonaliteten vi ønsket. Derfor har vi her brukt Seaborn, i kombinasjon med Grafana, for å best mulig realisere de satte kravene.

Målet med heatmappet er å raskt kunne identifisere trender av defekter. Dersom en ventil konstant drypper olje på platene, vil dette raskt kunne oppdages gjennom fargedistribusjon i et heatmap. For å visualisere denne konsentrasjonen, benyttet vi omrisset av hver individuelle defekt. Hver skanner lagrer informasjon om størrelsen på platen, samt fire parametere som beskriver et rektangulært

omriss av defektene. Med dette datagrunnlaget kunne vi beskrive hver plate i et koordinatsystem, hvor platens lengde og bredde representeres som henholdsvis x- og y-akser. Dermed kunne vi for hver plate plotte omrissene av dens defekter som i figur 28, og bruke grad av tetthet for å beskrive trendene.



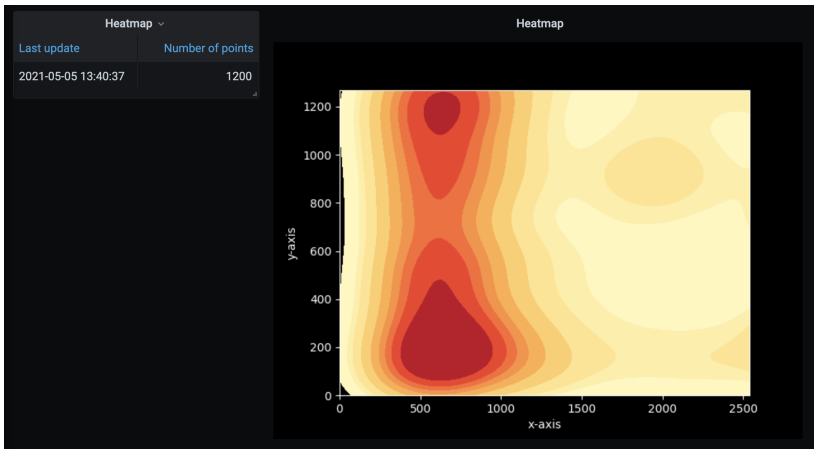
Figur 28: Eksempel på hvordan defektene blir definert.

For å visualisere denne konsentrasjonen av defekter, brukte vi Python-biblioteket Seaborn sitt Kernel Density Estimation (KDE)-diagram. Seaborn [36] er et Python bibliotek for datavisualisering basert på matplotlib [37]. KDE farger basert på konsentrajonen av punkter i et område. På denne måten blir områder med høy tetthet av defekter tydelig uthevet, noe som kommer tydelig frem i vårt eksempel, se figur 29. For å vise dette diagrammet i Grafana, gjøres det først tilgjengelig som et bilde gjennom webtjeneren. Deretter sender vi en forespørsel gjennom et AJAX-panel som henter og viser bildet i dashbordet. Da denne forespørselen er statisk, tolker ikke Grafana dette som et panel som skal oppdateres. Dette løste vi ved å implementere et sekundært panel som oppdateres på samme frekvens som resten av dashbordet. Panelet kunne dermed sende ny forespørsel til webtjeneren hver gang det ble oppdatert. Ved å la et tilhørende panel fungere som en kontroller, var det også mulig å legge inn ekstra parametere i spørringen mot API-et. I vårt tilfelle valgte vi å bruke dette for å bestemme hvor mange av de siste platene diagrammet skulle basere seg på. Dette gjøres gjennom panelkonfigurasjonen i Grafana, og er dermed enkelt for brukere å justere.

6.5.3 Konfigurere data i Grafana

Fra kravspesifikasjonen var det viktig at dasbordet kunne konfigureres. Grafana sitt brukergrensesnitt har mange muligheter for å justere utseende. Figur 30 og 31 viser hvordan man kan:

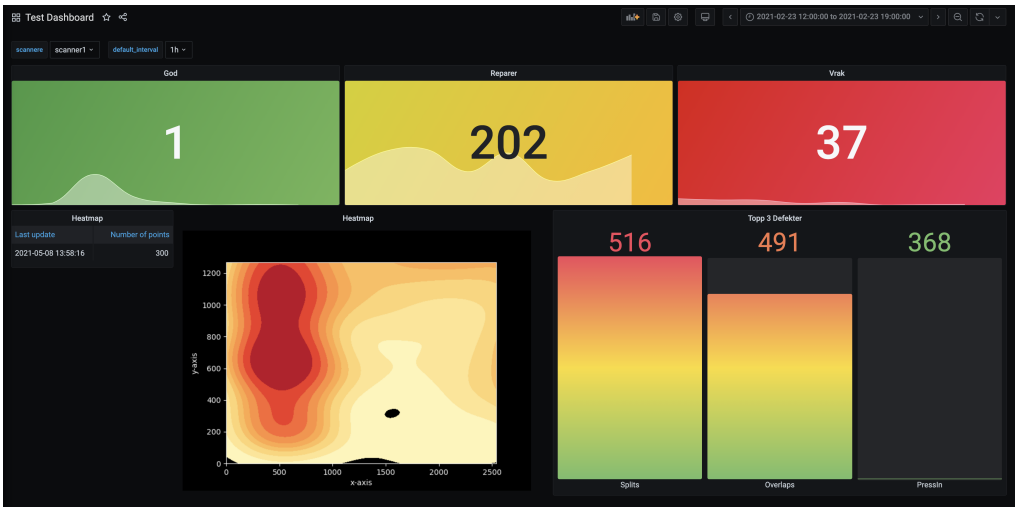
- Justere størrelsen på paneler
- Endre visualisering av data



Figur 29: Implementasjon av heatmap i dashbord.

- Endre terskelverdier, slik at farger på grafen justeres etter verdi
- Justere tidsbolker

Figur 30 viser antall plater per klasse på hele øverste rad, hvor klassene er representert som grønn, oransje og rød. Figur 31 viser de samme panelene, men nå med den historiske dataen i tillegg. Farger og tekststørrelse kan også endres. Dette viser hvordan Grafana legger til rette for universell utforming. Forskjellen i topp 3 defekter i figur 30 og 31 viser hvordan dashbordet kan skreddersys etter ønske.



Figur 30: Eksempel på utforming av dashbord av Grafana.

Figur 31: Grafana sin konfigurabilitet gjennom brukergrensesnittet.

7 Testing

I dette kapittelet beskriver vi hvordan systemet har blitt på forskjellige måter. Vi forklarer hvordan enhets- og integrasjonstestene er implementert, samt hvordan vi utførte grunnleggende manuell testing.

7.1 Pipeline

For å automatisk teste kildekoden har vi brukt Black, Pylint, enhetstesting og integrasjonstesting. For å forsikre oss om at disse verktøyene blir brukt, har vi etablert en pipeline der disse verktøyene blir kjørt når koden i BitBucket oppdateres. Dette gjorde at vi fikk automatisk beskjed dersom verkøyene ikke godkjente kildekoden.

Pipelinen består av en sammensetning av BitBucket og GitLab. Gjennom NTNU har vi tilgang til en GitLab instans med pipeline som vi, i motsetning til BitBucket Cloud, ikke trengte å betale for. Løsningen ble dermed å la GitLab kjøre pipelinen, mens BitBucket viser resultatet.

Instruksene til GitLab agenten oppgis gjennom `.gitlab_ci.yml`. Her defineres alle stegene, jobbene i hvert steg, avhengighetene mellom jobbene og hvilke resultater som skal lagres fra hver jobb. På grunn av kombinasjonen av GitLab og BitBucket, vil det siste steget vil alltid være å rapportere resultatet tilbake til BitBucket. Et utdrag fra pipelinen kan sees i listing 20.

```
1  stages:
2    - ci_status
3
4  success:
5    stage: ci_status
6    script:
7      - BUILD_STATUS=passed BUILD_KEY=push ./build_status
8    when: on_success
9
10 failure:
11   stage: ci_status
12   script:
13     - BUILD_STATUS=failed BUILD_KEY=push ./build_status
14   when: on_failure
```

Listing 20: Utdrag fra `.gitlab_ci.yaml`.

7.2 Automatisk testing

7.2.1 Kodekvalitet

For å sikre god kodekvalitet har vi benyttet hovedsakelig av to verktøy. For å sikre at koden er formattert på en standardisert måte har vi brukt Black [38]. Ved å følge denne standarden for kodeformattering vil det bli enklere for andre utviklere å komme inn i prosjektet, da koden er formattert på et kjent format. For statisk sjekking har vi benyttet oss av Pylint [39]. Dette er et verktøy som advarer utvikleren om for eksempel ubrukte variabler, manglende dokumentasjon, for komplekse funksjoner og så videre. Pylint har mange regler for utforming av kode. Noen av disse har ikke vært ønskelig å rette på i vårt prosjekt, så vi har fortalt Pylint at den skal ignorere disse. Ved å bruke dette verktøyet er vi sikre på at koden følger den etablerte Python stilstandarden PEP 8 [40], noe som også gjør det enklere for fremtidige utviklere å sette seg inn i kildekoden.

7.2.2 Enhetstester

I utviklingen har vi hatt fokus på å lage et robust rammeverk for fremtidig bruk. Vi har dermed også fokusert på å skrive tester til kjernen av dette rammeverket. Her har vi benyttet en kombinasjon av enhetstesting og interne integrasjonstester for å verifisere at de forskjellige delene internt i Python modulen fungerer sammen. I tillegg kjøres Coverage, som gir en pekepinn på hvor mye av koden som er dekket av testene.

Vi har skrevet testene i Pytest. Pytest krever minimalt med kode for å kjøre enkle tester, med valgfri ytterligere funksjonalitet dersom dette skulle være nødvendig. Det er mulighet for å initialisere tester gjennom fixtures [41]. Fixtures gjør at hver test hat et felles utgangspunkt. Det kan konfigureres at en fixture kjører enten for utvalgte tester, eller at den alltid skal kjøre. Listing 21 viser hvordan vi har implementert oppsettet av en testklient for API-testene. Her overstyres API-applikasjonen som benyttes i FastAPI, og avhengigheten som beskrevet i implementasjon. Dette gjør at Pytest slipper å koble seg opp mot en database for å kjøre, som øker testhastighet og reduserer kompleksiteten i testmiljøet betraktelig [42].

```
1  # test_connect.py
2  from package import connect
3  import pytest
4
5  @pytest.fixture(name="client", autouse=True)
6  def init_tests():
7      app = APIApp(
8          load_from_settings={
9              "reflect": False,
10             "debug": False,
11             "use_database": False
```

```
12          }
13      )
14
15      app.dependency_overrides[global_dependencies["get_db"]] =
            override_get_db
16      return TestClient(app)
17
18  def test_feature(client):
19      expected_result = {}
20      result = client.get("/")
21
22      assert result == expected_result
```

Listing 21: Minimalt Pytest eksempel.

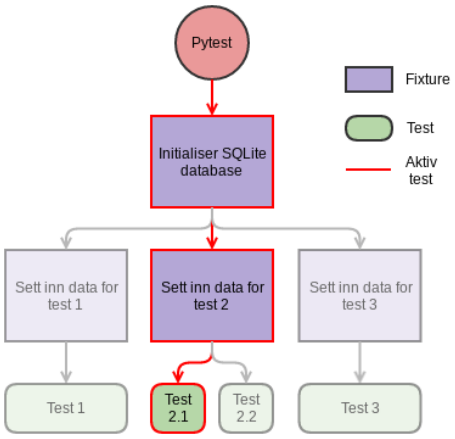
7.2.3 Integrasjonstesting

Der enhetstesting verifiserer hvordan individuelle deler av systemet fungerer, er det også viktig å teste at samspillet mellom de forskjellige delene fungerer som det skal. Dette gjøres ved hjelp av integrasjonstester. Det er mulig å gjøre integrasjonstesting i forskjellig skala, fra testing av grensesnittet mellom to Python funksjoner til å sette opp et komplett produksjonsmiljø for den mest reelle testingen. I vårt system har vi valgt å fokusere på en mellomting av disse ytterpunktene, der vi har skrevet API-tester for å teste dataflyten hele veien gjennom systemet.

For å teste denne dataflyten var vi avhengige av å ha en database i bakgrunnen for å gi data som resten av systemet kunne prosessere. Her hadde vi alternativene å enten sette opp en instans av en reell database, eller forsøke å fjerne avhengigheten til databasen fra koden under testing. Med databasetilgangslaget vårt hadde det vært mulig å fjerne avhengigheten av databaser totalt. Dette hadde lagt til rette for rask eksekvering av testene, men det hadde også medført at vi ikke kunne teste SQLAlchemy-spørringene i seg selv. Vi vurderte muligheten for å sette opp en testdatabase i pipelinen. Vi så derimot at dette ville medføre at pipelinen vil ta enda lengre tid å utføre, samt at det ville vært vanskeligere for utviklere å utføre testene lokalt.

Løsningen vi har implementert består av å utnytte SQLAlchemy sin styrke vedrørende hvordan spørringsstrukturen kan konverteres til mange forskjellige databasedialekter. En av disse dialektene er SQLite. SQLite er en lettvekts databaseløsning som ikke krever en dedikert tjener, men lagrer heller innholdet i databasen rett i filsystemet. Filene kan deretter aksesseres gjennom et forhåndsdefinert grensesnitt rett fra Python. Ved å bruke SQLite som database kan testene kjøres både lokalt og i pipeline, samtidig som at utføringen kan skje raskt. Ulempene ved å bytte databasedialekt er at noen detaljer kan være forskjellige mellom dialektene, men vi har ikke funnet at dette er et problem med nåværende løsning.

For at SQLite skal kunne benyttes som database, må en instans forberedes før testene kan gjennomføres, og data må legges inn. Dette blir gjort av gjentatt bruk av Pytest sine fixtures. I den første fixturen settes tilkoblingen til SQLite opp. For å sette opp de nødvendige tabellene benyttes SQLAlchemy sin ORM-struktur. Her defineres tabellene i form av Python klasser, som SQLAlchemy deretter reflekterer inn i databasen. Nødvendige databaseobjekter blir deretter lagt inn i den eksisterende databasehåndteringsklassen slik at den kan aksesseres på normalt vis i systemet. Den andre fixturen kan deretter legge inn nødvendig data for spesifikke tester, og videre benyttes av en eller flere tester. Figur 32 viser hvordan Pytest setter opp databasen og setter inn data spesifikt for test 2.1. Kode for hvordan denne kjeden er konstruert kan sees i vedlegg R.



Figur 32: Sammenheng mellom Fixtures og tester.

7.2.4 Automatiske tester i pipeline

For å kjøre Black, Pylint og Pytest i pipeline la vi de til som to separate steg, der ett kjører Black og Pylint, mens den andre kjører testene. For å spare tid har vi benyttet Shortest Job First [43] algoritmen fra *scheduling* i operativsystemer. Dette innebærer å dele inn stegene slik at de jobbene som bruker kortest tid, i dette tilfellet kodekvalitetstestene, kjører først. På denne måten unngår vi å vente på resultatet fra mer omfattende tester, dersom pipelinen uansett ikke hadde blitt godkjent. I listing 22 har listing 20 blitt utvidet med de nye stegene.

```
1 stages:
2   - ci_status
3   - linting_and_formatting
4   - testing
5
6 black:
7   stage: linting_and_formatting
8   script:
```

```
9       - black --check ArgosAnalysePlatform tests
10
11  pylint:
12    stage: linting_and_formatting
13    script:
14      - pylint ArgosAnalysePlatform
15
16  unit-tests:
17    stage: testing
18    script:
19      - export PYTHONPATH="$PYTHONPATH:."
20      - pytest
21
22  success:
23      ...
24
25  failure:
26      ...
```

Listing 22: Pipeline med automatiske tester.

7.3 Manuell testing

For å teste koden under utvikling og demonstrere funksjonalitet til Argos og veileder, satt vi opp to replikaer av databasen til en Argos scanner på NTNU i Gjøvik sin interne sky, *SkyHigh*. Vi fikk hjelp av Argos til å installere nødvendig programvare på to instanser av Windows Server 2016 for å replikere deres interne testmiljø. Det ble deretter lastet unike datasett på hver scanner. Det ene datasettet besto av data som kontinuerlig ble generert av et script som kjørte på samme maskin. Det andre datasettet er et utdrag fra reell produksjonsdata, men anonymisert for å beskytte eieren av dataen. Denne kombinasjonen av datasett ga oss mulighet til å vise hvordan rapportene og dashbordene ville sett ut for en reell kunde, og samtidig teste hvordan systemet håndterte større datamengder.

8 Installasjon

I dette kapittelet vil vi gå nærmere inn på hvordan systemet kan bygges, og hvordan man kan benytte støtteverktøyene Swagger og Sphinx. Til slutt beskrives hvordan skymiljøer skiller seg fra lokal utrulling med tanke på ytelse, risiko og kostnad.

8.1 Bygging

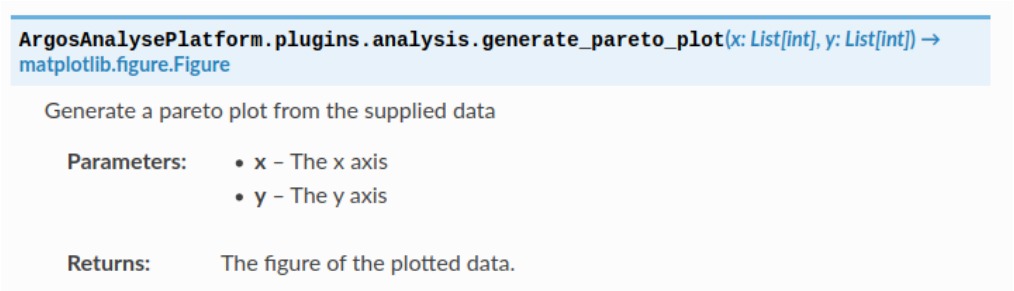
Som nevnt i implementasjon er rammeverket utformet som en Python pakke. Denne pakken administreres gjennom Poetry. Poetry er et verktøy som hjelper utviklere håndtere pakkers avhengigheter og konfigurasjon.

Når en pakke er klar for utrulling, vil Poetry konvertere pakken med `poetry build`. Resultatet av dette kan deretter installeres direkte på andre maskiner med `pip install (...)` eller lastes opp til en package manager. Når pakken er installert kan API-et startes med `python3 -m ArgosAnalysePlatform (...)`

Denne byggeprosessen kan gjøres i GitLab pipelines på samme måte som linting og testing fra kapittel 7. Byggingen har blitt lagt som et ekstra steg til slutt. Dette medfører at bygging ikke gjøres med mindre prosjektet ellers har god kvalitet. Listing 23 beskriver hvordan `.gitlab_ci.yaml` må utvides for å oppnå dette.

```
stages:
  - linting_and_formatting
  - testing
  - build
  - ci_status

# ...

build:
  stage: build
  only:
    - master
  before_script:
    - *python_before_script
  script:
    - poetry build
  artifacts:
    paths:
      - dist
    expire_in: 2 days
```

Listing 23: Bygging av Python pakke i GitLab pipeline.



Figur 33: docstring eksempel i HTML-format .

8.2 Dokumentasjon

Rammeverket i analyseplattformen er dokumentert ved hjelp av verktøyet Sphinx. Dette er et verktøy med tilhørende formateringsstandard som kan generere brukervennlig dokumentasjon basert på kildekoden direkte [44]. Fordelene med at dokumentasjonen ligger direkte i kildekoden er at den er enklere å holde vedlike, som gjør at den er enklere å benytte av nye og erfarne utviklere [45].

For å tolke dokumentasjonen korrekt må den formateres etter Sphinx sin standard. I listing 24 vises noen utvalgte nøkkelord.

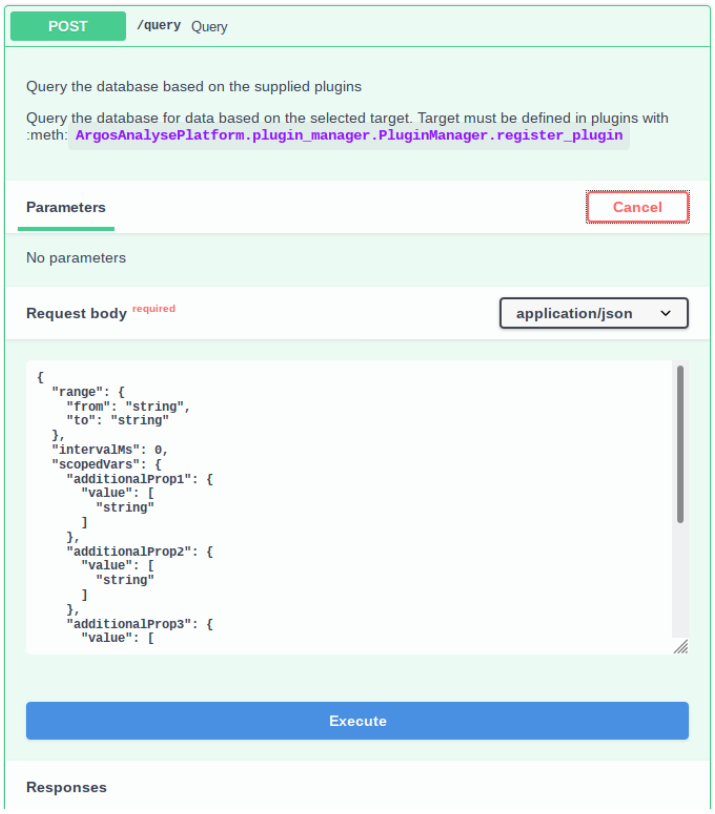
```
1 def generate_pareto_plot(x: List[int], y: List[int]) -> plt.Figure:
2     """Generate a pareto plot from the supplied data
3
4     :param x: The x axis
5     :param y: The y axis
6
7     :return: The figure of the plotted data.
8     """
9     ...
```

Listing 24: Docstring-eksempel.

Sphinx vil tolke denne docstringen og konvertere den til et brukervennlig format som kan sees i figur 33.

Analyseplattformens pipeline er satt opp til å generere Sphinx-dokumentasjonen i samme stadie som byggingen av Python pakken. Dette gjøres ved å legge inn den ekstra konfigurasjonen i listing 25.

```
1 docs:
2   stage: build
3   before_script:
4     - *python_before_script
5   script:
6     - sphinx-apidoc -o sphinx/source/ ArgosAnalysePlatform/ -eFa -A
          Argos
7     - cp sphinx/conf.py sphinx/source/conf.py
8     - make html -C sphinx/source
9
```



Figur 34: Swagger representasjon av /query .

```
10  artifacts:
11    paths:
12      - sphinx/source/_build/html
```

Listing 25: Bygging av Sphinx dokumentasjon.

For å dokumentere endepunktene til API-et har FastAPI inkludert Swagger [46]. Swagger er et verktøy som genererer dokumentasjon for hvert API-endepunkt i et web-grensesnitt, og gir brukeren mulighet til å gjøre spørringer mot endepunktene rett i nettleseren. Dette gjør det enkelt for utviklere å se utfallet av spørringer med forskjellige parametre, uten å måtte bruke andre verktøy. Figur 34 viser et eksempel på dette for `/query` endepunktet.

I tillegg til å lese docstrings av dataklassene og endepunktene som FastAPI tjener, vil også Swagger vise ekstra dokumentasjon som er spesifisert i FastAPI parametre, som for eksempel `start_time: str = Form(..., description="Start time in isoformat")`. For å unngå å repetere den samme informasjonen flere ganger har vi antatt at brukeren leser dokumentasjonen i Sphinx, og referert til at det finnes mer i Swagger.

8.3 Produksjonssetting i sky

Argos ønsket at Koios skulle kunne implementeres både på en lokal tjener, men også i skyen. Dette medførte at Argos ønsket at vi gjorde en grunnleggende undersøkelse der vi sammenlignet lokale og skyløsninger med tanke på ytelse, risiko og kostnad. Hele undersøkelsen kan sees i vedlegg E.

I undersøkelsen satt vi 'Hvordan påvirker faktorene ytelse, risiko og kostnad en migrering fra lokal infrastruktur til sky?' som innledende problemstilling. For denne undersøkelsen var det ikke ønskelig fra Argos at vi gikk i dybden, men heller fokuserte på de overordnede vurderingene. Vi satt derfor en grense på 2 uker til å utføre undersøkelsen, da arbeidet raskt kan øke i størrelse og omfang.

Ved å undersøke akademiske rapporter og statlige utredninger, fant vi at skytjenester har en fordel over lokal infrastruktur med sin dynamiske ytelse, men har svakheter vedrørende krav om pålitelig internettforbindelse. Skyplattformer stiller også sterkere med tanke på risiko, da leverandørende ofte tilbyr sin sikkerhetsekspertise som en tjeneste. Det er ikke tydelige tegn på at infrastruktur som er rullet ut i skyen er mer utsatt for sikkerhetshendelser enn tilsvarende utrulling lokalt. Kostnadsmessig er det også en fordel å kjøre i skyen, da forretningsmodellen legger opp til at man bare betaler for nøyaktig de ressursene som benyttes, i motsetning til å måtte investere i maskinvare.

