Høgskulen på Vestlandet

BACHELOROPPGAVE

Kundens Profil - Tredjeparts-Integrasjon

Customer Profile - Third-Party-Integration

Audun Stien,

Christopher Nguyen,

Knut Anders Aabø Opstad og

Peter Liessem

Fakultet for ingeniør- og naturvitenskap (FIN)

Studie: dataingeniør/informasjonsteknologi

Veiledere: Bjarte Kileng og Kjetil Salomonsen

Vi bekrefter at arbeidet er selvstendig utarbeidet, og at referanser/kildehenvisninger til alle kilder som er brukt i arbeidet er oppgitt, jf. Forskrift om studium og eksamen ved Høgskulen på Vestlandet, § 10.

TITTELSIDE FOR HOVEDPROSJEKT

Rapportens tittel: Kundens Profil - tredjeparts-integrasjon	*Dato:* *03.06.21*
Forfatter(e): Audun Stien, Christopher Nguyen, Knut Anders Aabø Opstad og Peter Liessem	*Antall sider u/vedlegg:* *31* *Antall sider vedlegg:* *4*
Studieretning: Informasjonsteknologi/Dataingeniør	*Antall disketter/CD-er:* 0
Kontaktperson ved studieretning: Bjarte Kileng	*Gradering:* Ingen
Merknader: Prosjektet tok plass under COVID-19 pandemien.	

Oppdragsgiver: Stacc Insight AS	*Oppdragsgivers referanse:*
Oppdragsgivers kontaktperson: Kjetil Salomonsen	*Telefon:* 416 38 642

Sammendrag:

I denne rapporten beskriver gruppen Kundens Profil, ressursbruken fra en gammel til en ny løsning og prosessen med å videreutvikle denne tjenesten. Tjenesten videreutvikles med å integrere registre inn i løsningen. Registrene som brukes blir implementert i form av en Web API løsning. Løsningen vil også jobbes med å gjøre skybasert.

Stikkord:

Web API	ASP.NET	Stacc Insight

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

Forord

Denne rapporten dokumenterer bacheloroppgaven “Kundens Profil - tredjeparts integrasjon”. Oppgaven er skrevet av: Audun Stien, Christopher Nguyen, Knut Anders Aabø Opstad og Peter Liessem.

Vi vil takke Stacc Insight for selve ideen og hjelp gjennom prosjektet. Oppgaven ville ikke vært mulig om de ikke gav tilbudet.

Vi vil også rette en stor takk til Bjarte Kileng og Kjetil Salomonsen. Bjarte har vært vår interne veileder, han har gitt oss råd underveis og svart på spørsmål om oppgaven. Kjetil har vært vår eksterne veileder, han har stått på for å få ting til å gå opp og pushet selskaper for at vi kunne bruke deres APIer. Det hadde vært ennå vanskeligere å utført oppgaven om det ikke var for veilederne våre.

Innholdsfortegnelse

Table of Contents

1 Innledning

1.1 Motivasjon og mål

Stacc Insight leverer i dag en rådgivningstjeneste til banker kalt Kundens Profil. Hovedfunksjonen til Kundens Profil er å gi rådgivere hos bankene en god oversikt over privatpersoners økonomiske situasjon. Ved å bruke en kombinasjon av data skrevet inn av privatpersonen selv, og informasjon hentet fra diverse registre, skal bankene kunne nøyaktig estimere blant annet hvor mye en person skal kunne få i lån.

Figur 1: Stacc sin logo

Insight har et ønske om å benytte seg av teknologi med lavere eksekveringstid og mindre ressursbruk i forhold til dagens løsninger som kjører i en eldre løsning med navn Apache Tomcat. Prosjektet gikk ut på å kunne utvikle en oversiktlig og veldokumentert tjeneste for å hente ut relevant informasjon fra fire registre. Denne tjenesten skulle da senere bli implementert inn i Kundens Profil av Stacc Insight.

De fire registrene gruppen skulle hente informasjon fra var: Kjøretøyregisteret, Skipsregisteret, Rettssjekk og Løsøreregisteret. Stacc Insight informerte gruppen om hvilke informasjon som var relevant å hente ut fra alle registrene utenom Rettssjekk, så gruppen vurderte selv hva som var relevant for Rettssjekk.

Registrene ble implementert i form av en Web API løsning. Dette er for at Kundens Profil skal kunne sende en forespørsel til gruppens API-løsninger om å hente informasjon fra et register. Dataene inneholder kun informasjon som er relevant for bankene. Kundens Profil bruker informasjonen til å fylle ut skjemaer som ellers måtte ha blitt fylt inn for hånd. Dette sparer banken for tid og ressurser.