9 Sikkerhet

Under utvikling av et system er det viktig å tenke på sikkerhet gjennom hele prosjektløpet. Det er mye enklere å avdekke og rette på sikkerhetsproblemer dersom de avdekkes tidlig i prosjektfasen, i motsetning til etter systemet har blitt satt ut i produksjon [47]. Vi har gjennom hele prosessen benyttet Confidentiality, Integrity, Availability (CIA)-prinsippene for å best mulig ivareta sikkerhet i valgene av teknologi, design og implementasjon.

9.1 Konfidensialitet

Konfidensialitet handler om at uvedkommende ikke skal få tilgang til sensitiv informasjon. Dette er viktig for Argos, da deres kunder er veldig beskyttende ovenfor lekkasje av informasjon om hva som produseres, og hvordan produksjonen går i deres fabrikker. Konfidensialitet ivaretas i all hovedsak i et system ved å avgrense brukertilganger til det minst nødvendige for formålet og sikre at alle relevante datakilder krever autentisering og autorisasjon.

I systemet vårt har vi lagt til rette for beskyttelse av konfidensialitet i API-et og databasetilgangslaget. Vi har ikke implementert en konkret autentiseringsmekanisme. Implementasjon av dette kan gjennomføres ved å benytte seg av samme konsept for avhengigheter, som API-et benytter seg av for tilgang til databaseklassen. Autentisering kan implementeres enten ved hjelp av sin egen løsning, eller gjennom et av FastAPI sine klasser for Basic eller OAuth2 [48]. Gjennom datastrukturvalideringen som nevnt i implementasjonen av rammeverket vil eventuelle data som kommer inn og går ut bli validert, slik at ukjente datastrukturer eller datatyper blir stoppet. SQLAlchemy sitt rammeverk for definisjon av spørringer legger til rette for trygg databasekommunikasjon. Dette innebærer at all data fra eksterne kilder blir validert, slik at risikoen for SQL-injeksjoner blir sterkt redusert.

9.2 Integritet

Integritet handler om at dataen som ligger i databasen er korrekt og intakt. Det vil være viktig for kundene til Argos at datagrunnlaget for dashbordet og rapportgenerering er korrekt. Dersom dette ikke er tilfellet, kan det potensielt tas feil valg om endring i produksjonen. Integritet opprettholdes ved å sikre seg at dataen som ligger i databasen ikke har blitt endret i etterkant. Det er også viktig at data som er i bevegelse, i tillegg til lagret stille på harddisk, ikke kan bli endret på. Systemet vårt håndterer ikke permanent lagring av data, og derfor var det ikke mulig for vårt system å sikre mot tap av produksjonsdata.

Vi har lagt til rette for at systemet skal kunne ta vare på disse kravene i database-

tilgangslaget. Gjennom API-ets datavalidering og SQLAlchemy sin beskyttelse mot SQL-injeksjoner, er sannsynligheten for at data kan endres på gjennom vårt system betydelig redusert. Sikring av data i bevegelse har hatt mindre fokus under utviklingen, da de viktigste tiltakene for dette gjøres i sammenheng med utrullingen. Hovedtiltaket for å beskytte mot slike angrep er bruk av HTTPS i all web-kommunikasjon.

9.3 Tilgjengelighet

Tilgjengelighet handler om at systemet er tilgjengelig for brukeren når de trenger det. Avbrudd kan bli problematisk for kundene til Argos da de vil benytte seg av systemet vårt i sin daglige drift for å få en oversikt over produksjonen. Dersom vårt system benyttes av mange dashbord samtidig kan ytelse bli et problem, da oppdateringer vil oppleves tregt. Vi anser samtidig ikke disse faktorene som kritisk for den kontinuerlige driften til kundene, da nedetid i vårt system i liten grad vil påvirke selve produksjonslinjen.

Vi har lagt til rette for tilgjengelighet ved å utvikle et stabilt system. Gjennom FastAPI sin bruk av Uvicorn er det mulig å spesifisere flere prosesser for å håndtere mange innkommende spørringer samtidig. Gjennom modulkontroller-modellen kan det også legges opp til at moduler legges til og fjernes fortløpende, for å unngå at hele API-tjeneren må restartes.

Generelt sett er det gjennom valg av design og teknologier utøvd sikkerhetstiltak i hvert lag i arkitekturen. Dette medfører økt motstandsdyktighet hele veien fra API til database, og tilbake ut. I systemet er det enten implementert sikkerhets-tiltak, og der det ikke er implementert er det lagt opp til enkel implementasjon i fremtiden.

10 Drøfting

I dette kapittelet reflekterer vi rundt resultatet vi har oppnådd og hvordan utviklingsprosessen har vært.

10.1 Resultat

10.1.1 Fellesplattform

Når vi ser tilbake er gruppen fornøyd med hvordan fellesplattformen har blitt utviklet. I designfasen ble det tatt gode valg om struktur som la godt til rette for funksjonalitet som ble oppdaget senere i utviklingsprosessen. Vi er også godt fornøyd med hvordan designet ble implementert.

Teknologivalgene vi tok la godt opp til dette. Vi vurderer Python som et godt valg på grunn av faktorene som nevnt tidligere, men i etterkant ser vi at gruppens eksisterende kunnskap om språket var uvurderlig da andre aspekter av oppgaven tok mer tid enn vi forventet. Dette gjorde at gruppen heller kunne fokusere på dette enn å lære seg et nytt programmeringsspråk. SQLAlchemy og FastAPI lot seg godt kombinere, både med hverandre og resten av plattformen. Dette gjorde at utviklingen av fellesplattformen ikke traff på mange hindre underveis.

Valget av arkitektur endte opp med å tilrettelegge for all funksjonaliteten vi hadde behov for. Denne planleggingen gjorde at vi slapp å gå tilbake for å gjøre større strukturelle endringer i etterkant. Vi føler at den helhetlige modellen vi kom frem til er enkel å forstå, mens den samtidig gir stort rom for potensielle utvidelser i fremtiden.

Vi er også fornøyd med hvordan fellesplattformen har blitt implementert. Ved å strukturere plattformen som en Python pakke ga det oss gode muligheter for å kunne kjøre den i alle slags miljøer. Dette viste seg å være fordelaktig allerede i utviklingsprosessen, da alle tre gruppemedlemmene benyttet seg av hvert sitt operativsystem i prosjektperioden. Dette kombinert med et enkelt kommandolinjegrensesnitt gjorde at det også er enkelt å starte API-tjeneren i et produksjonsmiljø.

Modulkontrolleren endte opp med å bli en teknisk enkel implementasjon, men vi er veldig fornøyd med løsningen for hvordan vi detekterer og laster moduler. Ved å både kunne laste moduler fra internt i pakken, men også en dynamisk ekstern mappe, gjorde at vi fikk en løsning som er enkel i bruk både for Argos' ansatte og deres kunder.

Gruppen er fornøyd med hvordan databasetilgangslaget har blitt implementert. Her har vi benyttet oss av SQLAlchemy for selve grovarbeidet, som gjorde at vi kunne fokusere på hvordan dette kunne kobles sammen med Argos' behov. Ved

å gi hver modul tilgang til en SQLAlchemy sesjon direkte, gir det abstraksjon i forhold til hvordan dataen ligger lagret i databasen. Dette gjør den samtidig som at modulene har full tilgang til å skreddersy spørringer. Samtidig som at dette er en fordel, ser vi også at vi ikke har implementert et like tydelig skille mellom databaselaget og resten av plattformen i forhold til designet vi satt. Når vi ser tilbake på implementasjonen ville vi laget hver spørring som en metode i klassen for databasehåndtering, og latt den holde styr på SQLAlchemy sesjonsobjektet, og heller angitt hvilken database som skulle hentes fra gjennom et parameter til disse metodene. Vår eksisterende løsning sender allerede databasenavnet til modulene, så denne endringen vil ikke være stort arbeid å implementere.

Generelt sett er vi fornøyd med omfanget av fellesplattformen, men i ettertanke ser vi at vi gjerne kunne gjort mer på områdene testing og utrulling. Testene vi har skrevet frem til nå er enhetstetster og lokale integrasjonstester. Vi ser at vi hadde hatt fordel av å lage mer omfattende integrasjonstester som inkluderte å sette opp faktiske instanser av databasen og Grafana for å teste reell integrasjon med de eksterne tjenerene. Dette la vi spesielt merke til når vi, etter noen oppgraderinger i fellesplattformen, oppdaget problemer i Grafana. Vi har satt opp en pipeline for å bygge prosjektet, men vi hadde ikke tid til å sette opp bygging av Docker-bilde.

I starten av prosjektet fikk vi tilgang til programvare for å installere en tilnærming av en skanner på våre egne maskiner. Dette tok litt å sette opp, men var vel verdt tiden, da vi fikk muligheten til å vise en mer reell tilnærming av hvordan systemet vårt ville fungert i produksjon

Vi har ikke implementert noen eksplisitte sikkerhetstiltak i vårt system, men vi føler at vi har ivaretatt noen av de største sikkerhetsutfordringene gjennom våre valg av teknologier og design. FastAPI og SQLAlchemy har begge gode vilkår for sikker programvareutvikling.

10.1.2 Simuleringsverktøy

Simuleringsverktøyet var noe helt nytt både for oss og for Argos. Dette medførte at det tok en del tid i starten å avdekke kravene til hva dette verkøyet faktisk skulle innebære. Et eksempel på dette er hvordan det i oppgavebeskrivelsen står at '*Dersom man for eksempel øker en toleranse fra 5 til 6mm (...)*'. Simulering av denne type data viste seg å ikke være mulig med datagrunnlaget som lå i databasen. Dermed ble kravet, i samarbeid med Argos, endret til å være basert på antallet feil av hver type, istedenfor karakteristikaene til feilene direkte.

Brukergrensesnittet til simuleringsverktøyet er gruppen stort sett fornøyd med. I starten av prosjektet planla vi å skrive grensesnittet i stort sett standard JavaScript, med jQuery for å hovedsakelig gjøre AJAX-kall mot API-et. Dette første utkastet førte til kodekvalitet som gruppen ikke var fornøyd med i det hele tatt.

Dermed ble valget tatt å skrive om hele nettsiden etter SPA modellen, som kunne erstatte de delene av koden vi var minst fornøyd med.

Vi ser i etterkant at Vue var et godt valg. Omskrivingen til Vue ga et produkt som ga brukeren et godt inntrykk av verktøyet. I ettertid ser vi at det definitivt gjenstår arbeid i selve utseendet til verktøyet, men på bakgrunn at vi ikke skulle levere produksjonsklare produkter tenkte vi at det var riktig å ikke fokusere for mye på dette. I tillegg deres til eksisterende ønske om simulering ved å sette inn terskelverdier selv, kom også gruppen med et forslag om automatisk forslag av terskelverdier. Dette ble godt mottatt av Argos.

Vi er fornøyd med hvordan selve simuleringsprosessen ble implementert. Selv med endringene i kravene om hvilke data som skulle simuleres på var dette fortsatt en komplisert algoritme å utvikle. SQLAlchemy sin måte å skrive spørringer på ved å splitte det inn i kjedbare funksjonskall hjalp oss å strukturere sluttresultatet på en måte som gjør det enklere for andre utviklere å forstå. I etterkant ser vi at algoritmen muligens er noe spesialisert til nøyaktig de kravene vi hadde i denne oppgaven, og at visse utvidelser av logikken kan bli komplisert å legge til i fremtiden.

10.1.3 Dashbord

Vi er godt fornøyd med hvordan dashbordløsningen har blitt satt opp. I ettertid ser vi at Grafana har vært et godt valg for å dekke både de funksjonelle og ikke-funksjonelle kravene fra Argos. I tillegg til å ha brukt Grafana sine allerede innebygde funksjoner har vi også utvidet det med heatmap. Vi er generelt fornøyd over vår representasjon over hva Grafana kan tilby av visualiseringsløsninger.

Vi er også fornøyd med hvordan vi valgte å implementere metoder fra fagfeltet User centered design (UCD) i prosjektet. Brainstorming var til stor nytte da vi skulle finne ut hvordan dataen skulle visualiseres, da det var mange måter problemet kunne tolkes på. En annen metode som ble brukt i forberedelsen før intervjuet med salgsteamet var Crazy8 [49]. Metoden tvingte oss til å komme med nye ideer, og på den måten kunne vi forsikre oss om at vi stilte så mange av spørsmålene vi ønsket svar på som mulig.

Utviklingen av panelene har til tider vært noe utfordrende, men vi føler at vi har løst dette på en god måte. Mens fellesplattformen var under utvikling benyttet vi oss av Grafana sin innebygde støtte for å kjøre spørringer rett mot SQL-databasen. Dette gjorde at vi kunne jobbe med fellesplattformen parallelt med prototyping av Grafana, som sparte oss verdifull tid. Dette ga også en fordel for oss da våre SQL kunnskaper måtte friskes opp. Ved å skrive spørringene i 'rå' SQL først, hadde vi et godt utgangspunkt når disse spørringene senere ble implementert i SQLAlchemy.

Vi er fornøyd med vår implementasjon av heatmap, men i etterkant ser vi at

en bedre løsning hadde vært å lage en Grafana modul for heatmap, istedenfor å benytte seg av et bilde som lastes gjennom AJAX. Dette hadde resultert i økt brukervennlighet, da kontrollpanelet kunne bygges rett inn i Grafana, istedenfor å ha et sekundært panel.

I løpet av prosessen hadde vi lyst til å gjennomføre brukertesting av dashbordet med ansatte hos Argos. Dette fikk vi derimot ikke tid til, da vi måtte prioritere andre deler av løsningen. Vi anser ikke dette som kritisk, da vi har lagt opp til høy fleksibilitet i utformingen.

Et av de initielle produksjonsmålene vi ønsket å lage var en oversikt over linjehastighet. I etterkant fikk vi vite at andre systemer i produksjonslinjen måler dette. Vi valgte derfor å nedprioritere dette panelet, da vi ønsket å fokusere på annen funksjonalitet.

10.1.4 Rapportløsning

Gjennom hele prosessen har vi møtt på flere utfordringer vedrørende rapportgenereringen. Dette var den eneste delen av systemet der Argos hadde et eksisterende system, som enten skulle videreutvikles eller erstattes fullstendig. Dette gjorde at arbeidet vårt ble delt i to hoveddeler. Det første var å finne ut hva som var galt med eksisterende løsning, og videre finne ut hva de ønsket i en ny løsning. Den andre var å finne ut av hva slags produkt gruppa skulle levere, med de begrensningene vi hadde i ressurser.

Det eksisterende systemet er gammelt, og benytter seg av utdatert teknologi. I tillegg har dokumentasjonen om hvordan systemet fungerer gått tapt. Dette gjorde det vanskelig for oss å finne ut eksakt hvordan det eksisterende systemet fungerte. Formålet med dette arbeidet var å danne et grunnlag for hvordan rapportgenereringen kunne videreutvikles for å møte dagens behov. Det ble dermed brukt en del tid på å avdekke disse detaljene.

I startfasen av prosjektet utforsket vi flere potensielle løsninger for hvordan vi kunne lage et system for generering av rapporter. Det første alternativet vi vurderte var å benytte Grafana sitt innebygde system for å vise dashbordene sine i PDF-format. Dette verktøyet viste seg derimot å være under utvikling, slik at det ikke ble anbefalt av Grafana å benytte dette. Vi fant deretter et åpent verktøy som hadde tilsvarende funksjonalitet, og startet å tilpasse dette for vårt behov.

Parallelt med dette arbeidet fant vi ut at den mest sannsynlige årsaken til at eksisterende system opplevdes særdelig frustrerende, var at det ble brukt på en måte som ikke var ment da systemet ble laget for mange år siden. Dette faktumet satt oss i et dilemma. Dersom det viser seg at systemet som det sto idag faktisk dekket behovene, var det i bacheloroppgaveperspektiv lite nyttig for oss å fortsette å videreutvikle en ny løsning. Vi så det heller ikke formålstjenelig å

fortsette å bruke tid på å avdekke dokumentasjon for hvordan den eksisterende løsningen fungerte, da dette heller ikke bidrar til en god oppgave for vår del. Alternativet var da å legge fra oss hele rapportgenereringsløsningen og fokusere på de to andre delene av systemet. Dersom vi gikk for dette alternativet var vi samtidig litt bekymret over om omfanget av oppgaven vår ville blitt smal. Etter mye intern diskusjon og samtaler med Argos ble det i tilegg avdekket at grunnen til at de ikke hadde oppgradert databaseløsningen sin fra SQL Server 2012 var på grunn av denne eksisterende rapportløsningen. Dette var en begrensing for Argos, da denne versjonen ikke lengre er offisielt støttet av Microsoft. Vi ble derfor enige med Argos om at målet til rapportløsningen skulle endres til å lage et proof of concept for hvordan løsningen kunne oppgraderes til nye versjoner av Microsoft verktøyene, slik at Argos kan oppgradere eller bytte ut databaseløsningen sin på sikt.

Gruppen begynte raskt arbeidet med å utforske hvordan Report Builder 2019 kunne benyttes for å generere rapporter fra databasen. Her avdekket vi at det var mulig å benytte seg av XML som datakilde. Dette var fordelaktig for vår løsning, da dette tillot at vi kunne benytte oss av vår fellesplattform for data. Med dette kunne Report Builder benytte seg av data både fra den gamle databasen, samtidig som at det tillot fremtidige endringer i databaseversjon eller type. Ulempen med denne løsningen var at Report Builder sin integrasjon ikke var hovedfokuset til Microsoft når de utviklet tjenesten. Dette medførte at det til tider var frustrerende å arbeide med Report Builder sitt brukergrensesnitt, samt at dokumentasjonen om hvordan integrasjonen fungerte er manglende. Dette medførte at vi måtte bruke en god del tid på å skjønne hvordan integrasjonen fungerte, og hvordan dette kunne tilpasses for å løse våre behov.

I retrospekt innser vi at vi kunne spart en del tid i startfasen av prosjektet ved å tidligere grave dypere i hvordan eksisterende system faktisk fungerte, og hva som var de store utfordringene til Argos. Dette hadde gjort at vi slapp å bruke tid på generering av PDF gjennom Grafana, da dette arbeidet ikke ga oss noe verdi til slutt. Vi brukte mye tid på å finne ut av hvordan Report Builder kunne hjelpe oss, men når vi ser tilbake tror vi at dette allikevel var et godt valg da vi er fornøyd med hvordan sluttresultatet.

10.2 Prosess

10.2.1 Gjennomføring av utviklingsmodell

Overordnet sett er gruppen fornøyd med valget av utviklingsmodell og gjennomføringen av denne. Vi er spesielt fornøyd med hvordan vi avholdt de retrospektive møtene. Gjennom hele prosjektløpet bidro disse møtene til å tidlig avdekke forbedringsmomenter og utforme mitigerende tiltak. Samtidig ser vi at det helhetlig ville vært hensiktsmessig å prioritere mer tid til å gjennomgå hva som ble gjennomført i den foregående perioden, i motsetning til prosessen. Dette

på bakgrunn av at vi ved noen anledninger mistet oversikten over hva som hadde blitt gjennomført og ikke, i etterkant.

Vi anser Kanban som fordelaktig, da det ga oss fleksibiliteten til å omprioritere midt i en sprint, noe som strider med tradisjonell Scrum. Dette viste seg fordelaktig, spesielt i rapportdelen av prosjektet, hvor vi valgte å omprioritere ressurser fra simulering til rapport for å opprettholde progresjon. Samtidig var håndtering av issues utfordrende i perioder. Bruk av Kanban-tavlen ble noe neglisjert mot slutten, noe vi i ettertid anser som et resultat av vårt tette samarbeid. Motivasjonen var dermed ikke like stor for å opprettholde tavlen, da vi mesteparten av tiden visste godt hva de andre jobbet med.

10.2.2 Planlegging

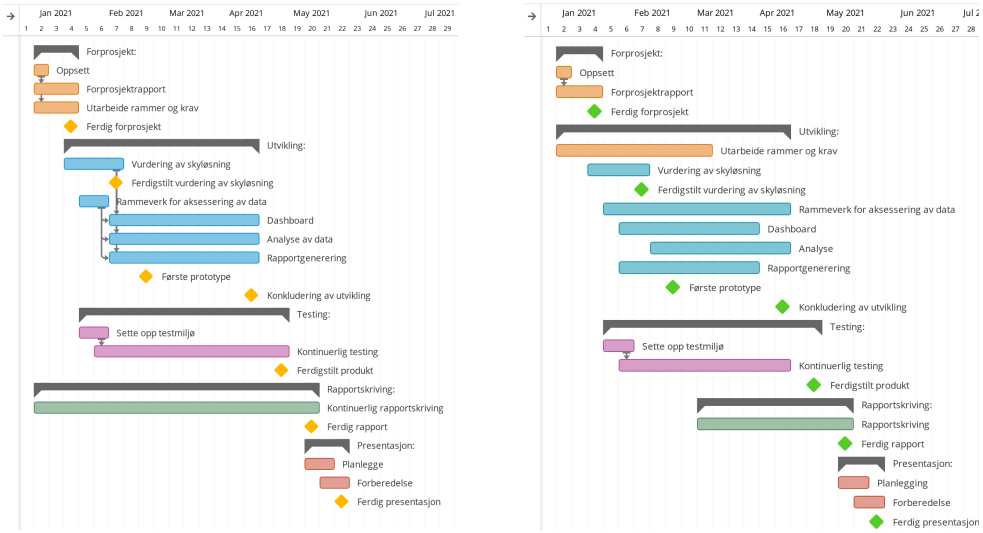
Vi brukte mye tid på planlegging, spesielt i begynnelsen av prosjektet, noe som reflekteres av hvor likt vårt initielle Gantt-diagram er vår faktiske gjennomføring (se figur 35). I ettertid er vi svært fornøyd med at vi prioriterte planlegging så høyt som vi gjorde, da vi gjennom prosjektperioden har unngått å måtte gjøre større endringer basert på feilberegninger av arbeidsmengde.

Den største forskjellen mellom Gantt-diagrammene er rapportskriving. Vi ser i etterkant at å begynne med rapportskriving tidligere hadde vært en fordel. Det hadde medført til at deler av utviklingsprosessen kunne blitt beskrevet i mer detalj, da det er lettere å skrive om valgene samtidig som de ble tatt. Dette løste seg allikevel fint, på bakgrunn av vårt fokus på dokumentering. Det var enkelt å henvise seg til dokumentasjonen i etterkant for å se hva som ble gjort og når. Den andre fordelen vi ser ved å begynne skrivingen tidligere er at vi kunne fått tilbakemelding fra veileder gjennom hele perioden.

Selv om vi hovedsakelig anser planleggingen som et positivt aspekt ved vår gjennomførelse, ser vi samtidig at det ved noen anledninger ble gjort i overdreven grad. Dette medførte til noe ineffektiv bruk av tid, i et forsøk på å planlegge ethvert aspekt ved en arbeidsoppgave.

10.2.3 Samarbeid og fordeling av arbeid

Overordnet har samarbeidet og fordelingen av arbeid innad i gruppen fungert godt. Selv om vi valgte å dele inn arbeidet innledningsvis, var planen fra start å samarbeide så mye som mulig, noe vi også har gjort. Denne beslutningen er vi veldig fornøyd med i ettertid, da vi ved flere anledninger har hjulpet hverandre videre dersom en av oss sto fast. I ettertid ser vi også nytteverdien av punkt 7 i gruppereglene (se vedlegg B), hvor en enstemmig beslutning skulle tilstrebes ved uenigheter. Denne regelen var spesielt relevant for oss, da antallet medlemmer i gruppen var oddetall. I stedet for at en demokratisk beslutning ble tatt, som i og



(a) Initielt diagram **(b)** Faktisk gjennomføring

Figur 35: Sammenligning av Gantt-diagrammer.

for seg betyr at noen må "tape", fortsatte vi diskusjonene til vi kom til enighet. Ved

flere anledninger erfarte vi at dette hadde positiv innvirkning på gruppedynamikken i kjølvannet av en diskusjon.

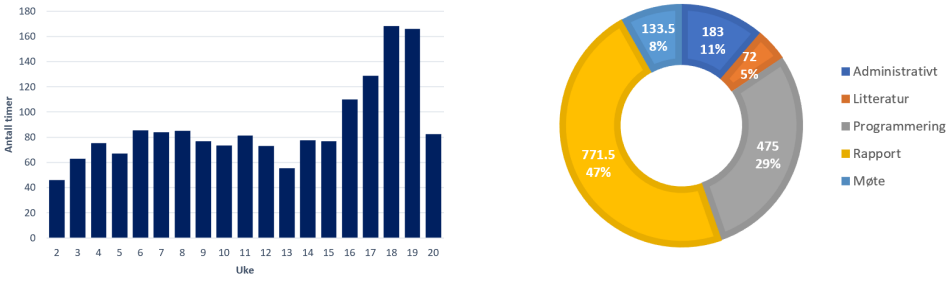
Samtidig var ikke samarbeidet feilfritt. Det oppsto spesielt en situasjon som bemerket seg ved at den gjorde samarbeidet mer utfordrende. Vi avdekket tidlig at samtlige på gruppen opplevde spørsmål fra de andre midt i en økt som distraherende. Ved å måtte forlate sitt eget fokusområde for å svare på et spørsmål, ble det vanskelig å returnere uten at tid gikk tapt i prosessen. I ettertid innser vi at denne situasjonen kanskje kunne vært unngått dersom vi var fire stykker på gruppen, da vi kunne jobbet i par.

Denne situasjonen ble diskutert på vår andre retrospektiv (se vedlegg F), og illustrerer nytteverdien av disse møtene. Mitigerende tiltak ble umiddelbart diskutert og det ble dermed foreslått å prøve ut Pomodoro-teknikken [50]. Gjennom tidsbolkingen satt vi tydeligere skiller mellom når vi stilte hverandre spørsmål og ikke. Dette skulle vise seg å være en stor suksess, og effekten dokumenteres allerede i neste retrospektiv (se vedlegg G).

10.2.4 Tidsbruk

Det ble tidlig i prosessen fastslått en overordnet forventning (se vedlegg B) om at hvert gruppemedlem skulle jobbe i gjennomsnitt 25 til 30 timer på ukentlig basis. Dette har vi oppnådd, noe som kommer frem i vår ukesoversikt i figur 36.

Mer utdypende oversikt over tidsbruk kan sees i vedlegg S.1



(a) Ukesoversikt **(b)** Tidsbruk per kategori

Figur 36: Statistikk over tidsbruk.

Samtidig, ser vi også at den siste perioden ble mer intensiv med tanke på arbeidsmengde enn det vi hadde forestilt oss. Vi ser i ettertid at dette kunne vært unngått ved å legge mer vekt på rapportskrivingen tidligere i prosessen.

Gruppa er fornøyd med hvordan tiden har blitt prioritert, spesielt med den endelige oversikten over tidsbruk per kategori tatt i betraktning (se figur 36). Da bachelorrapport og utviklingsresultatet legger hovedgrunnlaget for vurdering, synes vi det reflekterer positivt på vår prioritering at disse er blitt mest vektlagt.

10.2.5 Kritikk til oppgaven

Oppgavens brede nedslagsfelt var svært tiltalende for oss, og la også deler av grunnlaget for hvorfor vi valgte nettopp denne oppgaven. I tillegg forelå det få føringer for valg av teknologier og utforming, som i praksis betydde at vi var fleksible til å dra den dit vi ville. Dette medførte i praksis at store mengder tid ble brukt på utforming av krav under utførelsen av prosjektet. Selv om dette til tider var ufordrende, bidro det til å øke læringsutbyttet vårt betraktelig, noe vi i etterkant er svært takknemlig for. Vi tar også selvkritikk på området, da vi fra start burde gjort en bedre jobb i arbeidet med avgrensninger.

10.3 Videre arbeid

I dette prosjektet har vi har utviklet grunnfunksjonalitet og vist noen forslag til hvordan systemet kan benyttes for de kravene som har blitt satt. Videre arbeid blir primært å gjøre systemet klart til å settes i produksjon hos kundene.

For å gjøre dashbordløsningen klar til produksjon anbefaler vi å skrive flere spørringer som må til for å dekke kundenes individuelle behov. Det bør også etableres et system for å rulle ut nye eller oppdatere eksisterende dashbord.

Eksisterende databaseløsning hos Argos har begrenset lagringsplass på disk. Mellomlagene Prometheus og Graphite (som nevnt i kapittel 4) kan bidra til å løse denne problemstillingen, da den aggregerte dataen tar mindre lagringsplass. Dette vil gjøre det mulig å se data for lengre perioder. Det vil også være mulig å kombinere på tvers av resultatene fra API-et med en slik løsning.

For rapportløsningen vil det være optimalt om det settes opp en web-tjeneste for å generere rapportene. På den måten vil det ikke være nødvendig for kundene å installere Report Builder, dersom de ikke ønsker å gjøre endringer i rapportmalene. Et system for å rulle ut nye rapporter og oppdatere eksisterende, kunne også gjort det lettere for Argos å tilby rapportmaler til deres kunder.

Før produksjonssetting bør det grafiske utseendet til simuleringsverktøyet forbedres. Det bør også legges inn synlige feilmeldinger dersom det oppstår problemer. Per idag ser algoritmen bare på individuelle terskelverdier. Videre arbeid på verktøyet vil dermed være å gjøre algoritmen dypere, for å gjøre det mulig å simulere dersom man endrer på flere terskelverdier i sammenheng. Simuleringen som implementert ser kun på tilfeller der man ønsker å gjøre terskelverdien høyere. Det kan derfor være ønskelig å utvide verktøyet slik at den støtter en reduksjon i tillegg.

Vi har i tillegg følgende konkrete anbefalinger for å ivareta sikkerhetsutfordringene som nevnt i kapittel 9:

1. Implementer en løsning for autentisering av spørringer mot API-et. Dette anbefaler vi at gjøres gjennom spesialiserte API-nøkler som er unike for hver tjeneste som benytter seg av API-et.
2. Benytt en web-proxy som første møte for spørringer. Ved å benytte seg av en separat tjeneste som Nginx [51] vil den kunne ta seg av statiske spørringer, lastbalansering og håndtering av HTTPS-sertifikater.
3. Lag databasebrukere som bare har tilgang til å gjøre SELECT-spørringer.
4. Ikke tillat koblinger til systemet fra eksterne kilder som ikke er autentisert via Virtual Private Network (VPN) eller lignende.

For å ytterligere segregere lagene i arkitekturen anbefaler vi å gjøre alle databasespørringene om til metoder i databasehåndteringsklassen. Ved å gjøre dette vil det ikke lengre være nødvendig å sende hele databaseklassen gjennom endepunktene. Dette vil bidra til enda lavere koblinger mellom lagene, som vil medføre at det også vil bli enklere å hente data fra andre datakilder enn SQL-databaser.

11 Konklusjon

Argos ønsket å begynne arbeidet med en plattform for å bedre benytte innsamlet data fra graderingsprosessen til skannerene. Eksisterende løsning var lite intuitiv, frustrerende å vedlikeholde og ble ansett som utdatert i forhold til dagens behov.

For å løse disse problemene har vi utviklet fellesplattformen Koios. Koios tilbyr et standardisert grensesnitt for å aksessere og aggregere data fra én eller flere databaser til eksterne tjenester. Plattformen består av et FastAPI drevet API, en implementasjon av mikrokjernemodellen for å dynamisk kunne utvide funksjonalitet, samt et databasetilgangslag drevet av SQLAlchemy, utvidet med støtte for å enkelt legge til flere databaser. For å vise potensialet til Koios, har vi gjennom dette prosjektet vist hvordan plattformen kan benyttes for å visualisere og analysere data.

Et av ønskene til Argos var å visualisere dataen i sanntid gjennom et dashbord, samt generere tradisjonelle rapporter for historisk produksjon. Dette har vi løst ved å benytte Grafana som visualiseringsverktøy for dashbord, og med Report Builder som grensesnitt for rapportgenerering. Begge disse verktøyene benytter seg av Koios som hoveddatakilde. Helhetlig gir Argos' kunder fleksibilitet i visuell utforming, samtidig som at det er enkelt å justere datagrunnlaget. Koios er konstruert for at slike justeringer ikke skal påvirke andre deler av systemet, som legger til rette for enkel vedlikehold og videreutvikling.

I tillegg til visualisering, ønsket Argos et verktøy for å simulere hvordan endring i graderingsparametrene hadde påvirket utfallet av historisk produksjon. Vi har implementert denne simuleringen ved å generere en oversikt over alle defekttyper som har nedgradert plater. Denne oversikten brukes deretter for å finne produksjonsendring basert justering av graderingskriteriene.

Brukergrensesnittet til dette verktøyet har vi implementert som en egenlaget web-applikasjon. Nettsiden er bygd etter SPA modellen gjennom rammeverket Vue, i tillegg til utvalgt funksjonalitet fra jQuery. Dette gjør web-applikasjonen enkel å utvide ved fremtidige behov.

Med dette prosjektet har vi dekket alle resultatmålene vi satt innledningsvis. Gjennom tett kommunikasjon med oppdragsgiver og deres erfarne selgere, har vi laget et system som vil øke utbyttet av Argos' tjenester. Det er nå mulig å se produksjonen i sanntid og det er enklere å skreddersy rapportene etter kundenes individuelle behov. I tillegg er det nå også mulig å simulere hvordan endringer i terskelverdier ville påvirket produksjonen.

Vi har benyttet flere av de samme prosessene og verktøyene som benyttes i arbeidslivet, samtidig som vi satt teori vi har lært i studiet ut i praksis. Gruppen har regelmessig reflektert over hvordan vi syntes arbeidsprosessen har vært, og kontinuerlig forbedret oss ut ifra dette. Gjennom dette bachelorprosjektet har vi

lært om nye teknologier, designprinsipper, og gjort oss mange verdifulle erfaringer rundt prosjektarbeid.

Referanser

[1] Ungt Entreprenørskap. *Brainstorming*. URL: https://ndla.no/subject:ee3f7a15-feb6-4e78-8b37-65930ad73a09/topic:126f94e7-7b54-44b2-be7d-3edc1ca3d21e/resource:928053b2-d312-410d-ba7a-51fd7996e5e6?filters=urn:filter:54b1727c-2d91-4512-901c-8434e13339b4. (accessed: 25/02/2021).

[2] Lovenskiold. *Argos Solutions*. URL: http://lovenskiold.no/argos-solutions. (accessed: 27/01/2021).

[3] Argos Solutions AS. *Grading System*. URL: https://www.argossolutions.no/grading-system/. (accessed: 13/04/2021).

[4] Greek Gods. *Coeus (Koios, Polos)*. URL: https://www.greek-gods.org/titans/coeus.php. (accessed: 17/05/2021).

[5] Wikipedia. *Coeus*. URL: https://en.wikipedia.org/wiki/Coeus. (accessed: 17/05/2021).

[6] Veysi Ozturk. «Selection of appropriate software development life cycle using fuzzy logic». I: *Journal of Intelligent and Fuzzy Systems: Applications in Engineering and Technology* 25 (mai 2013), s. 797–810. DOI: 10.3233/IFS-120686.

[7] Henrik Kniberg. *Kanban and Scrum - Making the Most of Both*. Lulu.com, 2010. ISBN: 0557138329.

[8] TIOBE. *TIOBE index for April 2021*. URL: https://www.tiobe.com/tiobe-index/. (accessed: 27/04/2021).

[9] Redmonk. *The RedMonk Programming Language Rankings: January 2021*. URL: https://redmonk.com/sogrady/2021/03/01/language-rankings-1-21/. (accessed: 27/04/2021).

[10] IEEE. *IEEE Spectrum: The Top Programming Languages*. URL: https://spectrum.ieee.org/ns/IEEE_TPL_2020/index/2020/1/1/1/1/1/50/1/50/1/50/1/30/1/30/1/20/1/20/1/5/1/50/1/100/1/50/. (accessed: 27/04/2021).

[11] Coding Dojo. *Top Programming Languages Of 2021*. URL: https://www.codingdojo.com/blog/top-7-programming-languages. (accessed: 27/04/2021).

[12] A Bogdanchikov, M Zhaparov og R Suliyev. «Python to learn programming». I: *Journal of Physics: Conference Series* 423 (apr. 2013), s. 012027. DOI: 10.1088/1742-6596/423/1/012027. URL: https://iopscience.iop.org/article/10.1088/1742-6596/423/1/012027/meta.

[13] Gunavaran Brihadiswaran. *A Performance Comparison Between C, Java and Python*. URL: https://medium.com/swlh/a-performance-comparison-between-c-java-and-python-df3890545f6d. (accessed: 18/05/2021).

[14] GraphQL. *GraphQL*. URL: https://graphql.org/learn/queries/. (accessed: 19/05/2021).

[15] Microsoft. *Python SQL Driver*. URL: https://docs.microsoft.com/en-us/sql/connect/python/python-driver-for-sql-server. (accessed: 18/05/2021).

[16] SQLAlchemy. *SQL Alchemy*. URL: https://www.sqlalchemy.org/. (accessed: 27/04/2021).

[17] Django. *Django*. URL: https://www.djangoproject.com/. (accessed: 01/05/2021).

[18] FastAPI. *FastAPI features*. URL: https://fastapi.tiangolo.com/features/. (accessed: 01/05/2021).

[19] Flask. *Flask foreword*. URL: https://flask.palletsprojects.com/en/1.1.x/foreword/. (accessed: 01/05/2021).

[20] Techempower. *Web framework benchmarks*. URL: https://www.techempower.com/benchmarks/#section=data-r20&hw=ph&test=composite&l=zijzen-sf&f=zhb2t3-zik0zj-zik0zj-zik0zj-zik0zj-zik0zj-zik0zj-ziimf3-zik0zj-zik0zj-zik0zj-cn3. (accessed: 01/05/2021).

[21] The Apache Software Foundation. *Apache license*. URL: https://www.apache.org/licenses/LICENSE-2.0. (accessed: 20/02/2021).

[22] Prometheus Authors. *Prometheus*. URL: https://prometheus.io/docs/introduction/overview/. (accessed: 22/02/2021).

[23] Graphite. *Graphite*. URL: https://graphiteapp.org/. (accessed: 22/02/2021).

[24] Grafana. *Grafana*. URL: https://grafana.com/products/cloud/. (accessed: 18/05/2021).

[25] IzakMarais. *Grafana Reporter*. URL: https://github.com/IzakMarais/reporter. (accessed: 18/05/2021).

[26] Mark Richards. *Software Architecture Patterns*. O'Reilly Media, Inc., 2015. ISBN: 9781491925409.

[27] Joydip Kanjilal. *Best Practices of Designing and Implementing a Data Access Layer*. URL: http://www.databasedev.co.uk/data-access-layer.html. (accessed: 01/05/2021).

[28] Mikael Jern. «Thin vs. fat visualization clients». I: (jan. 1998), s. 270–273. DOI: 10.1145/948496.948535.

[29] Olav Kristensen Stein Aanensen. *Linjediagram*. URL: https://ndla.no/subject:29/topic:1:164958/resource:1:91794?filters=urn:filter:b0a79538-d211-4254-852a-5aa2c4b89db7. (accessed: 25/02/2021).

[30] Olav Kristensen Stein Aanensen. *Søylediagram*. URL: https://ndla.no/subject:29/topic:1:164958/resource:1:91784?filters=urn:filter:b0a79538-d211-4254-852a-5aa2c4b89db7. (accessed: 25/02/2021).

[31] Olav Kristensen Stein Aanensen. *Sektordiagram*. URL: https://ndla.no/subject:29/topic:1:164958/resource:1:91790?filters=urn:filter:b0a79538-d211-4254-852a-5aa2c4b89db7. (accessed: 25/02/2021).

[32] Christine Anne Sætre. *Tabeller*. URL: https://innsida.ntnu.no/wiki/-/wiki/Norsk/Tabeller. (accessed: 25/02/2021).

[33] Adam Bertram. *the state of config file formats: XML vs. YAML vs. JSON vs. HCL*. URL: https://octopus.com/blog/state-of-config-file-formats. (accessed: 18/05/2021).

[34] Palletprojects. *Click*. URL: https://click.palletsprojects.com/en/8.0.x/. (accessed: 18/05/2021).

[35] Microsoft Contributors. *XML Query Syntax for XML Report Data (SSRS)*. URL: https://docs.microsoft.com/en-us/sql/reporting-services/report-data/xml-query-syntax-for-xml-report-data-ssrs?view=sql-server-ver15. (accessed: 12/05/2021).

[36] Michael L. Waskom. «seaborn: statistical data visualization». I: *Journal of Open Source Software* 6.60 (2021), s. 3021. DOI: 10.21105/joss.03021. URL: https://doi.org/10.21105/joss.03021.

[37] J. D. Hunter. «Matplotlib: A 2D graphics environment». I: *Computing in Science & Engineering* 9.3 (2007), s. 90–95. DOI: 10.1109/MCSE.2007.55.

[38] pdf. *Black*. URL: https://github.com/psf/black. (accessed: 14/05/2021).

[39] pylint. *Pylint: Star your Python code!* URL: https://www.pylint.org/. (accessed: 14/05/2021).

[40] Nick Coghlan Guido van Rossum Barry Warsaw. *PEP 8 – Style Guide for Python Code*. URL: https://www.python.org/dev/peps/pep-0008/. (accessed: 14/05/2021).

[41] Pytest. *Pytest fixtures: explicit, moduler, scalable*. URL: https://docs.pytest.org/en/6.2.x/fixture.html. (accessed: 18/05/2021).

[42] Benjamin Day. *Eliminate Database Dependencies in Test Driven Development*. URL: https://visualstudiomagazine.com/articles/2009/09/01/eliminate-database-dependencies-in-test-driven-development.aspx. (accessed: 18/05/2021).

[43] Guru99. *Shortest Job First(SJF): Preemptive, Non-Preemptive Example*. URL: https://www.guru99.com/shortest-job-first-sjf-scheduling.html. (accessed: 18/05/2021).

[44] Sphinx. *Sphinx*. URL: https://www.sphinx-doc.org/en/master/. (accessed: 18/05/2021).

[45] Andrew Etter. *Modern Technical Writing*. Amazon, 2016.

[46] Smartbear. *API Development for Everyone*. URL: https://swagger.io/. (accessed: 18/05/2021).

[47] Mano Paul. *Official (ISC)2 Guide to the CSSLP*. 2nd. USA: Auerbach Publications, 2013. ISBN: 9781466571273.

[48] FastAPI. *FastAPI Security*. URL: https://fastapi.tiangolo.com/tutorial/security/. (accessed: 10/05/2021).

[49] Google. *Crazy 8's*. URL: https://designsprintkit.withgoogle.com/methodology/phase3-sketch/crazy-8s. (accessed: 29/04/2021).

[50] Francesco Cirillo. *The Pomodoro technique*. URL: https://francescocirillo.com/pages/pomodoro-technique. (accessed: 18/05/2021).

[51] Nginx. *Nginx*. URL: https://www.nginx.com/. (accessed: 10/05/2021).

A Oppgavebeskrivelse

Analyseplattform

Oppdragsgiver:

Argos Solutions AS
Dyrmyrgata 35
3612 Kongsberg

Kontaktperson:
Tor Nordseth
✉ tor.nordseth@argossolutions.no
☎ +47 916 69 420

Bakgrunn: Argos Solutions AS leverer inspeksjonsverktøy for å detektere og klassifisere feil i trevirke. Man leverer da både fastvaren og programvaren som gjennomfører inspeksjonen. Systemet lagrer informasjon om hver enkelt plate i en database i hver enkelt skanner, da hovedsakelig informasjon om klassifisering og hvilke feil som ble funnet. I dag finnes det en utdatert og komplisert løsning for å hente ut rapporter fra en enkelt skanner, men det er ikke mulighet til å se på data på tvers av skannere.

Oppgave: Lage en analyseplattform som kan rulles ut enten internt i en fabrikk eller i skyen. Det bør gjennomføres en vurdering av fordeler og ulemper for begge løsningene som bør inneholde hovedsakelig ytelse, risiko og kostnad.

Plattformen skal basere seg på data fra hver enkelt skanner i fabrikken og må dermed aggregere disse dataene på en fornuftig måte. I første omgang er det kun snakk om små mengder data per plate, men det kan også bli aktuelt å laste opp bilder.

Plattformen skal hovedsakelig gi brukeren mulighet til å gjøre tre ting:

1. Se løpende statistikk over nåværende produksjon i sanntid. Dette bør gjennomføres som et dashbord som kan vises på en skjerm i fabrikken med den hensikt å gi enkel men konkret informasjon. Videre bør det være mulig å enkelt kunne tilpasse dashbordet til ulike fabrikker som da kanskje ønsker ulike måleparametere, har ulikt antall skannere eller ander forskjeller.
2. Kunne produsere rapporter som viser oversikt over en gitt tidsperiode (eksempelvis forrige måned). Disse rapportene bør kunne enkelt skrives ut i papirformat. Rapportene bør også være enkle å skreddersy for ulike fabrikker, slik at det kan tilpasses. Videre bør det tas hensyn til at man ofte ønsker flere ulike typer rapporter, og kanskje også rapporter som produseres på et gitt tidsintervall.
3. Se utfallene av å justere på deteksjonsalgoritmene ved at man enten øker eller senker toleransen for feil. Dette vil selvfølgelig kun innebære feil som er over minste deteksjonsgrad og som har blitt lastet inn i databasen. Brukergrensesnittet bør da enkelt vise at dersom man for eksempel øker en toleranse fra 5 til 6 mm så vil det føre til følgende produksjonsøkning.

B Grupperegler

Grupperegler

Dette skrivet inneholder utarbeidede grupperegler for bachelorgruppe 2 (BIDAT39) med følgende medlemmer:

Andreas Nygård Ljøterud	-	andrenlj@stud.ntnu.no	-	509250
Mads Olsen Nekkøy	-	madsone@stud.ntnu.no	-	509245
Lars Stormark Pedersen	-	larsspe@stud.ntnu.no	-	509263

Gruppens visjon er at….

a. vi, oppdragsgiver og universitetet blir strålende fornøyd med sluttresultatet.
b. vi skal sitte igjen med verdifull kunnskap som vi kan ta med oss videre.
c. sluttresultatet vil være nyttig for Argos og deres kunder.

---

1. Det forventes at hver gruppedeltager er en aktiv og konstruktiv bidragsyter i arbeidet.
2. Referat skal føres av alle gruppedeltagere under alle møter. Ved ferdigstilt møte skal disse samles i ett referat som legges i Confluence.
    a. For å holde oversikt over vesentlige forhold som f.eks beslutninger og råd fra veileder eller oppdragsgiver, skal det samlede referatet følge avtalte stilråd i Confluence.
3. Ved møter og felles arbeid skal mobiltelefoner legges vekk og settes på "ikke forstyrr".
4. Arbeidsoppgaver skal fordeles slik at arbeidsmengde og innsats blir likt fordelt.
5. Hver gruppedeltager forventes å jobbe i gjennomsnitt mellom 25 og 30 timer hver uke.
    a. Daglig aktivitet i arbeidstimer samt en kort beskrivelse av arbeidet som er gjort skal føres i individuell side i avtalt regneark.
    b. Møteinnkallelser skal føres i felles område i samme regneark.
6. Vi oppmuntrer hverandre og motiverer hverandre når vi ser at noen er demotiverte.
7. Ved uenighet streber vi etter å finne et alternativ som fungerer for alle. Hvis vi ikke kommer til enighet, spør vi etter rådgivning fra veileder.

Regler og rangering vil oppdateres med jevne mellomrom på fastsatte datoer. Dette for å sikre at regler følges og for å evaluere nytteverdien.

Andreas Nygård Ljøterud Andreas N. Lj , dato 30/01

Mads Olsen Nekkøy Mads O Nekkøy , dato 30/01

Lars Stormark Pedersen , dato 30/01

C Forprosjekt

Forprosjekt

Andreas Nygård Ljøterud
Mads Olsen Nekkøy
Lars Stormark Pedersen

30. januar 2021

Innhold

1 Mål og rammer

1.1 Bakgrunn

> "Argos Solutions AS er et globalt høyteknologi-selskap som holder til på Kongsberg. Selskapet utvikler og produserer avanserte systemer for automatisk visuell inspeksjon til ledende treplateprodusenter verden over. Systemene benyttes i fabrikker for å automatisere og optimalisere produksjonsprosesser."[5]

Levering og drifting av det automatiserte inspeksjonsverktøyet "Argos Grading System" er del av Argos' kjernevirksomhet. AGS analyserer forskjellige typer treplater og gir en vurdering basert på fabrikkens krav. Dataene om hver enkelt plate og informasjon om eventuelle defekter lagres i en database, som kunden deretter kan gjøre spørringer mot ved behov.

For å forenkle prosessen for ikke-teknisk personell finnes det en løsning som genererer forhåndsdefinerte rapporter. De ansatte bruker deretter disse rapportene for å evaluere produksjonen.

Over tid har det blitt oppdaget svakheter ved denne løsningen. De eksisterende forhåndsdefinerte rapportene inneholder data som ofte ikke er interessant for kunden, og det er vanskelig å skreddersy rapportene etter fabrikkenes individuelle behov. Det ikke mulig å hente ut data fra systemet i sanntid.

AGS er en av få maskiner i produksjonslinjen som kan bygge et helhetlig bilde over produksjonen. Det er derfor mye potensiale i dataene som blir samlet.