Minimumskravene for prosjektet var å kunne lage og dokumentere en tjeneste som henter og leverer data fra alle registrene. Hvis gruppen fullførte minimumskravene tidlig nok kunne de utføre et eller flere delmål:

- Proof of Concept på bruk av flere forskjellige teknologier i skyløsning.
- Styrke autentiseringen for å kunne eksponere APIene.
- Integrere flere registre i løsningen.

1.2 Kontekst

For at en bank skal kunne estimere hvor mye en kunde kan låne, er det hjelpsomt å ha en god oversikt over eiendelene til kundene fra troverdige kilder. Denne informasjonen ligger spredt utøver forskjellige registre, og bankens rådgivere må gå manuelt inn på registrene for å dobbeltsjekke om informasjonen kunden gir stemmer overens. Gruppen har jobbet sammen med prosjekteier, Stacc Insight, for å lage en løsning som skal gi bankene tilgang til brukerens relevante eiendeler på ett sted. Dette vil gruppen få til ved å implementere forskjellige tredjeparts-registre. Denne løsningen skal føre til at bankene får mer sikkerhet, ettersom den relevante infoen alltid blir dobbeltsjekket. Det vil også spare tid for ansatte siden de vil da slippe å sjekke registrene manuelt og skrive inn dataen.

Mye av dataen som kommer fra registrene er overflødig eller i feil format, for eksempel årstall i stedet for alder. På grunn av dette må det avklares hvilke data som er relevant slik at den kan prosesseres på en hensiktsmessig og oversiktlig måte.

1.3 Avgrensninger

Gruppen valgte å ikke starte med kodingen av løsningen før midt under prosjektprosessen, ettersom det tok tid for gruppen å få tilgang til registrene. For å unngå å kaste bort tid, brukte gruppen startperioden til å lære seg .NET, se på forskjellige løsninger og jobbe med akademisk skriving.

Ingen i gruppen var godt kjent med .NET, så det tok en stund før alle medlemmene følte seg komfortable med rammeverket. Mye av tiden ble også planlagt til akademisk skriving, ettersom det heller ikke var noe de har jobbet mye med.

Denne bacheloroppgaven har blitt skrevet mens det var Covid-19 restriksjoner i Bergen. Situasjonen hadde en påvirkning på flyten til arbeidet, ettersom noen gruppemedlemmer, gjennom prosjektperioden, endte opp enten syke eller måtte i karantene. Siden prosjektet gikk ut på å utvikle en programvare var det mulig å arbeide uten fysiske møter. Dette har gjort at Covid-19 ikke hadde en veldig stor innvirkning på prosjektet som helhet.

1.4 Ressurser

Den største ressursen gruppen brukte var ASP.NET rammeverket og dokumentasjonen som fulgte med disse og brukt til å konstruere applikasjonen. Gruppen brukte også dokumentasjonen til registrene for å finne ut hvordan de kunne bli hentet, og en IDE (Integrated Development Environment) til å kunne programmere prosjektet i. Ekstern og intern veileder var gode ressurser, ettersom ekstern veileder kunne mye om .NET og intern veileder hadde mye erfaring innen akademisk skriving og tidligere bacheloroppgaver.

1.5 Organisering av rapporten

Kapittel 1: Introduksjon	Presentasjon av teamets mål og motivasjon til oppgaven, konteksten og alt relevant for Stacc Insight. Det blir presentert gruppens begrensninger i forhold til å utføre prosjektet. Til slutt blir det presentert de ressursene gruppen tar tenkt å bruke.
Kapittel 2: Prosjektbeskrivelse	Presentasjon av bakgrunnen til problemstillingen, og videre blir det en presentasjon av oppdragsgiveren, tidligere relevant arbeid og en kort avsnitt av teamets initielle løsning til tjenesten, Kundens Profil.
Kapittel 3: Design av prosjektet	Presentasjon av mulige framgangsmåter og løsningsalternativer for å løse problemstillingen. Valg av verktøy, språk og utførelsen av prosjekt med prosjektmetodikk som innebærer utviklingsmetodikk, prosjektplan, risikovurdering og evalueringsplan.
Kapittel 4: Detaljert plan	Presentasjon av design, modeller og diagrammer som ble brukt til forklare løsningen vår, detaljert.

Kapittel 5: Evaluering av tjenesten	Presentasjon av de ulike evalueringsmetodene gruppen har brukt i gjennom prosjektprosessen.
Kapittel 6: Diskusjon	Presentasjon av diskusjon rundt fremgangsmåten, konsekvenser av valgt gruppen har gjort, problemer som dukket opp og hvordan det påvirket resultatet.
Kapittel 7: Konklusjon	Presentasjon av denne endelig konklusjonen og hvordan prosjektet eventuelt kan arbeides videre med.
Kapittel 8: Referanser	Referanser som ramser opp kilde- og figurliste brukt i rapporten.
Kapittel 9: Vedlegg	Vedlagt innlegg som er referert til i rapporten.

2 Prosjektbeskrivelse

Prosjektet bygger videre på tjenesten Kundens Profil som Stacc Insight leverer i dag. Denne tjenesten lar banker hente ut økonomiske opplysninger fra forskjellige tredjeparts-registre for å få et helhetlig bilde av en kunde. Grunnlaget til prosjektet er å utvide tjenesten til flere registre.

2.1 Praktisk bakgrunn

Stacc Insight ønsker å utvide Kundens Profil med å hente ut data fra flere registre, slik at bankene som benytter tjenesten får et bedre og mer pålitelig syn på privatpersoners økonomiske tilstand. Ved å utnytte registre slipper kunden å gi informasjon, som allerede ligger lagret i registre, til banken. Saksbehandleren slipper også å manuelt skrive inn informasjonen. Dette fører til en enklere opplevelse for kundene, saksbehandleren og i tillegg er registrene en mer pålitelig kilde for banken. Ideen og visjonene til denne tjenesten ble videreført under første møte mellom gruppen og Insight, ledet av Kjetil Salomonsen.

Kjetil ga informasjon og foreløpige krav som var stilt til prosjektet, i tillegg til sine egne ideer og forslag om løsninger. Kjetil er ansatt hos Insight som senior utvikler/arkitekt, og har flere års erfaring som utvikler i flere banker. Hans rolle i prosjektet er som ekstern veileder.

2.1.1 Prosjekteier

Prosjekteieren, Stacc Insight, er del av et konsern, Stacc, som leverer forskjellige typer tjenester til banker og finansselskaper som Sparebank1, Trygg og Nordea. Stacc har over 100 ansatte og 40 års erfaring som leverer cloud-basert, digitale løsninger og konsulenttjenester. Stacc-konsernet består av Insight, Flow, Core og X.

2.1.2 Tidligere arbeid

Som nevnt under 1.1 Motivasjon og mål, tilbys Kundens Profil til banker for å få en oversikt over deres kunder. Dette skjer ved at kunden skriver inn informasjon om seg selv. Kundens Profil bruker denne informasjonen til å hente mer informasjon fra flere registre. Stacc Insight leverer foreløpig dagens tjeneste som en skyløsning, der integrasjonene mot registrene blir kjørt på Java Servlet-implementasjonen Tomcat. Denne tjenesten arbeides med for å lage APIer for enkeltmoduler og enkelttjenester.

Tjenestene gruppen skal lage går mot registre som ikke er implementert og er skrevet ved hjelp av annen teknologi, noe som fører til at det tidligere arbeidet gjort i stor grad ikke er relevant. Det som kan være relevant er å se på hvordan de tidligere løsningene er satt opp for å kjøre i skyen.

2.1.3 Initielle krav

Gruppen fikk selv velge hvilken teknologi som skulle bli brukt til integrasjon og hadde muligheten til å innføre flere teknologier om det var tid. Det ble satt opp flere delmål slik at hvis det var mer tid og kapasitet, ville det vært mulig å gjøre mer.

Kravene til prosjektet er:

- Tredjeparts integrasjon.
    - Integrere de fire valgte registrene inn i en RESTful Web API-løsningen.
- Oversiktlig og veldokumentert tjeneste.

    - Koden må inneholde kommentarer som forklarer hva funksjonen til alle delene av den implementerte koden gjør. I tillegg må gruppen sørge for at det endelige produktet bruker en vel anerkjent spesifikasjon til dokumentasjon.
- Testing av tjenesten
    - Løsningen må evalueres kontinuerlig gjennom aktuelle testmetoder som ressursevaluering, enhetstesting og manuell evaluering.

2.1.4 Initiell løsningsidé

Den initielle løsningsidéen var å lage en rekke APIer som henter relevant kundeinformasjon fra hvert sitt tredjeparts-register utvalgt av Stacc Insight. Kundens Profil kaller på APIene for å få tak i den dataen som trengs. Denne løsningen skal utvikles som en Internal API kombinert med Web API og må være oversiktlig og veldokumentert slik at Stacc Insight kan fortsette å arbeide videre med denne tjenesten. Løsningen bruker backend-teknologien ASP.NET.

3 Prosjektdesign

Prosjektet besto av å hente og levere data fra fire registre; Kjøretøyregisteret, Skipsregisteret, Rettssjekk og Løsøreregisteret. Dette gjøres ved bruk av et API (Application Programming Interface). Et API er definert som en spesifikasjon av mulige interaksjoner med en programvarekomponent (Hubspire, u.å.). Den kan kjøres på mange konfigurasjoner, en av de mest brukte er å kjøre på en nettverksserver, noe de aller fleste programmeringsspråk støtter. Dette betyr at mulighetene er mange for hvordan prosjektet skal bli utført.