Vi fikk høre om Argos gjennom en bekjent av oss, som nå er ansatt der. Vi tok tidlig kontakt og fikk valget mellom 3 forskjellige oppgaver. Forslaget om en analyseplattform vekket umiddelbar interesse blant gruppens medlemmer, da den ville gi oss faglige utfordringer på flere relevante områder.

1.2 Prosjektmål

Resultatmål

Prosjektet kan deles i fire hovedområder:

1. Det skal gjøres en vurdering av fordeler og ulemper ved bruk av skyløsninger.

2. Det skal være mulig å lage skreddersydde rapporter til hver enkelt fabrikk. Rapportene må enkelt kunne skrives ut i PDF format.
3. Det skal være mulig å se løpende oversikt over produksjonen innenfor angitte tidsområder.
4. Det skal være mulig å simulere historisk produksjonsdata med forskjellige målekriterier.

Ved prosjektets slutt ønsker Argos at de fire punktene over er operative, slik at det kan vises og eksperimenteres med. Argos ønsker også at de ansatte i bedriften enkelt kan ta over videreutvikling og vedlikehold av platformen.

Effektmål

Argos har et ønske om at analyseplattformen skal bidra til å:

- gi kunder enklere tilgang til produksjonsdata slik at feil oppdages raskere og svinn minimeres. Dette vil øke antall godkjente plater per skift.
- gi kundene økt utbytte av deres tjenester. Dette vil ha positiv effekt på selskapets fortjeneste og tilfredsstillelse blant kundene.

Læringsmål

Etter endt bachelor ønsker vi å ha mer kunnskap om:

- verktøy som er aktuelle i arbeidslivet, slik at vi er mer effektive i daglig arbeid.
- jobbe sammen i team, for å bli bedre forberedt til arbeidslivet.
- hvordan det er å jobbe med en smidig utviklingsmetode.
- dokumentasjon, mer spesifikt hvordan dokumentere slik at programvaren lett kan videreføres.
- testing og kvalitetskontroll
  - rutiner for testing og bruk av ulike verktøy
  - forstå viktigheten til testing
  - effekten av god kvalitetskontroll
- rapportskriving, utforme og skrive en informativ rapport

1.3 Rammer

Når vi planlegger dette prosjektet har vi identifisert noen eksterne rammer:

- Tidsrammen for prosjektet er satt ved leveringsfrist 20. mai 2021.
- Alle serverene som står hos kundene idag benytter Windows Server som operativsystem.

2 Omfang

2.1 Fagområde

UX

UX(User-experience) handler om hvordan en bruker opplever og samhandler med et produkt. For å levere et tilfredstillende produkt, vil vi gjennom hele utviklingsløpet fokusere på prinsipper fra den iterative prosessen *"user-centered design"*. Denne prosessen går i hovedsak ut på å kontinuerlig bearbeide produktet med brukernes tilbakemeldinger i sentrum.

Skyløsninger

Datatilsynet har definert skytjenester som "en samlebetegnelse på alt fra dataprosessering og datalagring til programvare på servere som er tilgjengelig fra eksterne serverparker tilknyttet internett. Serverparkene kjennetegnes ved at de er laget for dynamisk skalering." Skytjenestebegrepet kan videre deles ned i 3 hovedkategorier: Infrastruktur som tjeneste, plattform som tjeneste og programvare som tjeneste. Det finnes også tre forskjellige leveransemodeller: Offentlig, privat og hybrid sky. [3].

Flere og flere bedrifter benytter seg av skyløsninger fordi det kan tilby umiddelbar skalering, kontraktfestet oppetid, lave initielle investeringskostnader og redusert behov for lokalt ansatte for drift. Samtidig er det fortsatt noen utfordringer: Sikkerhet vedrørende tilgang til ressursene, mindre kontroll over prosessen og data, uklare juridiske forhold, leverandør lock-in og kompetanseheving. [1]

2.2 Avgrensning

For å fullføre prosjektet etter oppsatte mål har vi lagt noen avgrensinger til løsningen:

- Vi vil levere et produkt som støtter internasjonalisering, men vi tar ikke ansvar for implementasjon av språk og enheter utover det som er oppgitt i kravspesifikasjon.
- Sluttproduktet skal være fungerende, men ikke nødvendigvis klart til og rulles ut i produksjon.
- Analyseplattformen skal aggregere og analysere data fra produksjonen, men eventuelle endringer som skal gjøres som følge av dette vil ikke gjøres gjennom vårt produkt.

2.3 Oppgavebeskrivelse

Oppgaven fra Argos består av en analyseplattform som kan rulles ut enten internt i en fabrikk eller i skyen. Følgende fire hovedpunkter er hentet fra oppgavebeskrivelsen til Argos:

1. Gjøre en vurdering av fordeler og ulemper ved bruk av skytjenester, hovedsaklig med fokus på ytelse, kostnad og risiko.
2. Se løpende statistikk over nåværende produksjon i sanntid. Dette bør gjennomføres som et dashbord som kan vises på en skjerm i fabrikken, med den hensikt å gi enkel men konkret informasjon. Videre bør det være mulig å enkelt kunne tilpasse dashbordet til ulike fabrikker som da kanskje ønsker ulike måleparametere, har ulikt antall skannere eller andre forskjeller.
3. Kunne produsere rapporter som viser oversikt over en gitt tidsperiode. Disse rapportene skal kunne enkelt skrives ut i papirformat. Rapportene bør også være enkle å skreddersy, slik at de kan tilpasses ulike fabrikker. Videre bør det tas hensyn til at man ofte ønsker flere ulike typer rapporter, og kanskje også rapporter som produseres på et gitt tidsintervall.

4. Se utfallene av å justere på deteksjonsalgoritmene ved at man enten øker eller senker toleransen for feil. Dette vil selvfølgelig kun innebære feil som er over minste deteksjonsgrad og som har blitt lastet inn i databasen. Brukergrensesnittet bør da enkelt vise at dersom man for eksempel øker en toleranse fra 5 til 6 mm så vil det føre til følgende produksjonsøkning.

3 Prosjektorganisering

3.1 Ansvarsforhold og roller

- Gruppeleder
  I tillegg til å være utvikler, har Lars Stormark Pedersen blitt utnevnt gruppeleder. Dette innebærer et overordnet ansvar for fremdriften i prosjektet.

- Utviklere
  Andreas Nygård Ljøterud og Mads Olsen Nekkøy er utviklere. Med ulik kompetanse fra tidligere valgfag, vil ansvarsområdene være noe flytende.

- Veileder
  Tom Røise, universitetslektor ved NTNU-Gjøvik.

- Oppdragsgiver
  Argos Solutions AS, representert ved Tor Nordseth og Ole Martin Ruud.

3.2 Rutiner og regler i gruppa

- Det forventes at hver gruppedeltager er en aktiv og konstruktiv bidragsyter i arbeidet.

- Referat skal føres av alle gruppedeltagere under alle møter. Ved ferdigstilt møte skal disse samles i ett referat som legges i Confluence.

  – For å holde oversikt over vesentlige forhold som f.eks beslutninger og råd fra veileder eller oppdragsgiver, skal det samlede referatet følge avtalte stilråd i Confluence.

- Ved møter og felles arbeid skal mobiltelefoner legges vekk og settes på 'ikke forstyrr'.
- Arbeidsoppgaver skal fordeles slik at arbeidsmengde og innsats blir likt fordelt.
- Hver gruppedeltager forventes å jobbe i gjennomsnitt mellom 25 og 30 timer hver uke.
  - Daglig aktivitet i arbeidstimer samt en kort beskrivelse av arbeidet som er gjort skal føres i individuell side i avtalt regneark.
  - Møteinnkallelser skal føres i felles område i samme regneark.
- Vi oppmuntrer hverandre og motiverer hverandre når vi ser at noen er demotiverte.
- Ved uenighet streber vi etter å finne et alternativ som fungerer for alle. Hvis vi ikke kommer til enighet, spør vi etter rådgivning fra veileder.

4 Planlegging, oppfølging og rapportering

4.1 Valg av utviklingsmodell

I valg av utviklingsmodell begynte vi ved å identifisere følgende krav til en ideell modell. Nåværende prosjektkrav er relativt åpne, vi antar dermed at disse kan endre seg i løpet av prosjektperioden. En viktig del av prosjektet er å gjennomføre en vurdering av skyteknologi. Vi ser for oss at enkelte krav til prosjektet kan endres som resultat av undersøkelsen. Gruppen har i tillegg lite erfaring med prosjektarbeid av denne størrelsen. Dette medfører at vi ønsker hyppig oppfølging fra oppdragsgiver og veileder for å raskt kunne korrigere retning om nødvendig.

Med disse kravene begynte vi å se på de lineære utviklingsmodellene, fossefall og inkrementell modell. Vi fant raskt ut at disse modellene ikke passet vårt prosjekt. De viktigste kravene disse utviklingsmodellene ikke oppfyller er å kunne planlegge utifra resultat av tidligere utvikling, samt mindre mulighet for å endre kurs dersom det er nødvendig. Vi har begrenset med tid, og må derfor bruke den så effektivt som mulig.

Vi fortsatte ved å undersøke de smidige modellene Scrum og Kanban. Scrum gir oss god mulighet for å planlegge basert på resultater fra tidligere

sprinter. Hvert "Sprint review meeting" gir oss også mulighet til å få tilbakemeldinger underveis. Ulempen vi finner med Scrum er at den kan være litt rigid for teamet da ingen av oss har erfaring ved å være Scrum master. Kanban er en enda mer smidig metodikk, som passer godt inn i våre krav til en ideell utviklingsmodell. Vi har også erfaring med metoden fra tidligere prosjekter. Ulempen med Kanban er dog at den kan være litt for løs.

Vi antar at uforberedte omstendigheter vil oppstå underveis i utviklingen, og anser derfor en smidig metodikk som optimalt. Ved kontinuerlig feedback og veiledning, fra både oppdragsgiver og veileder, er det også en fordel at modellen er fleksibel. Dette kombinert med vår tidligere erfaring med Kanban, gjorde at vi valgte å gå for en Scrumban modell.

4.2 Bruk av modellen

Det finnes ikke én riktig måte å sette opp en utviklingsprosess, den må tilpasses til hvert prosjekt og team-medlemmene som skal jobbe på prosjektet. Som inspirasjon til vår modell har vi brukt råd og eksempler fra 'Kanban and Scrum - Making the Most of Both' [4]. Vi har valgt å ha møter med oppdragsgiver hver andre uke for å vise ny funksjonalitet og få tilbakemeldinger. Rett etter møtet med oppdragsgiver vil vi holde et retrospektivt møte for å vurdere perioden og evaluere om vi bør gjøre noen endringer.

Kravene fra oppgaven vil grupperes i epics. Dette vil gjøre det enklere å holde oversikt over den generelle progresjonen i prosjektet. Kravene vil etter beste evne bli delt opp i like store deler, hvor målet er at hver oppgave skal i gjennomsnitt ta 3-5 dagsverk. Videre vil konkrete oppgaver kobles til hver epic slik at vi kan se kontinuerlig fremgang på scrumban tavlen. Hver uke vil vi ha et team-møte der vi går gjennom aktive epics, deres status og eventuelle omprioriteringer. I møtet henter vi også inn nye oppgaver fra *backloggen*. Scrumban-tavlen vår har 5 kolonner:

- **Selected for development** Oppgaver som er hentet fra *backlog*, men som ikke enda er påbegynt.
- **In progress** Påbegynte oppgaver.
- **Waiting** Oppgaver som er påbegynt, men som nå venter på en ekstern faktor før arbeid kan fortsette.
- **Review** Oppgaver som er ferdige, men som venter på gjennomgang av de andre gruppemedlemmene for kvalitetssikring.

- **Done** Ferdige oppgaver.

Initielt har vi satt WIP grensene for hver kolonne til 6, det dobbelte av antall medlemmer. Vi ser for oss å modifisere disse grensene når vi får satt prosessen ut i praksis.

Plan for statusmøter og beslutningspunkter i perioden

For dette prosjektet har vi satt opp faste statusmøter internt, med oppdragsgiver og med veileder. Innad i gruppen jobber vi sammen veldig ofte, så vi holder uformelle statusmøter for å oppdatere hverandre på progresjon hele tiden. Hver tirsdag vil vi ha et møte der vi bruker Scrumban tavlen for å evaluere progresjon og fremdrift. Vi har møte med veileder tirsdager kl. 1300 til 1330. Annenhver torsdag kl 0930 til 1030 har vi møte med oppdragsgiver. For å bekrefte at vi holder nødvendig tempo vil vi sette milepæler med konkrete tidsfrister. Disse utdypes videre under plan for gjennomføring (6). Dersom vi oppdager at vi ligger bak skjema må vi vurdere om vi må endre på tidsestimatet, eller om det er mulig å komme i mål ved å legge inn litt ekstra innsats i en avgrenset periode.

5 Organisering av kvalitetssikring

5.1 Dokumentasjon, standardbruk og kildekode

Når vi skriver commit meldinger vil vi følge *conventional commits* [2] standarden. Når vi skriver kildekode vil vi holde oss til godkjente kodestilstandarder for språket. Eksempler på dette er C++ sin *C++ core guidelines*[1] og Python sin *Black*[2] og *Flake8*[3].

For at Argos kan videreutvikle programvaren er det nødvendig med tilstrekkelig dokumentasjon. Gruppen kommer til å tilrettelegge for dette, gjennom bruken av blant annet Confluence, commit-meldinger og kommentering av kildekode.

[1] https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines
[2] https://github.com/psf/black
[3] https://flake8.pycqa.org/en/latest/

5.2 Konfigurasjonsstyring

Git vil benyttes for å gjøre endringer i kildekoden. Det er et krav at minst en annen person godkjenner arbeid før endringene flettes til hovedbranchen. Vi vil også benytte automatisk validering av endringer ved hjelp av automatiske pipelines ved fletting. Konkret oppsett av arbeidsflyt og utviklingsmiljø kommer når språk og rammeverk er avgjort.

5.3 Verktøy

Oppsumert er dette verktøyene vi planlegger å bruke:

Verktøy	**Beskrivelse**
Confluence	Dokumentasjon av arbeid, blant annet møtereferater, kravspesifikasjon og diverse dokumenter.
Jira	Styring og oversikt over arbeidsoppgaver. Brukes hyppig i ScrumBan-modellen.
Bitbucket	Lagring av kildekode.
Excel	Loggføre arbeidstimene.
Teams	Kommunikasjon mellom gruppen, veileder og oppdragsgiver.
Overleaf	Skrive forprosjektrapport.
SkyHigh	Intern sky for testmiljø.
Instagantt	Lage Gantt diagram.

5.4 Risikoanalyse

Under følger en identifikasjon og analyse av situasjoner som kan oppstå i løpet av prosjektperioden. Disse situasjonene er beskrevet og deretter knyttet opp mot tilhørende risiko og konsekvens. Videre er tiltak formulert og usikkerhetsvurdering gjennomført.

Sannsynlighet og konsekvens er klassifisert etter skala lav, middels og høy.

1. **Tidsfrist overskrides**
   Uforutsette hendelser som tap av arbeid, alvorlig sykdom, tekniske problemer eller mangel på kompetanse kan føre til forsinkelser i prosjektets fremdrift. Sannsynlighet vurderes som middels og konsekvens vurderes som høy.

2. **Overestimert omfang**
   Gruppas manglende erfaring med prosjekter av denne størrelsen, gjør at tidsestimering er vanskelig. Sannsynlighet vurderes som middels og konsekvens vurderes som middels.

3. **Tap av arbeid ved systemkræsj eller uaktsomhet**
   Tap av arbeid i utviklingsprosessen kan i verste fall ha katastrofale ringvirkinger. Sannsynlighet vurderes som lav og konsekvens vurderes som høy.

4. **Mangel på kompetanse**
   Gruppa har lite erfaring med prosjekter av denne størrelsen, og vi anser det som høyst sannsynlig at vi vil støte på arbeidsoppgaver som krever at vi må tilegne oss ny kunnskap. Sannsynlighet vurderes som høy og konsekvens vurderes som middels.

5. **Tap av gruppemedlemmer**
   Alvorlig sykdom, ulykker eller at gruppemedlemmer slutter kan medføre at gruppas kapasitet blir kraftig redusert. Sannsynlighet vurderes som lav og konsekvens vurderes som høy.

6. **Konflikter innad i gruppa**
   Uenigheter eller andre samarbeidslige faktorer underveis i prosjektløpet kan skape konflikter innad i gruppa. Sannsynlighet og konsekvens vurderes som lav.

7. **Data fra oppdragsgiver kommer på avveie**
   Dersom denne dataen kommer på avveie vil det kunne direkte påvirke kunden samt reflektere negativt på Argos. Sannsynlighet vurderes som lav og konsekvens vurderes som høy.

8. **Oppdragsgiver endrer krav underveis**
   Under utviklingsprosessen er det sannsynlig at enkelte krav endres, fjernes eller at det oppstår nye krav. Sannsynlighet vurderes som høy og konsekvens vurderes som lav.

9. **Oppdragsgiver utsettes for uforutsette hendelser**
   Argos blir utsatt for cyberangrep, terror eller andre uforutsette hendelser som gjør at de ikke lenger kan fortsette samarbeidet med prosjektgruppa og NTNU. Sannsynligheten vurderes som lav og konsekvens vurderes som høy.

		Sannsynlighet		
		Lav	Middels	Høy
Konsekvens	Lav			8
	Middels	6	2	4
	Høy	3, 5, 7, 9	1	

Figur 1: Risikovurdering

Fra risikovurderingen er det 3 risikoer som peker seg ut og som kategoriseres som høy risiko, nemlig 1, 2 og 4. Tiltak for å mitigere konsekvensen av disse beskrives i figur 2.

#	Risiko	Tiltak
1	Tidsfrist overskrides	Det etableres en plan for gjennomføring (6) som kontinuerlig brukes for å bekrefte at vi er *à jour*.
2	Overestimert omfang	En smidig utviklingsmodell (4.1) gir oss friheten til å endre kurs dersom det skulle være nødvendig.
4	Mangel på kompetanse	Det gjennomføres kontinuerlige status- og veilednigsmøter(4.2).

Figur 2: Tiltak med tilhørende beskrivelse

6 Plan for gjennomføring

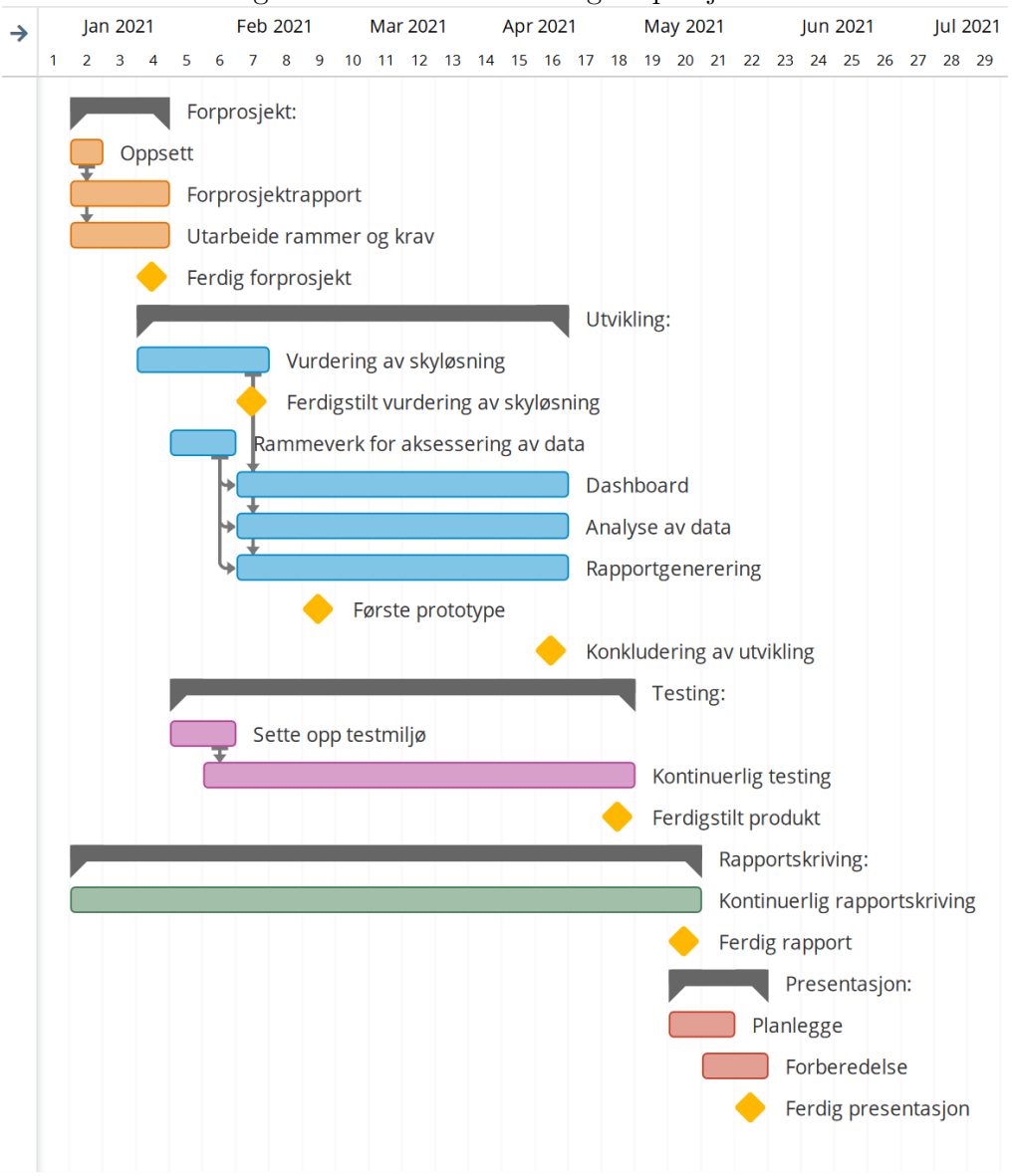
6.1 Gantt-skjema

I perioder med parallelle oppgaver (fig. 3), planlegger vi å prioritere mellom disse på våre ukentlige progresjonsmøter (4.2).

6.2 Milepæler

1. **31. januar:** Ferdigstilt forprosjekt
2. **15. februar:** Ferdigstilt vurdering av skyløsning
3. **1. mars:** Første prototype
4. **22. april:** Konkludering av utvikling
   - Ingen ny funksjonalitet blir påbegynt, fokus på finpuss og testing.
5. **6. mai:** Ferdigstilt produkt
   - Testing og koding er ferdig, fokuset går over på rapportskriving.
6. **20. mai:** Ferdigstilt rapport

Figur 3: Grov tidsestimering av prosjektet



Referanser

[1] Mahantesh Birje mfl. «Cloud computing review: Concepts, technology, challenges and security». I: *International Journal of Cloud Computing* 6 (jan. 2017), s. 32. DOI: 10.1504/IJCC.2017.083905.

[2] Conventional commits. *Conventional commit spesification*. URL: https://www.conventionalcommits.org/en/v1.0.0/#specification. (accessed: 27/01/2021).

[3] Datatilsynet. *Skytjenester*. URL: https://www.datatilsynet.no/personvern-pa-ulike-omrader/internett-og-apper/skytjenester/. (accessed: 26/01/2021).

[4] Henrik Kniberg. *Kanban and Scrum - Making the Most of Both*. Lulu.com, 2010. ISBN: 0557138329.

[5] Lovenskiold. *Argos Solutions*. URL: http://lovenskiold.no/argos-solutions. (accessed: 27/01/2021).

D Prosjektavtale

NTNU

Norges teknisk-naturvitenskapelige universitet

Vår dato Vår referanse

Prosjektavtale

mellom NTNU Fakultet for informasjonsteknologi og elektroteknikk (IE) på Gjøvik (utdanningsinstitusjon), og

ARGOS SOLUTIONS AS

(oppdragsgiver), og

ANDREAS NYGÅRD LJØTERUD

MADS OLSEN NEKKØY

LARS STORMARK PEDERSEN (student(er))

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer:

1. Studenten(e) skal gjennomføre prosjektet i perioden fra 11.01 til 20.05.

   Studentene skal i denne perioden følge en oppsatt fremdriftsplan der NTNU IE på Gjøvik yter veiledning. Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. Oppdragsgiver plikter på forespørsel fra NTNU å gi en vurdering av prosjektet vederlagsfritt.

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
   - Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon, reiser og nødvendig overnatting på steder langt fra NTNU i Gjøvik. Studentene dekker utgifter for ferdigstillelse av prosjektmateriell.
   - Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.

3. NTNU IE på Gjøvik står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av intern og ekstern sensor. Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

12. Deltakende personer ved prosjektgjennomføringen:

NTNUs veileder (navn): TOM RØISE

Oppdragsgivers kontaktperson (navn): TOR NORDSETH

Student(er) (signatur): Mads O Vekelyg dato 12.01

Andreas N. Ly dato 12.01

dato 12.01

dato

Oppdragsgiver (signatur): Tor Nordseth dato 21.01

Signert avtale leveres digitalt i Blackboard, rom for bacheloroppgaven.
Godkjennes digitalt av instituttleder/faggruppeleder.

Om papirversjon med signatur er ønskelig, må papirversjon leveres til instituttet i tillegg.
Plass for evt sign:

Instituttleder/faggruppeleder (signatur): dato

E Skyundersøkelse

Hvordan påvirker faktorene ytelse, risiko og kostnad en migrering fra lokal infrastruktur til sky?

1 Introduksjon

Det er ønskelig at analyseplattformen skal fungere både i skyen og internt i fabrikken. I den forbindelse ønsker Argos å sammenligne lokale og skybaserte løsninger, med fokus på risiko, ytelse og kostnad. Det er et spesielt fokus på sikkerhet, grunnet skepsis fra Argos sine kunder om å sende data ut i skyen.

1.1 Problemstilling

I denne vurderingen skal vi besvare følgende spørsmål:

Hvordan påvirker faktorene ytelse, risiko og kostnad en migrering fra lokal infrastruktur til sky?

1.2 Metode

Innhenting av informasjon og kunnskap skjedde fortrinnsvis gjennom litteratursøk. Hovedressursene vi tok i bruk var akademiske rapporter og statlige utredninger.

1.3 Avgrensning

Skytjenester er et stort fagfelt og vi har begrenset med tid. På bakgrunn av dette må vi definere et sett med avgrensninger.

- Begrense antall tekniske detaljer, da oppdragsgiver ønsker en overordnet vurdering i motsetning til en teknisk detaljert rapport.
- Vurderingen er del av en større oppgave, gruppen må derfor prioritere tidsbruken. Det ble bestemt at hovedinnsatsen i oppgaven skulle gå over en 2 ukers periode.

1.4 Hva er skytjenester?

Det finnes ikke en entydig definisjon på hva skytjenester er, eller et sett med standarder en tjeneste må oppfylle for å bli kalt en skytjeneste [1]. Regjeringens definisjon av skytjenester tar utgangspunkt i den amerikanske standardiseringsorganisasjonen National Institute of Standards and Technology

(NIST). NIST definerer skytjenester ved å bestemme fem essensielle egenskaper en tjeneste må inneha for å kunne kalles en skytjeneste [8]:

1. **Selvbetjent oppsett**
   Kunder kan administrere ressurser ved behov, uten noen form for menneskelig interaksjon med tjenesteleverandør.

2. **Tilgang via nettverk**
   Tjenestene er tilgjengelig over nettverk gjennom standardmekanismer fra ulike typer klienter.

3. **Samling av ressurser**
   Leverandørens ressurser er samlet slik at de dynamisk kan fordeles etter behov.

4. **Umiddelbar fleksibilitet**
   Ressursene bak en skytjeneste er fleksible i det at de raskt kan skaleres opp eller ned etter behov.

5. **Ressursbruk måles**
   Ressursbruk styres og optimaliseres ved hjelp av automatiske måleverktøy. Oversikt over bruk og kostnader er dermed lett tilgjengelig for både kunde og leverandør.

Tjenestemodeller

Skytjenester deles opp i tjenestemodeller, som hver sikter på å dekke virksomheters ulike behov. Tjenestemodellene beskriver ansvarsfordelingen mellom skyleverandør og kunde. De tre vanligste er: Programvare som tjeneste, plattform som tjeneste og infrastruktur som tjeneste.

- **Programvare som tjeneste** (Software-as-a-Service, SaaS) overlater mest ansvar til skyleverandøren. Kunden benytter simpelthen leverandørens applikasjoner gjennom for eksempel nettleser eller annen tynn klient. Med SaaS administrerer kunden verken underliggende infrastruktur, servere, operativsystemer eller lagringsmuligheter.

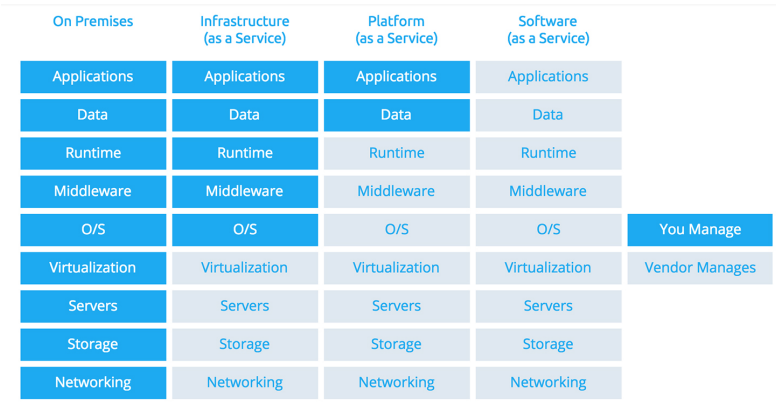
  Eksempler på SaaS er Google Workspace og Microsoft Office 365.

- **Plattform som tjeneste** (Platform-as-a-service, PaaS) gir kunden tilgang til en forvaltet plattform for bygging og levering av digitale tjenester i skyen. Kunden kan innføre egne applikasjoner og benytte disse verktøyene i skyen, i stedet for å selv måtte investere i infrastruktur. I likhet med SaaS har leverandør ansvar for drift og vedlikehold, men kunden har kontroll over egne applikasjoner.

  Eksempler på PaaS er AWS Lambda og Red Hat OpenShift.

- **Infrastruktur som tjeneste** (Infrastructure-as-a-Service, IaaS) gjelder umiddelbar levering av datainfrastruktur, klargjort og administrert over internett. Med andre ord er leverandøren ansvarlig for infrastruktur, mens kunden benytter seg av leverandørens virtualiserte ressurser. Dette medfører at kunden selv administrerer operativsystemer, lagringsmuligheter og distribuerte applikasjoner.

  Eksempler på Iaas er Microsoft Azure og Linode.



Figur 1: Hver av tjenestemodellenes anvarsfordeling sammenlignet med lokal infrastruktur [5].

Leveransemodeller

Skytjenester kan leveres på flere måter:

- **Offentlig tilgjengelig sky** (public cloud) er skytjenester som er åpent tilgjengelige for allmenheten.

- **Privat tilgjengelig sky** (private cloud) er en lukket skytjeneste som kun er tilgjengelig for den virksomheten tjenesten skal gjelde for.
- **Hybridsky** (hybrid cloud) er en blanding mellom offentlig og privat sky.
- **Gruppesky** (community cloud), hvor flere virksomheter deler en privat sky.

2 Diskusjon

I følge en undersøkelse gjort i 2018 av Vanson Bourne på vegne av Teradata [12], er flertallet av de største teknologiselskapene i verden enige om at skyen er det beste stedet å kjøre analyser. Samme undersøkelse viser også at noen av de største barrierene for å flytte analyser til skyen er bekymringer knyttet til sikkerhet, ytelse, overholdelse av regelverk, integrasjon med eldre systemer, mangel på kompetanse og mangel på tillit.

2.1 Ytelse

Undersøkelser viser at en av de mer fremtredene bekymringene blant bedrifter er lav ytelse i skyen [12]. Faktorer som kan påvirke ytelse er blant annet geografisk plassering mellom fabrikk og datasenter, bedriftenes nettverksstruktur og systemets optimalisering [2].

De største skyleverandørene har serverparker over hele verden, og som kunde har man mulighet for å bestemme geografisk plassering av data [3], nettopp for å imøtekomme disse utfordringene. Dette kan føre til ekstra kostnader, men samtidig forenkle juridiske utfordringer knyttet til overføring av data på tvers av landegrenser [11].

Selv om skyløsninger reduserer avhengigheten av lokal infrastruktur, er det essensielt at det eksisterer pålitelig nettverksstruktur. Systemer setter ulike krav, da ytelsen for enkelte tjenester er direkte knyttet til overføringshastighet. Eksempelvis er det helt avgjørende for et overvåkingssystem at data som blir representert ikke er utdatert. Geografisk plassering av fabrikken kan påvirke kostnaden ved utvidelse av kapasitet.

En av de største fordelene med skytjenester er den dynamiske tilgangen på ressurser. Den umiddelbare tilgangen til skalert prosessorkraft og lagring gir stor fleksibilitet. Tjenester som er optimalisert for skyen vil yte bedre og

dermed koste mindre, da man kun betaler for de ressursene man bruker. En løsning som ikke er utviklet for skyen, vil ikke ha spesielt utbytte av fordelene en skytjeneste tilbyr. Det er derfor viktig at utviklere planlegger spesifikt for ytelse i skyen [2].
Det er viktig å nevne at man bør være oppmerksom på vendor lock-in [7]. For høyest mulig fleksibilitet bør ikke applikasjonen være optimalisert for én spesifikk skytjeneste, men skyen generelt.

Utdraget fra Nasjonal strategi for bruk av skytenester, viser nytteverdien av dynamisk ressurstildeling [4]:

> *Ï desember 2010 flytta BaneDanmark informasjonsnettsida si til ei skyteneste (Microsoft Azure). Om vinteren blei det store problem med transporttenestene i Danmark. Dei andre transportselskapa opplevde at informasjonstenestene svikta på grunn av stor pågang frå publikum. Dette skjedde ikkje med Banedanmark. På det meste hadde dei 5,5 millionar brukarar på ein dag, mot normalt 50 000. Den auka kapasiteten dei trong i denne perioden betalte dei 179 DKK for.¨*

2.2 Kostnad

Stadig flere bedrifter benytter seg i dag av skytjenester [10]. Eksempelvis har regjeringen i Storbritannia innført preferansepolitikk på skytjenester grunnet ønsket om økt standardisering og redusering av kostnader [6]. Skytjenester reduserer ikke bare kostnaden, men kan også øke bedriftens fokus på egen virksomhet og legge til rette for nyskapning. Lavere risiko og mindre behov for nødvendig startkapital kan medføre økt tålmodighet ved innovasjon i organisasjonen. Et eksempel er Telenor sin Comoyo satsing [4]:

> *C̈omoyo var Telenor si satsing på strøyme-TV. Tenesta blei etablert allereie i 2011. Så seint som i mai 2013 uttalte Telenor: «Med nyetablerte Comoyo skal Telenor kapre 130 millionar forbrukarar i alle kanalar og på alle plattformer.» Telenor avvikla Comoyo i november 2013, etter at store internasjonale aktørar som Netflix*

og HBO etablerte seg med strøyme-TV i Norden og tok det meste av marknaden. Telenor brukte Amazon sin infrastruktur til å levere tenesta på. Dermed betalte dei berre for den kapasiteten dei trong for å betene dei kundane dei hadde til kvar tid. Då tenesta blei lagt ned, satt dei derfor ikkje igjen med store investeringar i infrastruktur dei ikkje lenger hadde bruk for.¨

Lokale tjenester har den ulempen at skalering må skje gjennom nye investeringer i hardware og software. En skytjeneste krever derimot ingen investering utover allokering av nye ressurser. I tillegg til kostnader ved skalering, må en lokal server driftes. Det innebærer ressurser til oppsett, vedlikehold, oppgraderinger og overvåkning. En skyløsning vil også redusere eller eliminere behovet for å drifte en lokal løsning, og dermed redusere antall årsverk.

Både lokal- og skyløsninger krever infrastruktur i henhold til nettverk, men hvilken som er mest kostnadseffektiv krever en dypere granskning.
Løsninger med komplisert arkitektur og tett integrasjon med eksisterende systemer kan gjøre en skyløsning mindre kostnadseffektiv sammenlignet med en lokal løsning. En applikasjon som er designet for bruk i skyen vil ha bedre ytelse og gi lavere kostnader, men arkitekturen og designet av tjenesten er avgjørende.
Ved at forbrukere samler ressursene på ett sted, er energieffektiviteten ved bruk av skytjenester høyere. Dette er mer effektivt kontra at hver kunde har sine egne datasentre [4].

2.3 Sikkerhet

"Bruk av skytjenester er kommet for å bli, men det er viktig å ikke se seg blind på lavere kostnader alene." sier NSM i sitt Helhetlige digitale risikobilde for 2020 [5]. Den tidligere nevnte undersøkelsen av Vanson Bourne [12], viser at nærmest halvparten av de 700 store teknologiselskapene som ble intervjuet var bekymret for sikkerheten i skyen. NSM stiller seg positive til at virksomheter tar i bruk skytjenester, så lenge det gjennomføres gode vurderinger i forkant [5].

Regjeringen ønsker at IKT-virksomhet skal gjøres sikrere og mer kostnadseffektiv, og vil derfor gjøre det lettere for virksomheter å vurdere skytjenester som et alternativ. I Nasjonal strategi for bruk av skytjenester har de derfor gjort rede for noen viktige vurderinger som bør gjøres før en anskaffer

skytjenester. Disse rådene gir virksomheter et overblikk over de viktigste faktorene for å benytte skytjenester på forsvarlig vis [4]. Nedenfor er punktene gruppa anså som sikkerhetsrelevante beskrevet og koblet opp mot relevant tematikk:

- **Sourcing** dreier seg om hvilke valg en virksomhet foretar seg når de skal vurdere hvilke tjenester de skal håndtere eller drifte selv, og hvilke ansvar som skal delegeres eksterne aktører.

  Kjøp av skytjenester vil være et slikt valg, da skyleverandørene vil stå ansvarlig for varierende deler av driften ut fra valg av tjenestemodell (1.4). Da skyleverandørene gjerne satser tungt på sikkerhet [1], anser NSM dette aspektet ved skytjenester som fordelaktig, spesielt for mindre bedrifter [5]. Dette på bakgrunn av at bedriftene gjerne ikke har kapasitet eller kompetanse til å utføre nødvendig sikkerhetsarbeid.

- **Informasjonssikkerhet** er et sett med prinsipper utrettet for å håndtere risiko relatert til data innad i en virksomhet. Nedenfor er disse prinsippene kort oppsummert og diskutert:

  Konfidensialitet - forhindre at data aksesseres av uvedkommende.

  Risikoen knyttet til oppbevaring av data i skyen, varierer med hvor sensitiv dataen er [4]. Dersom det er snakk om personopplysninger eller annen sensitiv informasjon, må bedrifter forholde seg til et strengere regelverk [9]. Regjeringen anbefaler at kunder gjennomfører en grundig kontroll av leverandør før en eventuell anskaffelse [4].

  Mange virksomheter stiller seg skeptisk til å legge data i skyen, da de føler på tap av kontroll når data håndteres av en tredjepart. Regjeringen har utarbeidet ulike mekanismer for å redusere denne frykten [4]. FFI fant i sin undersøkelse i 2018 ingenting som tydet på at datasentere som leverer skytjenester er mer sårbare for angrep enn tradisjonelle datasentere [1]. Dette står i tråd med NSM sin erfaring om at hovedandelen uønskede sikkerhetsrelaterte hendelser var grunnet konfigurasjon- eller brukerfeil, ikke sårbarheter hos leverandør [5]. I likhet med andre områder innen informasjonssikkerhet, har også skyleverandører egne

sertifiseringer[1]. I tillegg til spesialiserte driftsavdelinger, har skyleverandørene fysiske sikringer ved sine serverparker med strenge restriksjoner. Disse parkene er ofte sertifisert basert på nivået av sikkerhet ved anleggene[2].

Integritet - forhindre at data ikke kan endres av uvedkommende.

Overordnet har vi ikke funnet noe konkret som tilsier at det er store forskjeller mellom skytjenester og lokal infrastruktur som påvirker dataintegritet. Uansett hvilken modell man bruker, er det viktig å følge gjeldende retningslinjer for å ivareta integritet.

Tilgjengelighet - data skal være tilgjengelig ved behov.

Som nevnt i 1.4, er en essensiell karakteristikk ved skytjenester at ressursene er fleksible, og at de enkelt kan skaleres både opp og ned. Denne fleksibiliteten skaper redundans ved at dersom en server skulle bli utilgjengelig, kan tilsvarende ressurser spinnes opp umiddelbart for å dekke behovet til feilen er fikset. Denne egenskapen er svært fordelaktig, spesielt hvis dataene anses som kritiske, og vil være utfordrende å replikere ved bruk av lokal infrastruktur. Skytjenester er lokasjonsuavhengige, som gjør de helt avhengig av internett for å kunne aksesseres (1.4). En virksomhets nettverkstilkobling er derfor å anse som en sårbarhet ved bruk av skytjenester [1].

3 Konklusjon

Initielt i undersøkelsen stilte vi følgende spørsmål:

Hvordan påvirker faktorene ytelse, risiko og kostnad en migrering fra lokal infrastruktur til sky?

For å svare på dette har vi først introdusert hva skytjenester er, og deretter undersøkt hvordan de utvalgte faktorene påvirker en potensiell migrering.

En av de største fordelene med skytjenester er den dynamiske tilgangen på ressurser. For å utnytte denne fleksibilitene til det fulle, må systemene

[1] https://cloudsecurityalliance.org/star/
[2] https://datasenter.basefarm.no/datasenter/sikkerhet/

optimaliseres spesifikt til skytjenester. Dette kan medføre risiko for ytterligere kostnader og vendor lock-in. Skytjenester er helt avhengige av internett, og det stilles derfor større krav til pålitelig eksternt nettverk enn ved lokal infrastruktur.

En migrering til sky anses i all hovedsak som en fordelaktig beslutning med tanke på risiko. Den dynamiske tilgangen til ressurser skaper en redundans som er vanskelig å replikere ved bruk av lokal infrastruktur. Leverandørene har spesialisert kompetanse innen sikkerhet og drift, og er derav ofte bedre egnet til å gjennomføre sikkerhetsarbeid enn bedriften selv. For å unngå bekymringer knyttet til tap av kontroll er tillit til leverandørene essensielt.

En annen styrke med skytjenester er at man betaler for nøyaktig det man trenger, ved at ressurser enkelt kan allokeres og frigjøres. Lokale tjenester tilbyr ikke samme fleksibilitet, da de krever investering i infrastruktur ved skalering. Denne fleksibiliteten gir bedrifter frihet til å fokusere på egen virksomhet og kan legge til rette for nyskapning.

Basert på våre undersøkelser, støtter faktorene ytelse, risiko og kostnad oppunder en potensiell migrering til skyen. På generelt grunnlag vil vi anbefale Argos å ta i bruk skytjenester, med forbehold om at faktorene bør vurderes ytterligere ved hvert migreringstilfelle.

Referanser

[1] Arild Bergh Ketil Lund Johnsen Frank Trethan. *Bruk av skytjenester i Forsvaret - muligheter og utfordringer*. URL: https://www.ffi.no/publikasjoner/arkiv/bruk-av-skytjenester-i-forsvaret-muligheter-og-utfordringer. (accessed: 08/02/2021).

[2] D. S. Linthicum. «Approaching Cloud Computing Performance». I: *IEEE Cloud Computing* 5.02 (mar. 2018), s. 33–36. ISSN: 2372-2568. DOI: 10.1109/MCC.2018.022171665. (accessed: 02/02/2021).

[3] Microsoft. *Datalagring i Azure*. URL: https://azure.microsoft.com/nb-no/global-infrastructure/data-residency/. (accessed: 04/02/2021).

[4] Kommunal- og moderniseringsdepartementet. *Nasjonal strategi for bruk av skytenester*. URL: https://www.regjeringen.no/no/dokumenter/nasjonal-strategi-for-bruk-av-skytenester/id2484403/. (accessed: 08/02/2021).

[5] NCSC NSM. *Helhetlig digitalt risikobilde 2020*. URL: https://nsm.no/getfile.php/134267-1601027852/Demo/Dokumenter/Rapporter/NSM_IKT-risikobilde_2020_1609_LR.pdf. (accessed: 09/02/2021).

[6] Cabinet Office, Efficiency og UK Reform Group. *Government cloud strategy*. 2011. URL: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/266214/government-cloud-strategy_0.pdf. (accessed: 01/02/2021).

[7] Justice Opara-Martins, R. Sahandi og Feng Tian. «Critical analysis of vendor lock-in and its impact on cloud computing migration: a business perspective». I: *Journal of Cloud Computing* 5 (apr. 2016). DOI: 10.1186/s13677-016-0054-z.

[8] Timothy Grance Peter M. Mell. *The NIST Definition of Cloud Computing*. URL: https://www.nist.gov/publications/nist-definition-cloud-computing. (accessed: 08/02/2021).

[9] Regjeringen. *Ny personopplysningslov*. 2019. URL: https://www.regjeringen.no/no/tema/statlig-forvaltning/personvern/ny-personopplysningslov/id2340094/. (accessed: 01/02/2021).

[10] SSB. *Halvparten av norske føretak kjøper nettskytenester*. 2017. URL: https://www.ssb.no/teknologi-og-innovasjon/artikler-og-publikasjoner/halvparten-av-norske-foretak-kjoper-nettskytenester. (accessed: 04/02/2021).

[11] Narendra Rao Tadapaneni. «Cloud Computing: Opportunities And Challenges». I: *Available at SSRN 3563342* (2018). (accessed: 02/02/2021).

[12] Teradata. *Survey: Companies are Bullish on Cloud Analytics, But Need to Speed Up the Pace*. 2018. URL: https://www.teradata.com/Press-Releases/2018/Survey-Companiesare-Bullish-on-Cloud-Analyt. (accessed: 04/02/2021).

F Retrospektiv 2

06/03 Retrospektiv 2

Navn	Type	Kommentar	Respons	Oppfølging
@ Lars Pedersen	Positiv	Vi har stått på for å få ferdig prototype. Bra jobba! Deilig å få positiv tilbakemelding fra Tom og Argos.	@ Andreas Nygård Ljøterud Helt enig. Bra jobba gutta!	
@ Lars Pedersen	Positiv/ Nøytral	Funker bra å ha digitale møter i samme rom.	@ Andreas Nygård Ljøterud +1	
@ Lars Pedersen	Nøytral	Bruker vi av og til litt lang tid på ikke faglige diskusjoner? Skal vi heller sette av en lengre lunsj og så jobbe mer fokusert før og etter?	@ Andreas Nygård Ljøterud Ref. mitt punkt under her om strukturert jobbing mot pauser.	@ Andreas Nygård Ljøterud Vi bestemte oss for å prøve ut forskjellige tidsboksingsmetoder. Pomodoro (25+5x4) skal testes på tirdag og så skal vi ta videre oppfølging på onsdag morgen.
@ Lars Pedersen	Negativ	Føler ikke vi fikk utført alt det vi sa vi skulle gjøre i forrige sprint planlegging	@ Andreas Nygård Ljøterud Jeg føler vi ikke har nødvendig informasjon enda, da tilbakemeldingen fra Argos er noe vag f.eks med tanke på rapport. Altså, hva de faktisk er ute etter, med spesifikke bruksområder. Dette håper jeg vil løses på møtet med “ulvene” i Argos på onsdag.	
@ Mads Olsen Nekkøy	Nøytral	Få ned issues i Jira med en gang, klare mål å jobbe med.	@ Lars Pedersen +1. @ Andreas Nygård Ljøterud +1	☑ Foreslår at dette er noe av det første vi gjør hver morgen.
@ Mads Olsen Nekkøy	Nøytral	Jobbe sammen på skolen fungerer bedre enn via Teams.	@ Lars Pedersen Mye enklere å hjelpe hverandre når vi sitter sammen.	
@ Andreas Nygård Ljøter	Positiv	Romreservasjonene har fungert bra. Stemmer for at vi fortsetter med det, men at vi bytter mellom rom fra dag til dag for å skape variasjon.	@ Lars Pedersen Skal vi prøve å reservere rom som har tv for dager vi har digitale møter? @ Andreas Nygård Ljøterud +1	
@ Andreas Nygård Ljøter	Negativ	Stemmer for at vi holder oss unna politiske diskusjoner i lunsjen, da det ofte resulterer i at vi går langt over tiden.	@ Mads Olsen Nekkøy +1	
@ Andreas Nygård Ljøter	Negativ	Jeg har erfart at i den siste perioden har vi endt opp med å sitte lenge uten pauser. Ref. mitt forrige punkt tenker jeg at dette kan være et resultat av dette. Dermed foreslår jeg heller å gå for en 45 /15 løsning eller lignende, slik at pausene blir mer tydelige. Dette medfører dog at vi må være mer strukturert med å faktisk begynne å jobbe igjen når de 15 minuttene er omme.	@ Andreas Nygård Ljøterud Ref. over	
@ Andreas Nygård Ljøter	Negativ	Jeg tenker det kan være en god ide å legge inn sparrebolker ila. dagene, slik at man ikke nødvendigvis context switcher så mye. Jeg har opplevd at en slik switching kan bli forstyrrende for eget og andres arbeid. At vi heller sier at vi jobber frem til xx:yy og tar da en runde på om det er noen som trenger litt input fra gruppa.	@ Lars Pedersen +1. Føler at jeg ikke før gjort så mye pga distraksjoner. @ Mads Olsen Nekkøy +1 Jeg må bruke mer tid på å researche problem. ☑ Husk å notere ned at en nevneverdig kritikk til gruppa er at vi er tre stykker, slik at pararbeid innad i gruppa ikke er like enkelt. @ Andreas Nygård Ljøterud	☑ Planen er foreløpig å kjøre 2 Pomodoro runder og deretter ha en sparrebolk som evt. kan utvides til å vare gjennom neste runde eller ikke. @ Andreas Nygård Ljøter

Prototype 2:

1. Sende all data til dashboard og rapport gjennom API-et.
2. Innsetting av data i databasen samtidig som vi har et dashboard som oppdateres fortløpende.
3. Få til en protype av egenskrevet rapport-generator som kan hente data fra databasen og vise denne dataen i pdf eller csv.
4. Produsere intervju basert på DD og utføre dette på “ulvene”.
5. Justere eksisterende simulering til å ta høyde for at flere feiltyper individuelt kunne vært worst defect på samme plate.
6. (Til tirsdag uke 11) Ha første ramme av bachelorrapport klar.
7. (Til lørdag uke 10) Definere klart hva sluttproduktet i hver epic vil være.