3.1 Valg av løsning

Det endelige produktet skal integreres med Kundens Profil og må bli utviklet med tanke på dette og kravene til oppgaven. Et av kravene var å bruke en RESTful Web API. En Web API er et API som bruker HTTP-protokollen, altså protokollen som brukes av nettlesere. Denne typen definerer endepunkter og godkjente forespørsler, og kan kombineres med Composite, Open og Internal API.

REST, Representational State Transfer, er en arkitektonisk programvarestil som bruker en delmengde av HTTP. Det brukes ofte til å lage interaktive applikasjoner som bruker webtjenester. Et API er RESTful om den følger noen spesifikke retningslinjer, f.eks at APIet skal være 'stateless' og bruke en delmengde av HTTP (Wikipedia, 2021).

3.1.1 Open API

En Open API er en type API som er tilgjengelig for alle som er på internettet. En slik API kan av og til kreve et API nøkkel som man kan få tak i ved å registrere seg.

3.1.2 Internal API

Internal API refererer til et API som bare kan brukes innad i en bedrift med begrenset tilgang slik at forskjellige interne deler kan dele ressurser. Denne typen API er i motsetning til Open API vanligvis ikke tilgjengelig på det offentlige internettet.

Figur 2: Eksempel på Composite API respons

3.1.3 Composite API

En Composite API er et API som returnerer flere endepunkter i ett kall. Dette gjør at man, i stedet for å kjøre tre separate kall til et API, kan gjøre et kall og få respons fra flere endepunkter om gangen. Composite API kan altså redusere ressursbruken ved å unngå at en bruker må sende flere requests.

```
{
    "success": true,
    "ssn": "xxxxxx-xxxxx",
    "composite-response": [
      {
        "id": "kjoretoyRegisterRespons",
        "body": {
            // Informasjon fra Kjoretøyregisteret.
        },
        "success": true,
        "status": 201
      },
      {
        "id": "LoesoereRegisterRespons",
        "body": {
            // Informasjon fra Løsøreregisteret.
        },
        "success": true,
        "status": 201
      }
    ]
  }
```

I figur 2 vises et eksempel på hvordan en respons kunne sett ut om prosjektet ble laget som en Composite API. I eksempelet har brukeren sendt en request med et personnummer (SSN) og fått en respons som inneholder data fra Kjøretøyregisteret og Løsøreregisteret.

3.1.4 Diskusjon og valg av løsning

Open API integrert i en Web API er en kjent kombinasjon, men med tanke på at prosjektet skal brukes privat, gir dette lite mening. Internal API er derimot passende siden Kundens Profil bare skal være tilgjengelig innad i banken.

Registrene gruppen endte opp med å få tak i var Kjøretøyregisteret og Rettssjekk og disse har ulike parametre. De bruker registreringsnummer og vatnum, respektivt. Composite API kunne vært relevant om alle register-endepunktene brukte samme parameter og det ikke skulle legges til flere. Da kunne Kundens Profil settes opp slik at når man skriver inn det gjeldende parameteret og trykker enter, vil APIet bli kalt og alle feltene som dekkes av registrene fylles ut. En Composite API kan også settes opp slik at den kaller på flere APIer med *ulike* parametre. Det er ikke nødvendig i dette tilfellet siden det er like praktisk som å lage separate APIer, og når det er på så liten skala vil ikke ressursbruket ha noe å si. Denne løsningen vil i

tillegg føre til at tjenesten blir mindre pålitelighet, siden Kundens Profil vil miste tilgang til alle registrene om den ene modulen kræsjer.

Register-endepunktene brukte forskjellige parametere og det gav derfor mest mening å dele integrasjonen opp i flere APIer som bruker Web API kombinert med Internal API. Sikkerheten som gjør den til en Internal API vil bli forsikret av Stacc Insight og bankene selv.

3.2 Valg av verktøy

3.2.1 Programmeringsspråk/rammeverk

3.2.1.1 .NET

.NET er et kraftig rammeverk med bredt bruksområde. ASP.NET er et rammeverk som bygger videre på .NET for å tilby en effektiv måte å sette opp en web-server på. .NET tilbyr i tillegg ASP.NET tjenester som automatisk setter opp tilhørende OpenAPI dokumentasjon når du lager et api. Det finnes flere språk som kan bli brukt med .NET, hvor C# er det mest brukte.

3.2.1.2 Node.js

Node.js er en runtime bygget på JavaScript som gjør at man kan bygge raske og skalerbare nettverksapplikasjoner. Det krever få linjer med kode for å sette opp en server i Node.js, takket være effektiviteten til JavaScript. Ved hjelp av "node modules" er det god støtte for å sette opp større og mer kompliserte systemer. Ettersom Node.js kjører på JavaScript er det også god cross-platform støtte.

3.2.1.3 Java

Java er et av de største og mest brukte programmeringsspråkene, noe som betyr at det finnes mye dokumentasjon og støtte. Det finnes flere måter å sette opp servere i Java. En av de mer populære måtene er Tomcat. Kompleksitetsnivået til Java er noen lignende C, som betyr at det kan ta tid å sette opp prosjektet riktig. Prosjektet tar i tillegg opp mye diskplass som skyldes de store mengdene med biblioteker som krever for at Java skal kunne kjøre.

3.2.1.4 Andre alternativer

Det finnes flere alternativer enn de som er nevnt ovenfor, men disse ble ikke vurdert på grunn av dårlig støtte eller fraværende ønske fra gruppen. Alternativene er blant annet Python og PHP.

3.2.1.5 Diskusjon av alternativene og valgt løsning

Alle tre alternativene har blitt sett på som aktuelle, hver med sine styrker og svakheter. Java er et språk alle i gruppen har fått god kjennskap til gjennom skolen. Kundens Profil er i tillegg skrevet i Java. Det betyr at det både ville vært enkelt å få hjelp fra utviklerne av Kundens Profil, og det ville vært enklere for utviklerne å vedlikeholde koden etter prosjektet er over.

.NET ble raskt det mest populære alternativet i gruppen. Selv om de fleste ikke har god kjennskap til hverken .NET eller C# så alle på dette prosjektet som en god mulighet til å lære dette på. Ekstern veileder har tidligere jobbet mye med .NET og har dermed gode kunnskaper rundt dette.

Node.js ble også vurdert, men grunnet lite interesse fra gruppen ble ikke dette alternativet konkurransedyktig.

Valget endte derfor opp med å bli .NET, med det tilhørende ASP.NET rammeverket.

3.2.2 IDE/koderedigeringsprogram

Gruppen bruker forskjellige koderedigeringsprogram etter hvilke operativsystem de kjører i. Begrunnelsen til dette er fordi MacOS og Linux ikke kan kjøre “Visual Studio 2019” (som er et koderedigeringsprogram som er utviklet med god kompatibilitet til .NET rammeverket). MacOS brukerne i gruppen bruker “Visual Studio for Mac”, mens de som har Linux bruker “Visual Studio Code”, og de med Windows bruker “Visual studio 2019”. Alle de ovennevnte koderedigeringsprogrammene har alle egenskapene som trengs til prosjektet.

3.2.3 Kildekodedepot

Bitbucket er et Git-basert kildekodedepot. Gruppen har jobbet mye med et annet alternativ, Github, men siden Stacc Insight som standard bruker Bitbucket gjør gruppen det samme.

3.2.4 Spesifikasjon til dokumentasjon

Når et produkt eller en tjeneste opprettholder visse standarder sier man at den er laget til en spesifikasjon. Når to applikasjoner er enige om hvilke standard som blir brukt kan de kommunisere med hverandre. Etter ønske valgte gruppen å bruke OpenAPI som sin spesifikasjon.

3.2.5 Rammeverk

Det er standard å bruke ASP.NET som rammeverk i web-prosjekter ved bruk av .NET Core, da .NET Core er kryssplattform og mer rettet mikrotjenester (Microsoft, u.å). På grunn av dette, og det faktum at dette rammeverket passer bra til løsningen gruppen er ute etter, vil prosjektet ta bruk av dette.

3.2.6 Tekstredigering

Google Docs er en tjeneste levert av Google som lar deg opprette og redigere tekstdokument online sammen med andre. Alle medlemmene i gruppen har tidligere erfaring med bruk av Google Docs og opplever at dette er en pålitelig tjeneste. Google Docs vil derfor bli brukt av gruppen som tekstredigeringsverktøy.

3.2.7 Docker

Docker (u.å.) er et verktøy laget for å kunne kjøre tjenester på toppen av et operativsystem(OS). Docker fungerer ved å pakke all relevant informasjon og alle avhengigheter inn i et bilde, som gjør det enkelt å laste opp til registre for lagring og deling. I dette bilde står informasjonen som gjør det mulig for Docker å bygge en container som kjører tjenesten. Alle containere henter bare ut nødvendige biblioteker og ressurser fra det

underliggende OS-et. Ettersom containerne kjører i et lag over operativsystemet til maskinen som hoster containeren, er man i utgangspunktet begrenset til å kjøre containere på samme OS. Windows har sluppet unna dette med å kjøre et virtuelt linux-OS mellom Docker og Windows, som tillater maskiner med Windows å kunne kjøre containere med linux.

3.2.8 Kubernetes

Kubernetes (u.å.) er et system utviklet av Google for orkestrering av et miljø med automatisk distribusjon, skalering og behandling av containere. Utover dette tilbyr Kubernetes blant annet muligheten til å eksponere endepunkter, persistent lagring og oppbevaring av krypterte hemmeligheter.

3.2.9 Infra

Infra (u.å.) er en skrivebordsapplikasjon som forenkler det å overvåke og vedlikeholde et kubernetes-kluster. Dette gjøres ved å vise frem informasjon på en god og oversiktlig måte. Det største bruksområdet til Infra er å enkelt kunne se ressursbruken og loggene til tjenestene som kjører.