- [ ] Notere ned at en “tutorial” for bruk av Grafana/API/Rapport kan være kjekt. @ Andreas Nygård Ljøterud
- [x] Undersøke om Argos er interessert i optimalisering av simuleringsparametre. (Eksempel under) @ Andreas Nygård Ljøterud

Feiltype	Nåværende limit	Ny limit	Produksjonsøkning
Feiltype_1	3	4	15%
Feiltype_2	4	6	28%
Feiltype_3	2	5	60%

Feiltype	Nåværende limit	Ny limit	Produksjonsøkning
Feiltype_1	3	4	
Feiltype_2	4	5	22%
Feiltype_3	2	3	

Hva er den laveste økningen kombinasjonen av feil man kan gjøre sammen med høyest mulig produksjonsøkning?

G Retrospektiv 3

20/03 Retrospektiv 3

Navn	Type	Kommentar	Respons	Oppfølging
@ Lars Pedersen	Positiv	Bra møte med selgerene	@ Mads Olsen Nekkøy Må huske å fullføre double diamond prosessen.	☐ Snakke med Eivind angående de 2 siste delene av DD og utføre disse @ Andreas Nygård Ljøterud @ Mads Olsen Nekkøy
@ Lars Pedersen	Positiv	Føler at arkitekturen vi har valgt for backend er fleksibel, godt valg der.	@ Andreas Nygård Ljøterud +1 Enig, bra jobba @ Lars Pedersen !	☐ Få med hvor enkelt det var å utvide pluginmanageren til å tjene statiske nettsider i b. rapport @ Lars Pedersen
@ Lars Pedersen	Nøytral	Føler at "farten" i prosjektet har sakket litt ned. Holder vi tempoet som kreves?		Har det i bakhodet videre under prosessen.
@ Mads Olsen Nekkøy	Positiv	Tidsbolkingen fungerer bra.	@ Andreas Nygård Ljøterud Enig +1 @ Mads Olsen Nekkøy Liker at før hvert intervall må vi spesifisere hva som skal gjøres på den aktuelle sprinten.	
@ Andreas Nygård Ljøterud	Positiv	Det var en del usikkerhet knyttet til rapport-delen. Jeg synes vi sto på, diskuterte og endte opp med å løse det på en god måte.	@ Mads Olsen Nekkøy +1	
@ Andreas Nygård Ljøterud	Nøytral	Måten vi har tidsbolket på den siste tiden har fungert godt. Vi bør bli litt flinkere på å passe på hvilket rundenummer vi er på, slik at det ikke blir 25+5 hele dagen, men at vi får inn noen lengre pauser slik det er ment.	@ Mads Olsen Nekkøy +1. Vi bør prøve å holde styr på hvilken runde /sprint vi er på.	
@ Andreas Nygård Ljøterud	Negativ	Forrige sprint-planlegging ble noe vag på bakgrunn av manglende informasjon. Vi bør prøve å definere noe klarere mål denne gangen og bruk Jira mer aktivt.	@ Andreas Nygård Ljøterud Dette tror jeg automatisk vil bli litt lettere nå, da vi skal gå inn i en mer "konkret" fase.	
@ Andreas Nygård Ljøterud	Negativ	Selv om det er viktig at vi ikke sitter "å jobber med det samme alle sammen", så er det viktig at man teamer opp dersom noen er stuck på et eller annet. Det føler jeg vi har vært litt dårlig på denne perioden, da hvert fall både jeg og Mads har brukt mye tid på ett hinder.		Kan spørre om hjelp til neste sprint. Slik at andre kan bli ferdige med sine ting på den nåværende sprinten og planlegge for å hjelpe ved neste.

- "Say-something-syndrome"
  - Interessant refleksjon fra Lars som går ut på at ved disse retrospektivene trenger man nødvendigvis ikke si noe bare for å si noe.

Neste sprint:

Rapport:

1. Ferdigstille PoC
   a. Vise 2 "kopier" av eksisterende rapporter, men også vise muligheter til endring
      i. Lage noen nye rapporter, kanskje også muligens med noen av de "nye" parameterene fra salgsavdelingen

1. Lars: @Andreas Nygård Ljøterud, kan du finne ut av om det er mulig å hente data fra andre kilder enn SQL-databasen inn i rapportene? (Temperatur, fuktighet osv. fra annen kilde)
2. Hvis dette er mulig er det kanskje en mulighet å få inn simuleringen her også?

Dashboard:

1. Videre på https://bidat39.atlassian.net/browse/BAC-28
    - Hente ekstern data(temp, fuktighet) og få det inn i Grafana gjennom API-et
    - Lage "ferdig produkt" med mal til Argos > Teste det som en helhet for å finne eventuelle mangler
        - Husk å kryss-referere med krav fra salgsavdeling
        - Observerer Ole når han setter det opp(UX/brukertesting)
2. Strategi for videre arbeid med design
3. Heatmap

Simulering:

1. Sette maksgrenser på foreslåtte/egenvalgte verdier.
2. n-delt klassifisering istedenfor ok/vrak
3. Ignorere plater som ellers har vrak-defekter.
4. Velg type / subtype istedenfor ArogsName.
5. Velg alle mulighet.
    - @Andreas Nygård Ljøterud: Mulig å få opp en ved siden av navnene på denne oversikten for å utheve ønskede defekter.

- Begynne på b-.rapport.

H Oppstartsmøte med veileder

12/01 - Veiledningsmøte 1

Dato: 12 Jan 2021

Deltagere: @ Andreas Nygård Ljøterud @ Lars Pedersen @ Mads Olsen Nekkøy Tom Røise

Agenda:

- Introduksjon
- Tips fra Tom
    - Platform
    - Logg og tidsstyring
- Annet

Lars forteller innledningsvis om hvordan vi kom i kontakt med arbeidsgiver og generelt om bedriften ARGOS. Deretter noe spesifisering rundt hva oppgaven dreier seg om. Vi mangler dog noe informasjon rundt viktige avgjørelser, men en stor del av dette håper vi vil komme fra møtet vårt med ARGOS førstkommende fredag (15.01).

Ift. platform bør vi høre med ARGOS om hvor data kan ligge. Utover det kan vi velge platform selv, men vi bør være obs på at den er stabil etc.

Vi diskuterer videre rundt hvordan vi som gruppe endte opp sammen, mål innad i gruppa, forventninger til arbeidsinnsats osv.

Vi skal holde logg fordi det er en profesjonell praksis, men også fordi vi skal ha grunnlag til å skrive utfyllende om avgjørelser osv. i rapporten. Loggen må inneholde antall timer som hver person har brukt på diverse arbeidsoppgaver. Hvordan vi holder styr på antall timer er opp til oss.

Send epost til Erik om SkyHigh

Er det noen andre ressurser vi bør sette oss inn i?

- Boka om bachelor-skriving, ikke perm til perm, men kan være hjelpsom.
- Begynne å se på tidligere bachelor-oppgaver/rapporter.
    - Velg hver deres rapport, hva kan vi ta med oss?
- Start tidlig med å bestemme MÅ og KJEKTÅHA
    - Uklarhet litt for lenge kan være veldig ødeleggende
    - Husk å sjekke læringsmål i emnebeskrivelsen
- Ikke glem rapporten underveis!
- Grupperegler
- Hold fokus på de viktigste tingene av gangen.
    - Ikke heng seg opp i detaljer.
- Tilpass systemer basert på hva som fungerer på gruppa.
    - DOKUMENTER.
- Agenda til Tom minimum ett døgn i forveien, men to døgn hvis mengden er større.

Action points:

- [x] Send epost til Erik om Skyhigh prosjekt

I Møtereferat fra møte med Argos

21/01 - Feedback på forprosjekt fra Argos

Dato: 21 Jan 2021

Deltagere: @ Andreas Nygård Ljøterud @ Lars Pedersen @ Mads Olsen Nekkøy

Agenda:

- Feedback på forprosjekt
- Annet

Argos var fornøyde med utkastet. Hadde ikke så mange kommentarer til selve rapporten.

- Rammer:
  - Datamengde, mulig dette skal inn i kravspek. Høre med Tom.
- Effektmål
  - Penger, ikke bare omdømme
  - Abonnementsordningen
- Sanntid, natt og dag sammenligning? Dette snakket vi om tidligere?
  - Dette kunne vært interessant.
  - Femskiftsordning i USA - konkurranse?
  - Rammeverk, men nødvendigvis ikke kjernevirksomhet
- Det er også viktig å presentere data fra de individuelle kildene/linjene hver for seg også, side om side.
  - Intern konkurranse er veldig vektlagt i industrien.
  - Sammenligning av interne produksjonslinjer
- Fysisk oppmøte på neste møte?
  - Nei
- Endringslogging? Kanskje ikke relevant da det ikke er eksisterende data i db knyttet til dette i dag
- Ingen gammel maskinvare å ta hensyn til.
  - Det er ikke ønskelig at vi belaster de nåværende maskiner da de har “nok å styre med”
  - Hvis mye tygging > dedikert
  - Hvis det er kompatibelt med det de selger i dag er det godt nok
- Sky
  - Husk å undersøke båndbredde ved undersøkelser

⚠ Det kunne vært interessant med sammenligningsmuligheter mellom skiftene. F.eks natt/dag, skiftlag osv. Dette anses ikke som kjernefunksjonalitet, men om rammeverket kunne støttet senere implementasjon, hadde det vært kjekt.

⚠ Vi trenger ikke ta hensyn til at kunder innehar gammel maskinvare. Dersom vår tjeneste er kompatibel med den maskinvaren de selger i dag, er dette godt nok. Hvis det skal tygges mye data, er det bedre å ta i bruk en dedikert maskin, da nåværende maskiner allerede har stor belastning.

Det blir ikke fysisk oppmøte på neste møte, da dette ikke er gjennomførbart med nåværende COVID-retningslinjer

- Potensiell videreutvikling
  - Det vil antageligvis være ønskelig å sette opp alarmer som trigger ved visse kriterier.
    - F.eks. nå har du produsert 50% ødelagte plater. Her må du undersøke.
    - Viser at det er mange forskjellige bruksområder for dette vi må forholde oss til.

J Rapporteksempel - Grafana Reporter

Production Report

This is an example of a production report description.

Wed Feb 24 00:00:00 CET 2021
to
Wed Feb 24 01:00:00 CET 2021

Total

313

Accept

1

Repair

306

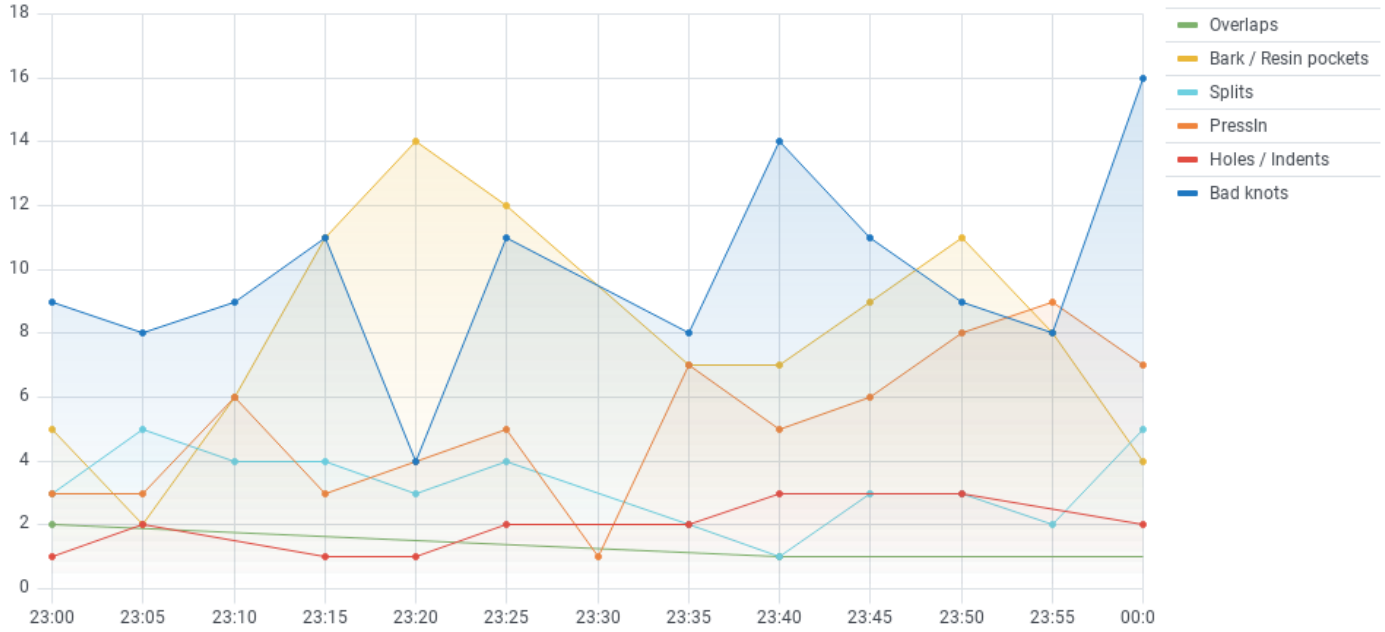
Reject

6

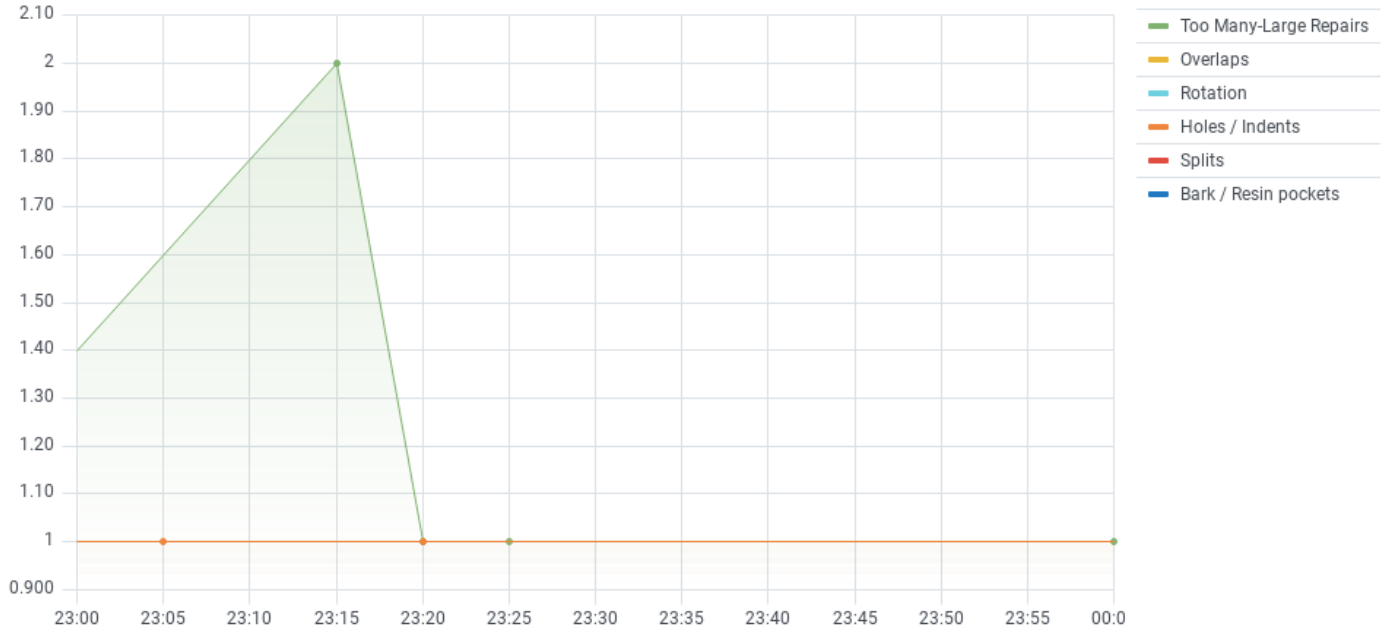
Downgrade by Type

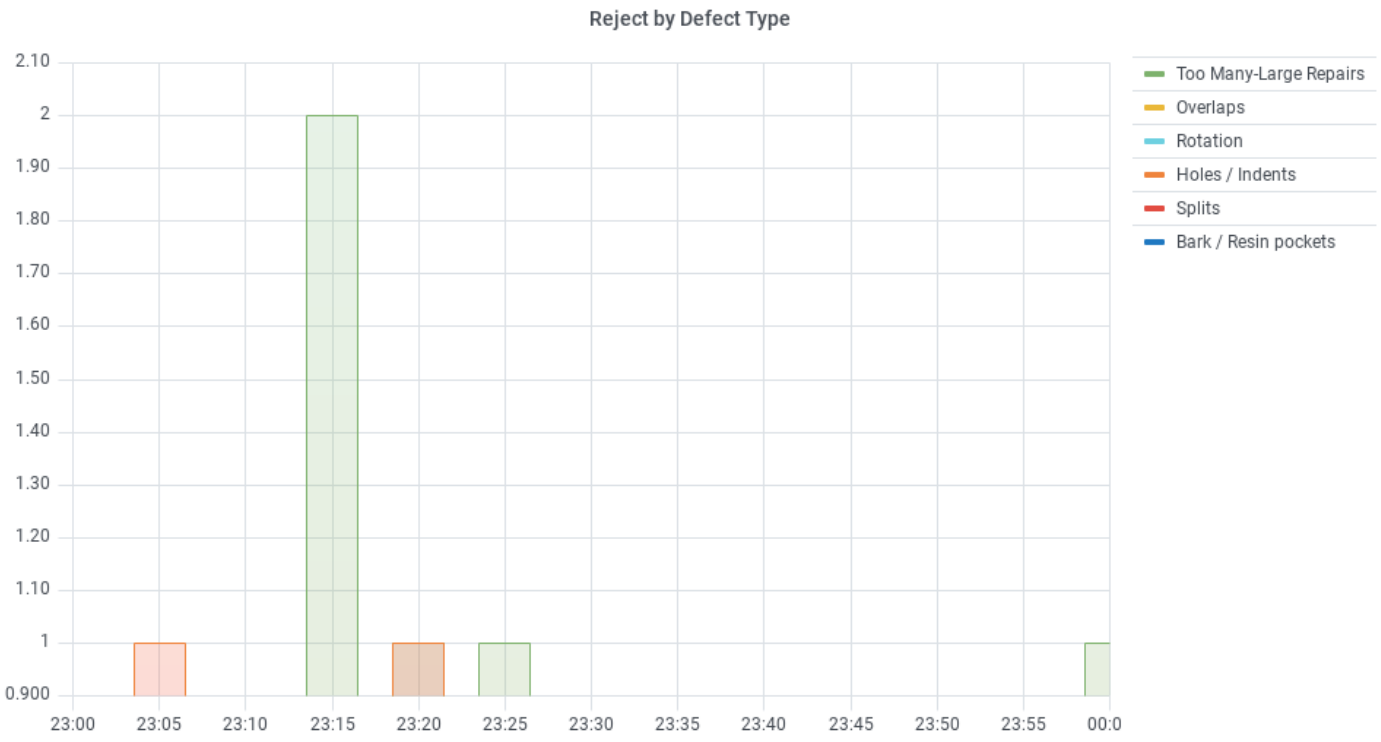
Type	Repair	Reject	Total
Bad knots	102	0	102
Bark / Resin pockets	92	0	92
PressIn	60	0	60
Splits	34	0	34
Holes / Indents	15	2	17
Too Many-Large Repairs	0	4	4
Overlaps	3	0	3
Rotation	0	0	1

Repair by Defect Type



Reject by Defect Type




Reject by Defect Type
2.10
2
1.90
1.80
1.70
1.60
1.50
1.40
1.30
1.20
1.10
1
0.900
23:00
23:05
23:10
23:15
23:20
23:25
23:30
23:35
23:40
23:45
23:50
23:55
00:0
Too Many-Large Repairs
Overlaps
Rotation
Holes / Indents
Splits
Bark / Resin pockets

K Kodeeksempel: /query

```
1
2  class TimeRangeType(BaseModel):
3      """Dict of from and to time range.
4
5      Used in :class:`JSONQueryInputType`
6
7
8      More detailed documentation can be found in Swagger.
9      """
10
11     from_: str = Query(..., description="From time in iso format (
           YYYY-MM-dd HH:mm:ss)")
12     to: str = Query(..., description="To time in iso format (YYYY-
           MM-dd HH:mm:ss)")
13
14     class Config:
15         fields = {"from_": "from"}
16
17
18 class ScopedVarsModel(BaseModel):
19     """Dict of scoped vars.
20
21     Used in :class:`JSONQueryInputType`
22
23     More detailed documentation can be found in Swagger.
24     """
25
26     value: Union[List[str], str, int] = Query(
27         ..., description="The value of the scoped variable."
28     )
29
30
31 class TargetsModel(BaseModel):
32     """Model for describing targets (=the plugin names)
33
34     Used in :class:`JSONQueryInputType`
35
36     More detailed documentation can be found in Swagger.
37     """
38
39     target: str = Query(..., description="Target plugin name")
40     data: Any = Query(..., description="Extra data for the plugin."
           )
41
42
43 class JSONQueryInputType(BaseModel):
44     """Final model of the input type for :func:`query`.
45
46     More detailed documentation can be found in Swagger.
47     """
48
49     range: TimeRangeType
50     intervalMs: int = Query(..., description="Time separation
           interval in ms.")
```

```
51      scopedVars: Dict[str, ScopedVarsModel] = Query(
52          ..., description="Variables defined in the dashboard."
53      )
54      targets: List[TargetsModel] = Query(..., description="Target
            plugins.")
55
56
57  class JSONQueryTimeseriesResponseModel(BaseModel):
58      """Response model for timeseries-like responses from :func:`
            query`.
59
60      The other alternative is :class:`JSONQueryTableResponseModel`
61
62      More detailed documentation can be found in Swagger.
63      """
64
65      target: str = Query(..., description="The name of the dataset."
            )
66      datapoints: List[Tuple[int, int]] = Query(
67          ..., description="List of datapoints for the dataset."
68      )
69
70
71  class JSONQueryTableColumn(BaseModel):
72      """Define the columns for :class:`JSONQueryTableResponseModel`
73
74      More detailed documentation can be found in Swagger.
75      """
76
77      text: str = Query(..., description="The name of the column")
78      type: str = Query(..., description="Datatype of the column")
79
80
81  class JSONQueryTableResponseModel(BaseModel):
82      """Response model for table-like responses from :func:`query`.
83
84      The other alternative is :class:`
            JSONQueryTimeseriesResponseModel`
85
86      More detailed documentation can be found in Swagger.
87      """
88
89      columns: List[JSONQueryTableColumn]
90      rows: List[Any] = Query(
91          ...,
92          description="Rows of the table. Must have as many elements
                as count of columns.",
93      )
94      type: str = "table"
95
96
97  @router.post(
98      "/query",
99      response_model=Union[
100         List[JSONQueryTimeseriesResponseModel], List[
                JSONQueryTableResponseModel]
101     ],
```

```
102 )
103 def query(request: Request, inp: JSONQueryInputType):
104     """Query the database based on the supplied plugins
105 
106     Query the database for data based on the selected target.
107     Target must be defined in plugins with\
108     :meth:`ArgosAnalysePlatform.plugin_manager.PluginManager.
        register_plugin`
109     """
110     inp = inp.dict()
111 
112     start_time = isoparse(inp["range"]["from_"])
113     end_time = isoparse(inp["range"]["to"])
114     resolution = inp["intervalMs"] // 1000 // 60
115 
116     scanners = inp["scopedVars"]["scannere"]["value"]
117     metrics = [(t["target"], t["data"]) for t in inp["targets"]]
118     response = []
119     for metric, data in metrics:
120         for name, session in request.state.db.sessioniter(scanners)
            :
121 
122             response.extend(
123                 plugmanager.call_plugin(metric, "")(
124                     session,
125                     name,
126                     start_time,
127                     end_time,
128                     resolution,
129                     extra_data=data,
130                 )
131             )
132 
133     return response
```