3.2.10 Kommunikasjon og planlegging

Hovedsakelig vil kommunikasjon innad i gruppen skje via stemmechatten Discord siden alle i gruppen har god erfaring med den og den har de funksjonene gruppen er ute etter (f.eks. skjermdeling og fildeling).

Kommunikasjon med veilederne vil skje via Microsoft Teams, da dette er en mer anerkjent stemmechat.

Til planlegging bruker gruppen en kombinasjon av Notion og Google Calendar.
Notion er en tjeneste som leverer komponenter som notater, databaser, kanban-tavler, wikier, kalendere og påminnelser (Notion (app), 2021). Notions tjenester er online og gruppen har brukt den i tidligere prosjekter, av disse grunner bruker gruppen denne applikasjonen.

Google Calendar er en kalendertjeneste hvor man blant annet kan opprette møter i grupper slik at alle får det opp i sin egen kalender. Alle gruppemedlemmene bruker denne tjenesten daglig, av denne grunn er dette valgt.

3.3 Prosjektmetodikk

3.3.1 Utviklingsmetode - Scrum

Scrum er et prosessrammeverk som blir brukt til å kunne styre produktutvikling og annet kunnskapsarbeid. Scrum er en empirisk metode, ettersom selve metoden gir arbeidsgrupper et verktøy for å kunne etablere en hypotese om hvordan de tror noe fungerer. Prøve metoden ut, reflektere, og deretter utføre justeringer som kan være nødvendig.

De fleste verdiene Scrum anser som viktige er fokus, forpliktelse, mot, respekt og åpenhet. Alle verdiene er viktig for prosjektet, men for at gruppen, veilederne og Stacc Insight skal klare å jobbe godt sammen, kreves det mot fra alle medlemmene i gruppen. Dette trengs for å komme ut av komfortsonen med å kontakte, og muligens purre, kontaktpersoner for å få det som trengs for å jobbe med prosjektet. Ikke minst må det være respekt og åpenhet i mellom gruppen, veilederne og Stacc Insight.

Det er ulike og spesifikke roller, innen Scrum, som gruppen har delt ut. En produkteier, som har ansvaret for å håndtere det som kalles “Produkt backlog”, som inneholder gruppens prioriterte funksjonaliteter, og brukt for å oppnå teamets ønsket resultat. Scrum Master, som er ansvarlig for å sikre at teamet bruker agile prinsipper og verdier, og samtidig følger prosessen og praksiser som teamet har blitt enige om å bruke. Et utviklingsteam som lager et produkt etter produkteierens prioritering. Alle utviklerne er like ansvarlig for å levere et produkt som er i henhold til det som er bestilt.

Scrum er også tenkt på som en smidig rammeverk for prosjektledelse. Dette er i form av at Scrum også beskriver møter, verktøy, og roller som fungerer sammen for å hjelpe teamet.

Kort beskrevet er Scum et rammeverk for å få utført arbeid og agile er et tankesett. Dette tankesettet jobber godt sammen rammeverk som Scrum, og støtter smidig arbeidsmiljøer.

Dette prosessrammeverket ble valgt fordi det er en god og oversiktlig måte å fokusere på de små oppgavene, mens det store bildet i baktanken. Prosjektet gjennomføres på en inkrementell måte, hvor man starter med det viktigste og mest grunnleggende, og bygger

deretter videre. Ved å gjennomføre inkrementelt, jobber man mer effektivt. Det er også en god og enkel måte å dele opp teamet på, slik at alle har sine hovedroller og føler seg mer inkludert. Scrum er enkelt å forstå, men kan fremdeles være vanskelig å bruke. Derfor er det viktig å eksperimentere og utforske, samtidig som man har egen innsats.

3.3.2 Prosjektplan

Vedlagt ligger gruppens GANTT-skjema, som brukes for å beskrive mer detaljert planleggingen og kartleggingen av de ulike oppgavene teamet har gjennomført under prosjektperioden (Vedlegg 10.2, GANTT-skjema og Vedlegg 10.3, Detaljert GANTT-skjema).

Selve skjemaet er bygget opp i fem ulike faser, som hver inneholdt ulike viktige deloppgaver. De to første fasene gikk ut på diskutering rundt prosjektet gruppen fikk fra Stacc Insight, planen for videre arbeid og oppdeling av arbeidet med de obligatoriske innleveringene og utviklingen av tjenesten. Den tredje fasen gikk hovedsakelig ut på å gå dypere inn på utviklingen og opprettelsen av tjenesten, og i de to siste fasene brukte gruppen tiden på evaluere tjenesten og kjøre en siste sluttspurt av prosjektet.