L Kodeeksempel: _query_pareto

```python
def _query_pareto(
    session: Session,
    where_argosnames: List[str],
    start_time: datetime,
    end_time: datetime,
) -> Dict[Any, List[Tuple[Any, int]]]:
    """Query required data to make pareto charts.

    Returns a list of argos names (filtered by parameter), amount of boards
    grouped by the am

    :param session: Database session
    :param where_argosnames: A list of the argosnames to filter on.

    :return: Dict of dict of the results, grouped first by class, then type/defect count

    """
    Defects = DatabaseConnections.tables["Defects"]
    Boards = DatabaseConnections.tables["Boards"]

    # Get all boards that has the selected where_argosnames as worst defect.
    worst_subq = (
        session.query(Defects.c.BoardID, Defects.c.ArgosName)
        .join(Boards)
        .filter(Boards.c.Time.between(start_time, end_time))
        .filter(Defects.c.IsWorstDefect == 1)
        .filter(Defects.c.ArgosName.in_(where_argosnames))
        .subquery()
    )

    old_class = func.max(Defects.c.Class)
    # Get the highest class that is not worst defect.
    new_class = func.max(
        case(
            [(Defects.c.ArgosName != worst_subq.c.ArgosName, Defects.c.Class)], else_=0
        )
    )

    filter_values = (
        session.query(
            Defects.c.BoardID,
            old_class.label("old_class"),
            new_class.label("new_class"),
        )
        .join(
            worst_subq,
            worst_subq.c.BoardID == Defects.c.BoardID,
        )
        .group_by(Defects.c.BoardID)
```

```
50          # Only return boards that actually change class if worst
                defect is
51          # excluded.
52          .having(new_class < old_class)
53          .subquery()
54      )
55
56      query = (
57          session.query(
58              Defects.c.BoardID.label("bid"),
59              Defects.c.ArgosName.label("type"),
60              func.count(Defects.c.ID).label("defect_count"),
61              filter_values.c.old_class,
62              filter_values.c.new_class,
63          )
64          .join(
65              # Only include boards that are part of where_argosnames
                    and that
66              # has them as worst.
67              worst_subq,
68              and_(
69                  worst_subq.c.BoardID == Defects.c.BoardID,
70                  worst_subq.c.ArgosName == Defects.c.ArgosName,
71              ),
72          )
73          # Exclude boards that would not be affected by the change.
74          .join(filter_values, filter_values.c.BoardID == Defects.c.
                BoardID)
75          .group_by(
76              Defects.c.BoardID,
77              Defects.c.ArgosName,
78              filter_values.c.old_class,
79              filter_values.c.new_class,
80          )
81          # .subquery()
82          .order_by(new_class, Defects.c.ArgosName, func.count(
                Defects.c.ID))
83          .all()
84      )
85
86      groupby_new_class = {}
87
88      # Group the result by new class
89      for k, v in groupby(
90          sorted(query, key=lambda x: x.new_class), key=lambda x: x.
                new_class
91      ):
92          # Group the result by its type and defect count
93          groupby_type_defectcount = [
94              (*k2, len(list(v2)))
95              for k2, v2 in groupby(v, key=lambda x: (x.type, x.
                    defect_count))
96          ]
97
98          groupby_new_class[k] = groupby_type_defectcount
99
```

```
100     # Dict of the reults grouped first by class, then by type/
            defect count.
101     return groupby_new_clas
```

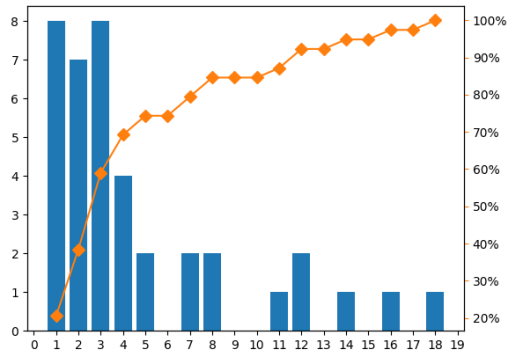
M Web-applikasjon for simuleringsverktøy

Start time 2021/02/16 11:15
End time 2021/05/12 11:15
Scannere scanner1
Remove scanner1
Typer / Subtyper All Bark / Resin pockets
Remove Resin Pocket rout-size
Suggest new limit Manual simulation

Subtype	Argos Name	Class limit	Old value	New value
Resin Pocket rout-size	MediumResinPocketsMDD	1_2	0.9	3
Resin Pocket rout-size	MediumResinPocketsMDD	2_3	999	999

Simuler

Argos Name	New class	Old limit	New limit	Production increase	%	Scanner	
MediumResinPocketsMDD	0	0	3	23	8.1	scanner1	Pareto chart



N Utdrag av opprinnelig produksjonsrapport

ARGOS

Production Report

2021-02-04 11:49:31

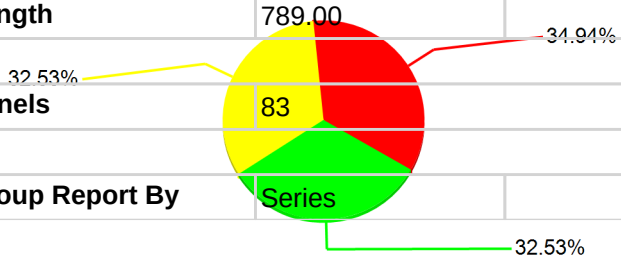
Production Line Name

From	2020-08-01 00:00:00	**To**	2021-02-04 11:49:00
Production Start	2020-09-03 16:10:24	**Production End**	2021-01-27 14:07:13
Product	TestProduct		
Series	3f708a10-9e80-4711-82f5-37fca6b3d869 e1482212-ec5f-468e-9abb-aa57629c8371 ec64cb0a-b489-4012-9f01-27cd984b8359 fdc340e5-f90e-4d64-8ddf-602827dd491d	**Shift**	TestShift
Pressplate Series: Total *Start Date : 2020-09-03 16:10:24*	BrightChristmas DarkWood ElefantMustard Fantasy FunnyWardrobe White	**Decor**	DarkWood Fantasy GrandmasDelight GreatBritanny ScisciMake White
Length	789.00	**Width**	647.00
Panels	83	**Down Grade Class**	Repair, Reject
Group Report By	Series	**Group Defect By**	Type

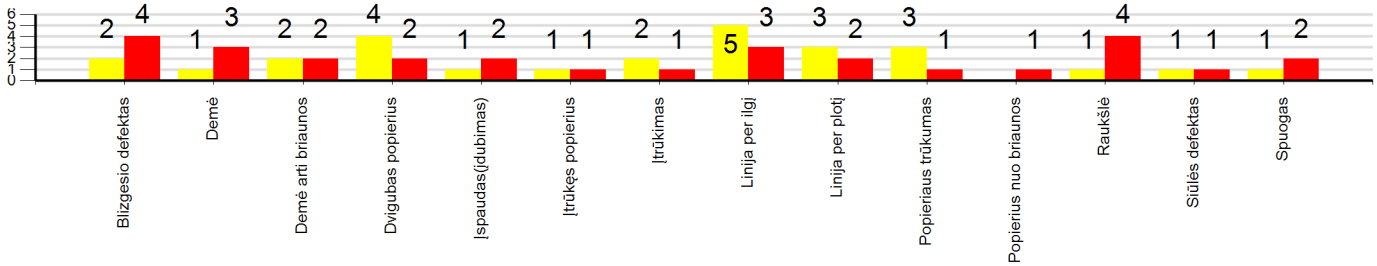
Thickness 18.00

Grades

	Panels	%	Linear m	Square m	Cubic m
Good	27	32.53%	21.30	13.78	0.25
Repair	27	32.53%	21.30	13.78	0.25
Reject	29	34.94%	22.88	14.80	0.27
Total	83	100.00 %	65.48	42.36	0.77



Downgrade by Defect Type



		Panels	%	Min #	Max #	Avg #	STDDev #	Linear m	Square m	Cubic m
Blizgesio defektas	Repair	2	2.41	1	1	1.00	0.00	1.58	1.02	0.02
	Reject	4	4.82	1	2	1.17	0.37	3.16	2.04	0.04
	Total	6	**7.23**	1	2	1.11	0.31	4.73	3.06	0.06
Demė	Repair	1	1.20	1	1	1.00	0.00	0.79	0.51	0.01
	Reject	3	3.61	1	1	1.00	0.00	2.37	1.53	0.03
	Total	4	**4.82**	1	1	1.00	0.00	3.16	2.04	0.04
Demė arti briaunos	Repair	2	2.41	1	1	1.00	0.00	1.58	1.02	0.02

Demė arti briaunos	Reject	2	2.41	1	1	1.00	0.00	1.58	1.02	0.02
	Total	4	**4.82**	1	1	1.00	0.00	3.16	2.04	0.04
Dvigubas popierius	Repair	4	4.82	1	2	1.17	0.37	3.16	2.04	0.04
	Reject	2	2.41	1	1	1.00	0.00	1.58	1.02	0.02
	Total	6	**7.23**	1	2	1.09	0.29	4.73	3.06	0.06
Įspaudas (įdubimas)	Repair	1	1.20	1	1	1.00	0.00	0.79	0.51	0.01
	Reject	2	2.41	1	1	1.00	0.00	1.58	1.02	0.02
	Total	3	**3.61**	1	1	1.00	0.00	2.37	1.53	0.03
Įtrūkęs popierius	Repair	1	1.20	1	1	1.00	0.00	0.79	0.51	0.01
	Reject	1	1.20	1	2	1.25	0.43	0.79	0.51	0.01
	Total	2	**2.41**	1	2	1.11	0.31	1.58	1.02	0.02
Įtrūkimas	Repair	2	2.41	1	1	1.00	0.00	1.58	1.02	0.02
	Reject	1	1.20	1	1	1.00	0.00	0.79	0.51	0.01
	Total	3	**3.61**	1	1	1.00	0.00	2.37	1.53	0.03
Linija per ilgį	Repair	5	6.02	1	2	1.11	0.31	3.95	2.55	0.05
	Reject	3	3.61	1	3	1.50	0.76	2.37	1.53	0.03
	Total	8	**9.64**	1	3	1.27	0.57	6.31	4.08	0.07
Linija per plotį	Repair	3	3.61	1	1	1.00	0.00	2.37	1.53	0.03
	Reject	2	2.41	1	2	1.25	0.43	1.58	1.02	0.02
	Total	5	**6.02**	1	2	1.11	0.31	3.95	2.55	0.05
Popieriaus trūkumas	Repair	3	3.61	1	1	1.00	0.00	2.37	1.53	0.03
	Reject	1	1.20	1	1	1.00	0.00	0.79	0.51	0.01
	Total	4	**4.82**	1	1	1.00	0.00	3.16	2.04	0.04
Popierius nuo briaunos	Repair	0	0.00	0	0	0.00	0.00	0.00	0.00	0.00
	Reject	1	1.20	1	1	1.00	0.00	0.79	0.51	0.01
	Total	1	**1.20**	1	1	1.00	0.00	0.79	0.51	0.01
Raukšlė	Repair	1	1.20	1	1	1.00	0.00	0.79	0.51	0.01
	Reject	4	4.82	1	1	1.00	0.00	3.16	2.04	0.04
	Total	5	**6.02**	1	1	1.00	0.00	3.95	2.55	0.05
Siūlės defektas	Repair	1	1.20	1	1	1.00	0.00	0.79	0.51	0.01
	Reject	1	1.20	1	1	1.00	0.00	0.79	0.51	0.01
	Total	2	**2.41**	1	1	1.00	0.00	1.58	1.02	0.02
Spuogas	Repair	1	1.20	1	1	1.00	0.00	0.79	0.51	0.01
	Reject	2	2.41	1	1	1.00	0.00	1.58	1.02	0.02
	Total	3	**3.61**	1	1	1.00	0.00	2.37	1.53	0.03

O Utdrag av gjenskapt produksjonsrapport

ARGOS

Production Report

30/04/2021 11:32:53

From: 2021-02-23 08:40:00 **To:** 2021-02-23 12:40:00

Production Start: 2021-02-23 08:40:09 **Production End:** 2021-02-23 12:39:56

Product: Class B

Series: 2021023-2
2021023-3

Shift: 1

Length: 1270.00000 **Width:** 2540.00000

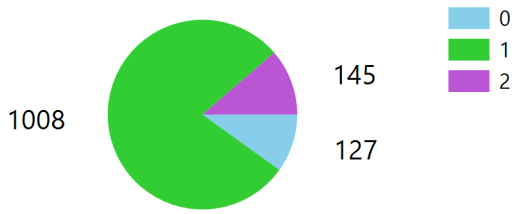
Thickness: 13.41100

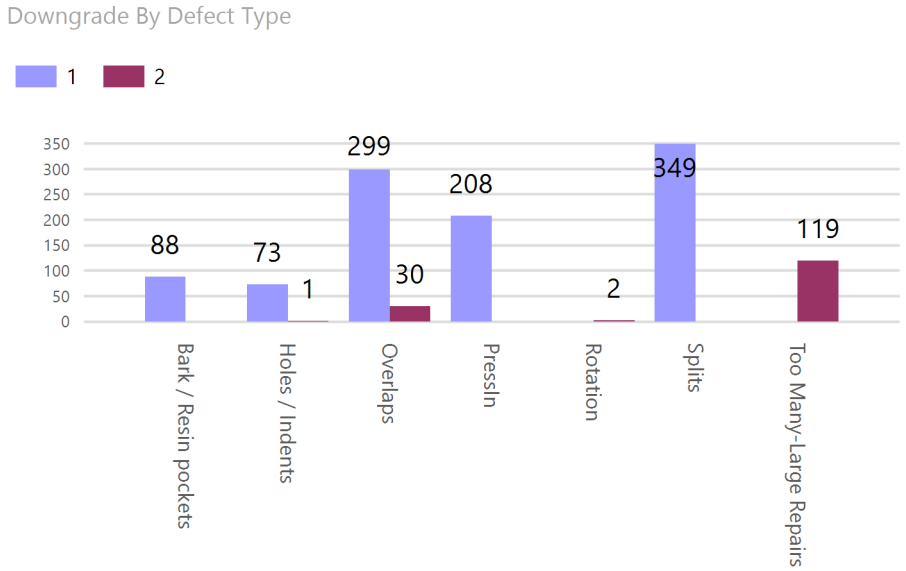
Panels: 1305

Downgrade class: Repair
Reject
No repair

Group Report By: Series

Class	Count	%
0	127	9.92%
1	1008	78.75%
2	145	11.33%
Total	1280	





ARGOS

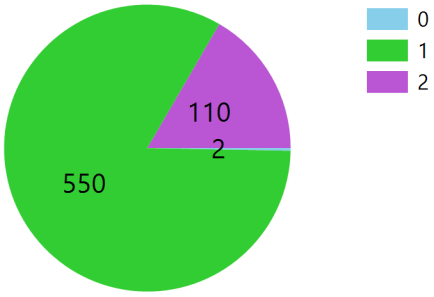
Production Report

30/04/2021 11:32:53

Too Many-Large Repairs	**2**	119
Rotation	**0**	136
	2	2
PressIn	**1**	208
Holes / Indents	**1**	73
	2	1
Overlaps	**1**	299
	2	30
Splits	**1**	349
Bark / Resin pockets	**1**	88

Series: 2021023-2

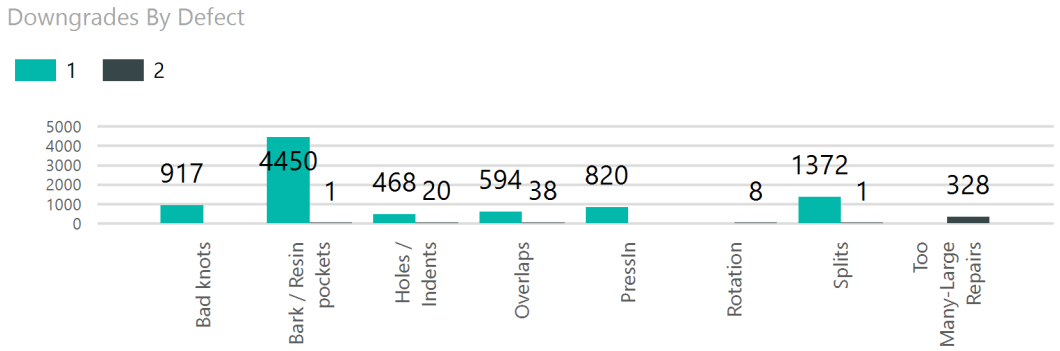
Class	Count	%
0	2	0.30%
1	550	83.08%
2	110	16.62%
Total	662	



ARGOS

Production Report

30/04/2021 11:32:53



Too Many-Large Repairs	**2**	328
Rotation	**0**	259
	2	8
PressIn	**1**	820
Holes / Indents	**1**	468
	2	20
Bad knots	**1**	917
Bark / Resin pockets	**1**	4450
	2	1
Overlaps	**1**	594
	2	38
Splits	**1**	1372
	2	1

P Utdrag av ny produksjonsrapport

Production Report

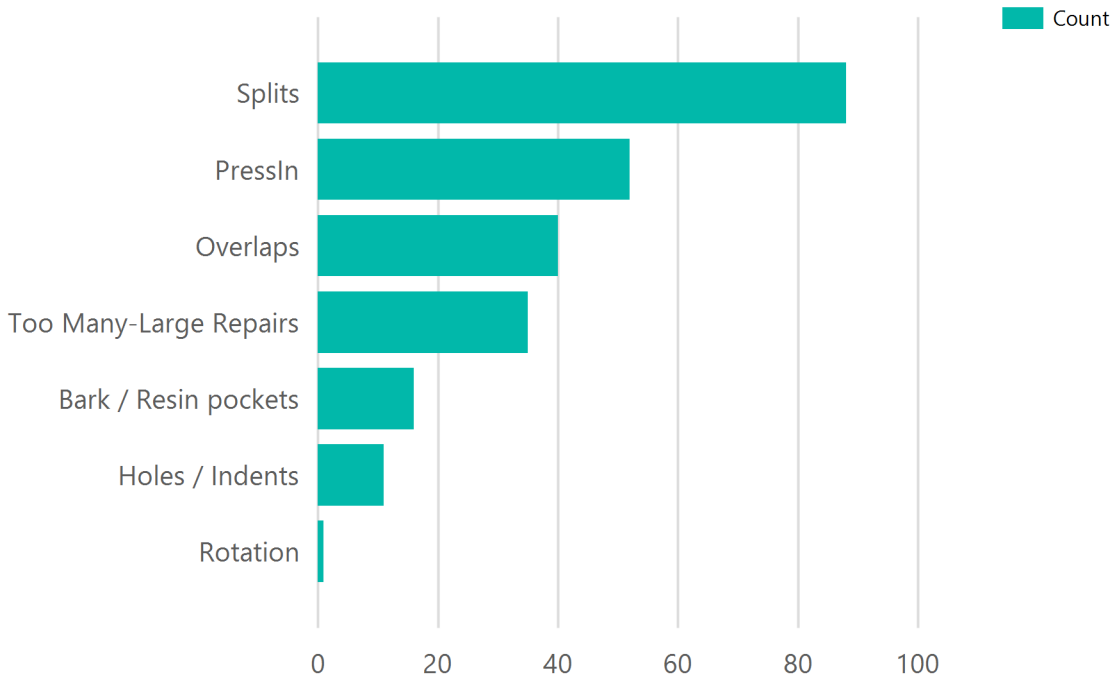
DESKTOP-MD2ULOV\andre | 08/04/2021 09:01:56

From: 2021-02-23 08:40:00
To: 2021-02-23 09:40:00
Product: Class B

Production Start: 2021-02-23 08:40:09
Production End: 2021-02-23 09:39:53
Series: 2021023-2

Class	Count
1	202
2	38
Total	**240**

Downgrades by Defect Type



Q Kodeeksempel: xml_get_downgrades

```
@plugmanager.register_plugin("Downgrades", category="report_builder")
def xml_get_downgrades(
    session: Session,
    _,
    start_time: datetime,
    end_time: datetime,
    __,
    extra_data: Dict[str, Union[str, int]],
) -> Iterable[ET.Element]:
    """Gets downgrade type, corresponding count and initial class

    :param session:     Current database session.
    :param start_time:  Start of query time limit.
    :param end_time:    End of query time limit.
    :param extra_data:  Any extra data, in this case sorting parameter.

    :return:            Iterable of XML Elements containing query data
    """

    Boards = DatabaseConnections.tables["Boards"]
    Defects = DatabaseConnections.tables["Defects"]
    column_names = ["DefectType", "Count", "Class"]

    data = (
        session.query(Defects.c.Type, func.count(Boards.c.ID), Defects.c.Class)
        .join(Defects)
        .filter(Boards.c.Time.between(start_time, end_time))
        .filter(Defects.c.IsWorstDefect == "1")
    )

    # If group-by column is included in call, filter data by it
    if "gbcolumn_name" in extra_data:
        data = data.filter(
            getattr(Boards.c, extra_data["gbcolumn_name"])
            == extra_data["gbcolumn_value"]
        )

    data = data.group_by(Defects.c.Type, Defects.c.Class)

    return xml_converter(data.all(), column_names)
```

R Pytest fixture oppsett

```
def _get_injected_api_app(injected_db_callable):
    app = APIApp(
        load_from_settings={"reflect": False, "debug": False, "use_database": False}
    )
    app.dependency_overrides[global_dependencies["get_db"]] = injected_db_callable
    return TestClient(app)

@pytest.fixture(name="memory_database")
def create_in_memory_database():
    tmp_file = tempfile.NamedTemporaryFile(suffix=".db").name
    engine = create_engine(f"sqlite:///{tmp_file}")
    smaker = sessionmaker(bind=engine)

    database = DatabaseConnections({})
    database.connections = {"memory": {"engine": engine, "sessionmaker": smaker}}

    Base = declarative_base()

    class MockedBoards(Base):
        __tablename__ = "Boards"
        ID = Column(Integer, primary_key=True)
        BoardID = Column(Integer)
        ClassName = Column(String)
        Class = Column(Integer)
        Time = Column(DateTime)

        def __repr__(self):
            return f"MockedBoards(ID={self.ID})"

    class MockedDefects(Base):
        __tablename__ = "Defects"
        ID = Column(Integer, primary_key=True)
        Type = Column(String)
        SubType = Column(String)
        ArgosName = Column(String)

        def __repr__(self):
            return f"MockedDefects(ID={self.ID})"

    DatabaseConnections.tables = {}
    DatabaseConnections.tables["Boards"] = MockedBoards.__table__
    DatabaseConnections.tables["Defects"] = MockedDefects.__table__

    Base.metadata.create_all(engine)

    def override_get_db_memory(request: Request):
        request.state.db = database

    return database, _get_injected_api_app(override_get_db_memory)

@pytest.fixture(name="mocked_app_boards_per_class")
```

```
52 def populate_data(memory_database):
53     Boards = DatabaseConnections.tables["Boards"]
54     database, app = memory_database
55 
56     for _, session in database.sessioniter("memory"):
57         session.execute(
58             Boards.insert(),
59             [
60                 dict(
61                     BoardID=1,
62                     Class=0,
63                     Time=datetime.fromisoformat("2021-01-01
                           01:00:00"),
64                     ClassName="No Repair",
65                 ),
66                 dict(
67                     BoardID=2,
68                     Class=1,
69                     Time=datetime.fromisoformat("2021-01-01
                           02:00:00"),
70                     ClassName="Repair",
71                 ),
72             ],
73         )
74 
75         session.commit()
76 
77     return app
78 
79 
80 def test_get_boards_per_class(mocked_app_boards_per_class):
81 
82     result = mocked_app_boards_per_class.post(
83         "/boards_by_class",
84         json=dict(
85             start_time="2021-01-01 00:00:00",
86             end_time="2021-03-01 00:00:00",
87             scanners=["memory"],
88         ),
89     ).json()
90 
91     assert result == {"0": [1, 50], "1": [1, 50]}
```