Gruppen møtte på noen uventede problemer underveis i utviklingsfasen, som førte til at gruppen måtte ta grep og gjøre noen justeringer av det estimerte tidsbruket som ble satt opp under GANTT-Skjemaet under punkt 10.3. Dette var imidlertid ikke en stor ulempe, ettersom gruppen fikk gjort det arbeidet som var nødvendig for å dekke kravspesifikasjonene, nevnt under 2.0 prosjektbeskrivelse.

3.3.3 Rollefordeling

Gruppen valgte tidlig i prosjektprosessen å ha en løs rollefordeling, ettersom alle medlemmene hadde en god relasjon med hverandre fra før av.

Fra start til slutt av prosjektet, hadde gruppen daglige møter, hvor hvert medlem har snakket om fremdrift og status rundt arbeidet sitt til de andre gruppemedlemmene kontinuerlig, og fikk mottatt tilbakemeldinger og eventuelle innspill. Planen var satt opp slik at alle i gruppen måtte jobbe sammen, for å holde kommunikasjonen og arbeidet gående rundt arbeidet av

tjenesten. Det var også planlagt at alle gruppemedlemmer kunne jobbe på egenhånd, men da måtte hele gruppen bli enige om dette på forhånd.

3.3.4 Risikovurdering og håndtering

Det ble valgt å bruke et risikoskjema, som da blir brukt til å avdekke ulike potensielle risikoer som kan forekomme innom prosjektperioden.

Risikohåndteringsplanen beskriver ulike tiltaker for å unngå at risikoen inntreffer og hvilke faser risikoen er aktuell (Vedlegg 10.1, Risikohåndteringsplan). På denne måten blir det avdekket hva som skal gjøres dersom noen av risikoene skulle inntreffe. Dette anses for å gi en bedre kontroll under prosjektperioden da potensielle trusler, som blir ansett som de mest truende for prosjektet, er minimert ved at de blir tatt hensyn til.

Sannsynlighet (S) og dens innflytelse eller konsekvenser (K) av de ulike suksessfaktorene blir skalert ut i fra 1-6, hvor:

- 1-2 viser lav, og har fargen grønn.
- 3-4 viser middels, og har fargen gul.
- 5-6 viser høy, og har fargen rød.

Ved å multiplisere S og K så blir risikofaktoren (RF) fremhevet, og dette viser hvor problematisk hver faktor er.

En risikofaktor på under 9 blir ansett som uproblematisk, mens en skala på 10-18 er middels problematisk og over 18 er problematisk. Til slutt blir det beskrevet hvilke tiltak som kan gjøre for å unngå at de mulige risikoene inntreffer, når risikoene er aktuelle og i hvilke faser de kan oppstå.

3.4 Evalueringsplan

Evalueringen av tjenesten som gruppen utviklet blir foretatt fortløpende. Den vil vurdere om tjenesten fungerer godt basert på forventningene til kundene av Stacc Insight og kravspesifikasjonene som har blitt satt opp i kapittel 2: Prosjektbeskrivelse. Prosjektet blir ikke evaluert på sikkerhet siden bankene selv er ansvarlige for restriksjon av tilgang.

Gruppen ble enige om tre ulike evalueringsmetoder som var aktuell å bruke gjennom prosjektprosessen.

- Ressursevaluering, som skulle teste om løsningen gruppen har utviklet, bruker mindre ressurser sammenlignet med de allerede kjørende tjenestene.
- Det blir lagd enhetstester som evaluerer om funksjonene i programmet fungerer som de skal.
- Til slutt har gruppen manuell evaluering, som vil hovedsakelig sjekke om løsningen er forståelig og følger de reglene som er spesifisert i dokumentasjonen.

3.4.1 Ressursevaluering

Ressursevaluering er testing som vil hovedsakelig gå ut på å sammenligne ressurseffektiviteten mellom integrasjonene gruppen har implementert, med de eldre løsningene som Stacc Insight har allerede opprettet for Kundens Profil. Selve prosessen er bedre forklart under kapittel 5: Evaluering. Her beskriver gruppen mer detaljert om hvordan selve handlingen kommer til å bli utført, hva slags utstyr og verktøy som er relevant i denne sammenheng, og om resultatet endte opp slik som gruppen hadde planlagt i utgangspunktet.

3.4.2 Enhetstesting

Enhetstesting er testing av ulike kodesegmenter for å kontrollere at en de returnerer en korrekt verdi. Planen var å bruke NUnit, som er et rammeverk for enhetstesting og blir brukt til å lage ulike unit-testing metoder for ulike programmeringsspråk innen .NET. Enhetstesting vil bli gjennomført under hele prosjektperioden for å kunne sjekke om de metodene og komponentene som kommer til å bli implementert, fungerer som det de skal.