S Tidsbruk

S.1 Timelogger

Tidsbruk, felles

Dato	Beskrivelse	ADM	LITT	PROG	RAP	MØTE
12/01/2021	Internt oppstartsmøte, planlegging og veiledning med Tom	2				1
13/01/2021	Definere grupperegler	2				
14/01/2021	Lynkurs 1: Prosjektstyring					2
14/01/2021	Forberedelse til oppstartsmøte med Argos	2				
15/01/2021	Oppstartsmøte med Argos					2
16/01/2021	Diskutert møte 15.01 og planlagt videre					2
19/01/2021	Jobbing med forprosjekt		1		5	
20/01/2021	Ferdigstilte forprosjekt første utkast og sendt				2	
21/01/2021	Møte med Argos om utkast til					1
21/01/2021	Veiledningsmøte med Tom og videre					1
23/01/2021	Rapportskriving og videre planlegging				2	
26/01/2021	Rapportskriving og videre planlegging	0.5			7	
27/01/2021	Rapportskriving og videre planlegging				4.5	
28/01/2021	Møter og administrativt arbeid	2				2
29/01/2021	Møte med norSIS og diskusjon					1
30/01/2021	Ferdigstille og levere forprosjekt				2.5	
02/02/2021	Planlegging	2				
03/02/2021	Repo + Undersøkelse av sky		5.5			
04/02/2021	Møte med Argos + planlegging	3			0.5	
06/02/2021	Oppsett til Git og sky + diskuterte møte med	2				
09/02/2021	Adm + møte med Tom	1	0.5			1
13/02/2021	Videre planlegging	1				
16/02/2021	Forts. skyundersøkelse				7	
17/02/2021	Forts. skyundersøkelse				8.5	1
18/02/2021	Møte med Argos og ferdigstilling av	4				1
20/02/2021	Retrospektiv og planlegging					2.5
23/02/2021	Planlegging og møte	5	1			1
24/02/2021	Planlegging, møte og jobbing med prototype	1		4		1
25/02/2021	Jobbing med prototype			6		0.5
27/02/2021	Planleggingsmøte					1.5
02/03/2021	Forts. prototype			7.5		
03/03/2021	Forts. prototype	1		5		0.5
04/03/2021	Forts. prototype og møte med Argos			4		1
05/03/2021	Planlegging	0.5				
06/03/2021	Planlegging av neste sprint	2.5				
09/03/2021	Diskuterte spørreundersøkelse og planla	3.5				
10/03/2021	Møte med Tom + planlegging videre	4.5				1
10/03/2021	Møte med salgsavdelingen					1.5
11/03/2021	Oppsummert møte og videre planlegging	6.5				
13/03/2021	Planlegging av oppløp til møte med Argos					1
16/03/2021	Spontanmøte med Tom og videre planlegging/jobbing	2		5		0.5
17/03/2021	Videre jobbing og møte med Tom	0.5		4.5		0.5
18/03/2021	Videre jobbing og møte med Argos			6		1
20/03/2021	Retrospektiv og planlegging	2				
23/03/2021	Videre jobbing			2		

24/03/2021	Møte med Tom					0.5
25/03/2021	Videre jobbing			7		
03/04/2021	Statusmøte innad i gruppa					1
08/04/2021	Møte med Argos, møte med gruppa, møte med					2
10/04/2021	Retrospektiv og planlegging					2.5
17/04/2021	Statusmøte innad i gruppa					0.5
21/04/2021	Møte					1
24/04/2021	Retrospektiv og planlegging					1

Tidsbruk, Andreas

Dato	Beskrivelse	ADM	LITT	PROG	RAP	MØTE
13/01/2021	Grupperegler og research	2				
14/01/2021	Forberedelse til oppstartsmøte med ARGOS	1				
18/01/2021	Forprosjekt og lesing av tidligere oppgaver		1		3	
19/01/2021	Forprosjekt og lesing av tidligere oppgaver				1	
22/01/2021	Forprosjekt forts., tidsstyring og adm.	1			2	
22/01/2021	Sendte forespørsel til NorSIS om møte ang. sky	0				
25/01/2021	Risikoanalyse i forprosjektsrapporten				4	
29/01/2021	Forprosjekt forts. og finpuss + Grov skisse av arkitektur		1		3	
02/02/2021	Oppsett av testmiljø med Lars og Ole.			4.5		
04/02/2021	Lese fagstoff knyttet til sikkerhet i sky		0.5			
05/02/2021	Skyundersøkelse: Tilegne fagstoff		1			
05/02/2021	Møte med Hjelmås og diskusjon etterpå	0.5				0.5
05/02/2021	Satt opp mal for undersøkelse med Mads				1	
05/02/2021	Tilegnet kunnskap om docker		0.5			
08/02/2021	Planlegging til møter og start skyundersøkelse	1	1		1	
08/02/2021	Forts. skyundersøkelse		0.5		0.5	
08/02/2021	Tilegnet kunnskap om docker/grafana/prometheus		0.5			
09/02/2021	Forts. skyundersøkelse				4	
10/02/2021	Forts. skyundersøkelse				7	
11/02/2021	Forts. skyundersøkelse				6	
12/02/2021	Forts. skyundersøkelse				4.5	
15/02/2021	Forts. skyundersøkelse				6	
22/02/2021	Planlegging av neste sprint	1	4			1
27/02/2021	Planlegging prototype	0.5				
01/03/2021	Forts. prototype		0.5	2.5		
05/03/2021	Møte med Mads og Eivind					1
08/03/2021	Forberedelse til intervju med salgsavdeling	5				
09/03/2021	Snakke med Argos om simulering					0.5
09/03/2021	Diskutert simulering og skrevet mer til b.rapport	1			1	
15/03/2021	Avgrensning av rapport-epic		2.5			2
17/03/2021	Jobbing med rapport-muligheter			2		
20/03/2021	Rapport PoC			1.5		
22/03/2021	Rapport PoC			3.5		
23/03/2021	Planlegging og videre jobbing PoC			3.5		
24/03/2021	Videre jobbing PoC og møte med Tom			5.5		
25/03/2021	Møte med Eivind					0.5
26/03/2021	Videre jobbing med PoC			2.5		
29/03/2021	Videre jobbing med PoC			4.5		
30/03/2021	Videre jobbing med PoC			3.5		
31/03/2021	PoC: Utforming av eksisterende rapport i RB			10		
01/04/2021	Mads PR, prep til skriving	1				
02/04/2021	Skriveprep, issues og startet på introduksjon	1	0.5		4.5	
05/04/2021	BR: Organiseringskapittel og funnet flere kilder til insp		0.5		8.5	
06/04/2021	BR: Hvorfor oppgaven, målgrupper				4	
07/04/2021	BR: Målgrupper forts., prep til møte med Argos			6.5	1	0.5

08/04/2021	Prep til møte med Argos, lese tbo, møte med Ole og Mads	0.5	0.5			1
09/04/2021	BR: prosjektbeskrivelse, info om argos/virke		0.5		5	
10/04/2021	BR: Prep introduksjon prototype	0.5				
12/04/2021	Møteinnkalling + BR: Gruppemedlemmer, argos/virke	0.5			3	1
13/04/2021	Gå gjennom PR (BR), hjalp Mads, BR: introduksjon første				7.5	
14/04/2021	BR: Finpuss introduksjon, start utviklingsprosess				6.5	
15/04/2021	BR: Utviklingsprosess, Planlegging/diskusjon: fremdrift				7.5	
16/04/2021	BR: Utviklingsprosess knot				0.5	
19/04/2021	BR: Gjennomføring av modell				6.5	
20/04/2021	BR: Utviklingsprosess				8	
21/04/2021	BR: Kravspek og teknologier				7.5	
22/04/2021	BR: Kravspek og teknologier				7	
23/04/2021	BR: Ferdigstille kravspek og start teknologier				7.5	
26/04/2021	PR, kodefletting og proofreading			5	2.5	
27/04/2021	Finpuss, proofreading og PR				9.5	
28/04/2021	Forts. teknologier, møte med Tom				4	1.5
29/04/2021	BR: teknologier, start design				7.5	
30/04/2021	BR: start implementasjon, planlegging, møte med Argos	0.5			5.5	0.5
01/05/2021	BR: implementasjon forts.				5.5	
03/05/2021	BR: implementasjon, ordliste, akronym og puss				11.5	
04/05/2021	BR: utviklingsprosess revisit og planlegging	1			5	
05/05/2021	BR: implementasjon rapp. + revidering utvp.				9	1
06/05/2021	BR: Hjelpe Mads, Proofreading og ferdigstille imp.				10.5	
07/05/2021	BR: Proofreading, utviklingsprosess ferdigstilt, start drøft				10.5	
08/05/2021	BR: Drøfting start, tidsbruk i utv.				6	
10/05/2021	BR: Drøfting (prosess) og proofreading				13.5	
11/05/2021	Proofreading, implementasjonspuss				6	
12/05/2021	Proofreading, møte med Tom, imp. rapport				10	1
13/05/2021	BR: proofreading, drøfting				7.5	
14/05/2021	BR: proofreading				2	
15/05/2021	BR: drøfting, annet				7.5	
16/05/2021	Proofreading og finpuss av BR				7.5	
17/05/2021	Proofreading og finpuss av BR				8	
18/05/2021	Proofreading og finpuss av BR				8.5	
19/05/2021	Proofreading og finpuss av BR				8	
20/05/2021	Proofreading og finpuss av BR				3	

Tidsbruk, Lars

Dato	Beskrivelse	ADM	LITT	PROG	RAP	MØTE
22/12/2020	Sendt møteinnkalling til Argos	0				
13/01/2021	Sendt epost til Erik om Skyhigh	0	1			
13/01/2021	Fått Skyhigh instans	0				
13/01/2021	Lest gjennom et utvalg av tidligere bachelorrapporters forprosjektrapporter for å de hvordan de har blitt formulert.		1			
19/01/2021	Skrevet på forprosjektrapport				2	
21/01/2021	Lest gjennom tidligere bachelorrapporter.		1			
21/01/2021	Sett på verktøy for modeller	1				
21/01/2021	Sett på arkitekturmodeller.		1			
21/01/2021	Eksperimentere litt med Flask/API rammeverk.			2		
21/01/2021	Startet konkretisering av use case og ikke funksjonelle krav		1			
24/01/2021	Lest gjennom scrumban dokument		1			
24/01/2021	Utdypet om bruk av utviklingsmodell i rapport				2	
24/01/2021	Raskt testet prototyper av potensielle rammeverk			1		
28/01/2021	Testet simuleringsprogram fra Argos, oppsett av Skyhigh runner	1				
28/01/2021	Grov skisse av arkitektur med Andreas		1			
02/02/2021	Oppsett av testmiljø med Andreas og Ole			4.5		
05/02/2021	Research om arkitektur og design		1			
05/02/2021	Startet prototype i Python			1.5		
05/02/2021	Satt opp en simuleringsscanner til			2		
07/02/2021	Prototyping i Python			2		
09/02/2021	Prototyping i Python			4		
10/02/2021	Prototyping i Python			7		
11/02/2021	Prototyping i Python			7		
12/02/2021	Prototyping i Python			2		
12/02/2021	Prototyping i Python			4		
25/02/2021	Import av relistisk datasett			2		
26/02/2021	Snakke om fremdrift	1				
27/02/2021	Begynne å kode simulering			2		
28/02/2021	90% klar prototype av simularing			2		
06/03/2021	Diskusjon av simuleringsprototype		1			
07/03/2021	Se på forbedring av plugin arkitekturen			3		
09/03/2021	Snakke med Argos om simulering			0.5		
09/03/2021	Diskutert simulering og skrevet mer til b.rapport	1			1	
14/03/2021	Installere report server, api rammeverk forbedring og gitlab CI.			7		
17/03/2021	Backend og frontend av simuleringsrammeverk			4		
23/03/2021	Utforske python XML bibliotek			2		
23/03/2021	Diskutere microsoft report builder		2			
24/03/2021	Diskutere og progge microsoft report builder		6.5			
25/03/2021	Jobbe med XML endepunkt (utenom fellesen)			2.5		
25/03/2021	Justering av simulering hvis plater har mer enn 1 defekttype			2		
30/03/2021	Bistå Andreas med report builder			3.5		

30/03/2021	Justering av simulering hvis plater har mer enn 1 defekttype			3.5		
31/03/2021	Bistå Andreas med report builder			10.5		
03/04/2021	Ignorere plater med ellers vrak			4		
04/04/2021	Ignorere plater med ellers vrak			2		
05/04/2021	Ignored plater med ellers vrak			3		
05/04/2021	BR: Om rapporten				2	
06/04/2021	Velge SubType istedenfor ArgosName i frontend			5		
06/04/2021	/xml/query extend istedenfor append			1		
06/04/2021	Bistå andre med innspill og debugging			1		
07/04/2021	Rydde i utsatte oppgaver, tester, ci			4		
07/04/2021	Bistå andreas med report builder			2		
07/04/2021	BR: Start utkast om analyse			5		
08/04/2021	Pynte på analyse frontend			3		
08/04/2021	BR: Sette opp b.rapport utviklingsmiljø				1	
13/04/2021	simulering: n-delt klassifisering			7.5		
14/04/2021	n-delt klassifisering			7.5		
15/04/2021	n-delt, forbedret algoritme for forslag, diskutere status			7.5		
16/04/2021	Bistå med dashboard spørring om plater.			1		
17/04/2021	Legge til request og responsemodeller i FastAPI			2		
18/04/2021	Legge til request og responsemodeller i FastAPI			3		
19/04/2021	BR: Teknologier/Design + Utviklingsprosess				6.5	
20/04/2021	BR: Teknologier/Design + Utviklingsprosess				8	
21/04/2021	Manuell simulering			9.5		
22/04/2021	Skrive om simulering frontend til Vue			8.5		
23/04/2021	Hjelpe Mads med optimalisering + Organisere python-kode			7.5		
26/04/2021	Skrive om design og implementasjon				7	
27/04/2021	Renskrive kravspek og teknolgier				9.5	
28/04/2021	Utrulling, finpusse kode				6.5	1.5
29/04/2021	CI/CD			7.5		
30/04/2021	Sphinx og sikkerhet			7.5		
01/05/2021	Design og implementasjon rammeverk			9.5		
03/05/2021	Implementasjon, utrulling og testing				13	
04/05/2021	Testing	1		6		
05/05/2021	Analyse design og implementasjon				12	
06/05/2021	Implememementasjon analyse og proofreading				10.5	
07/05/2021	Design rammeverk og proofreading				11	
08/05/2021	Design og implementasjon rammeverk og proofreading				9	
10/05/2021	Proofreading, diskusjon drøft og teknisk drøft.				13.5	
11/05/2021	Proofreading, implementasjonspuss				6	
12/05/2021	Proofreading, møte med Tom, sikkerhet				10.5	1
13/05/2021	Testing, proofreading				7.5	
14/05/2021	Gjennomlesning av andres presentasjoner og design dashbord				11.5	
16/05/2021	Installasjon, konklusjon og proofreading				7.5	
17/05/2021	Proofreading og finpuss av BR				8	
18/05/2021	Proofreading og finpuss av BR				8.5	

19/05/2021	Proofreading og finpuss av BR				8	
20/05/2021	Proofreading og finpuss av BR				3	

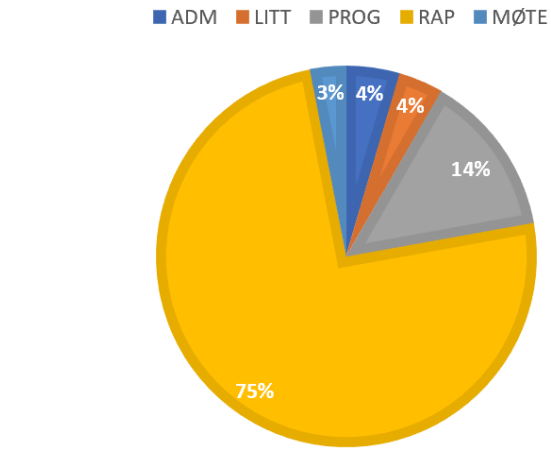
Tidsbruk, Mads

Dato	Beskrivelse	ADM	LITT	PROG	RAP	MØTE
13/01/2021	Forberedelse til møte med oppdragsgiver		1			
18/01/2021	Forprosjekt og lesing av tidligere oppgaver		1		3	
22/01/2021	Forprosjekt og videre planlegging	1			2	
25/01/2021	Forprosjekt og lesing av tidligere oppgaver		1		3	
29/01/2021	Forprosjekt forts. og finpuss				3	
02/02/2021	Planlegging + forarbeid vurdering skyløsning	2	3.5			
04/02/2021	Lese om skytjenester		1			
05/02/2021	Møte med Hjelmås + diskusjon og sette opp struktur til skyløsning	0.5				1.5
08/02/2021	Fordele oppgaver til vurdering + skrive vurdering				3	1
09/02/2021	Skrive vurdering av skyløsning				4	
10/02/2021	Forts. skyundersøkelse				7	
11/02/2021	Forts. skyundersøkelse				6	
12/02/2021	Forts. skyundersøkelse				4	
15/02/2021	Forts. skyundersøkelse				6	
22/02/2021	Planlegging av neste sprint	2	2			0.5
26/02/2021	Jobbe med prototype			4		
01/03/2021	Jobbe med prototype		0.5	2.5		
08/03/2021	Forberedelse til intervju	2.5				
15/03/2021	Jobbe med prototype 2			3.5		
22/03/2021	Jobbe med Dashboard og videre planlegging	0.5		3.5		
23/03/2021	Planlegging og videre jobbing			3.5		
24/03/2021	Heatmap og Produktinformasjon i Dashboard			6		
25/03/2021	Møte med Eivind					0.5
29/03/2021	Spørringer til rapport + refactor i boards.py			4		
05/04/2021	Dashboard + Rapport			3	0.5	
06/04/2021	Manual for oppsett av dashboard til utvikling + Dashboard		2	2		
07/04/2021	Heatmap og mal til Dashboard			4		
08/04/2021	Brukerteste manual for oppsett av dashboard					
09/04/2021	Revidere manual etter brukertest + rapport		1.5		1.5	
12/04/2021	Dashboard + Møte med Argos			5.5		1
13/04/2021	Dashboard			3.5		
14/04/2021	Dashboard			5.5		
16/04/2021	Manual + Dashboard		2	3		
19/04/2021	Kravspesifikasjon dashboard				5	
21/04/2021	Dashboard			7.5		
22/04/2021	Heatmap			7.5		
23/04/2021	Opprydding i dashboard-kode + kravspek			2.5	5	
26/04/2021	Kravspesifikasjon + Teknologi dashboard				7.5	
27/04/2021	Kravspesifikasjon + Teknologi dashboard + Gjennomgang av Pull Requests				9	
28/04/2021	Design dashboard + møte med Tom				6.5	1.5
29/04/2021	Design dashboard				4	
30/04/2021	Design + implementasjon dashboard				4	
01/05/2021	Implementasjon				5	
03/05/2021	Implementasjon				11	
04/05/2021	Design				5	

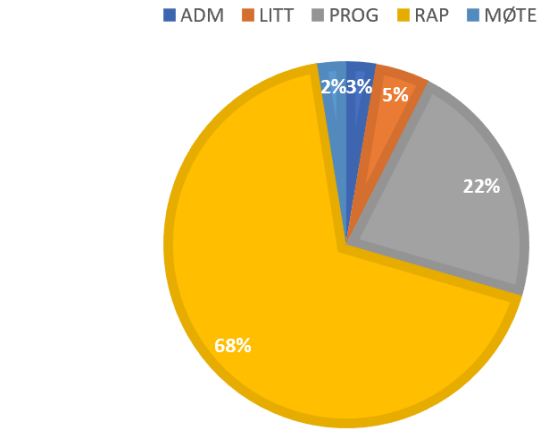
05/05/2021	Implementasjon + møte				8	1
06/05/2021	Implementasjon				9	
07/05/2021	Implementasjon				9	
08/05/2021	Implementasjon				8	
10/05/2021	Implementasjon + drøfting				10	
11/05/2021	Design dashboard + PR(implementasjon rammeverk)				8	
12/05/2021	Design, implementasjon, PR og møte				9	1
13/05/2021	Design + Revisjon + PR				7.5	
14/05/2021	Feedback + Design + Drøfting				11.5	
16/05/2021	Design				6.5	
17/05/2021	Rapport				8	
18/05/2021	Rapport				8.5	
19/05/2021	Rapport				8	
20/05/2021	Rapport				3	

S.2 Tidsbruk per kategori

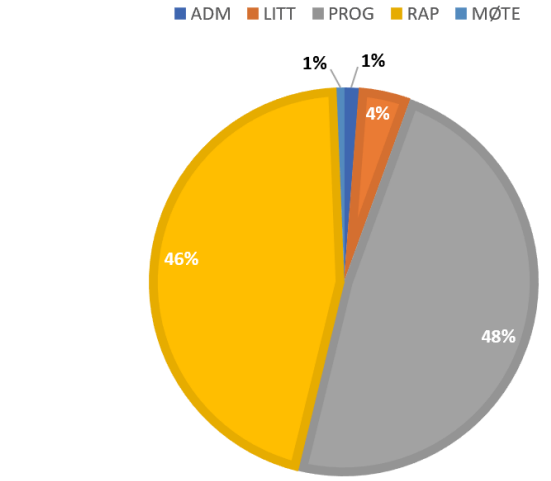
TIDSBRUK PER KATEGORI, ANDREAS

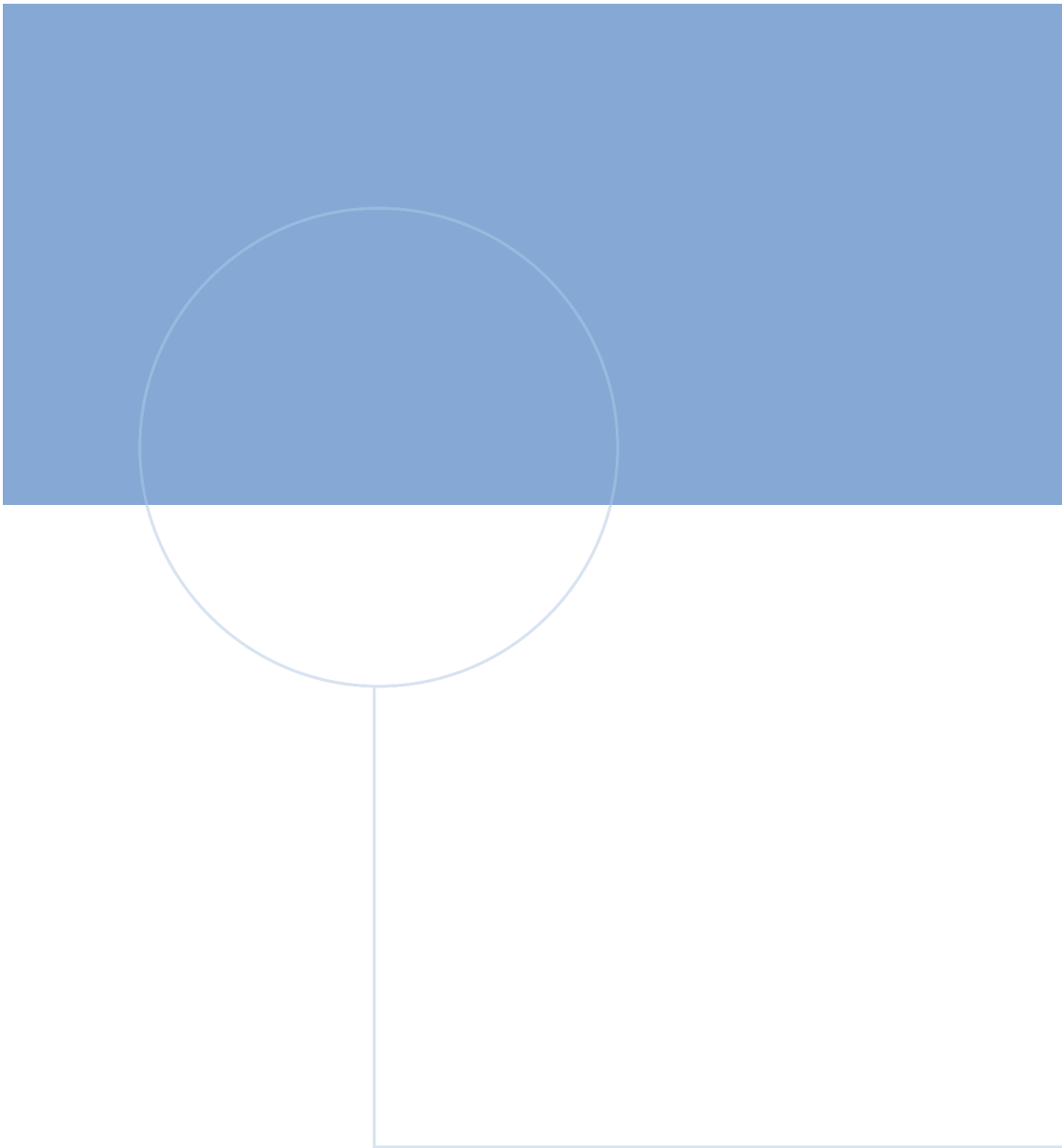


TIDSBRUK PER KATEGORI, MADS



TIDSBRUK PER KATEGORI, LARS





NTNU

Kunnskap for en bedre verden