3.4.3 Manuell evaluering

Manuell evaluering er testing som involverer tett samarbeid med prosjekteieren, Stacc Insight. Det føltes viktig for gruppen å holde tett kontakt med Stacc Insight kontinuerlig gjennom hele prosjektet, da deres påfølgende respons var viktig for prosjektets utvikling. Stacc Insight har tilgang til all kildekoden for utviklingstjenesten, som lar dem evaluere prosjektgruppens arbeid ut i fra kravene gruppen har spesifisert i punkt 2.1.3, for å så kunne gi konstruktive tilbakemeldinger.

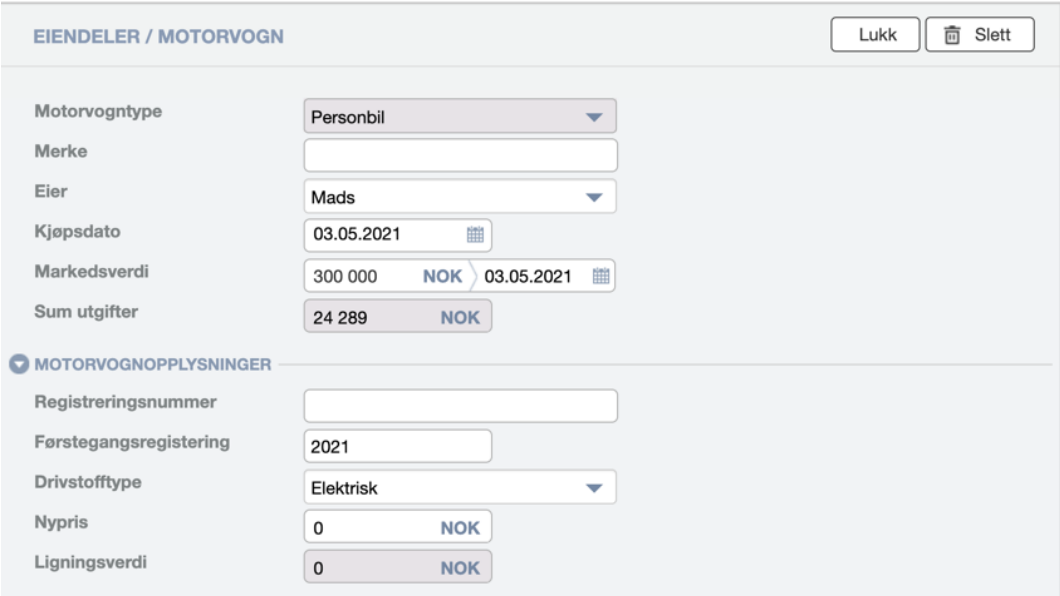
3.4.4 Konklusjon av evalueringsplan

Disse metodene ble kontinuerlig brukt i programmeringsperioden for å sjekke om gruppens løsning var god nok og oppfyller kravene gitt av oppdragsgiveren. Sluttvurderingen av løsningen var planlagt å gjøre når mesteparten av prosjektet er ferdigstilt.

4 Detaljert plan

All kode er lagt til som vedlegg under punkt 10.4.

4.1 Kundens Profil integrasjon



Figur 3: Kundens Profil skjermbilde

Figur 3 viser et skjermbilde av Kundens Profil slik den ser ut uten integrering med APIene gruppen lager. Dette eksempelet viser siden som skal integreres med Kjøretøyregisteret, men de andre sidene går ut på samme konsept.

Hvert felt må manuelt fylles inn av saksbehandleren, som ofte får informasjonen via kunden. Denne prosessen er tidkrevende, og derfor er målet til API-integrasjonen å automatisere så mye som mulig av prosessen.

Når APIene er integrert inn i løsningen så forandrer dette funksjonaliteten til applikasjonen. Når saksbehandleren fyller ut et felt som representerer et parameter i en av de relevante APIene vil Kundens Profil sende et kall til det APIet og legge returdataen i de gjeldende feltene. Dette vil føre til mindre manuelt arbeid, som igjen sparer tid.

Det mest ønskelige hadde vært om Kundens Profil kunne gitt et personnummer, som allerede ligger inne på bankenes systemer, til APIene og fått all relevant registerinformasjon tilbake. Kundens Profil kan deretter, automatisk, fylle inn informasjonen i de tilhørende feltene. Det er mulig å lage en slik løsning, men for å kunne lage det trenger gruppen en måte å få tak i registerdata ved å bruke et personnummer. Flere av registrene har slike endepunkter, men

gruppen fikk ikke tilgang til disse pga. treg kommunikasjon med register-administratorne. I tillegg fordi register-administratorne måtte forsikre seg om at tilgang til den sensitive informasjonen som er forbundet til register-endepunktene ikke blir gitt til feil person. Tregheten førte også til at gruppen endte opp med å kun bruke Rettssjekk og et suboptimalt endepunkt til Kjøretøyregisteret. Det var heller ikke nok tid til at gruppen fikk konkrete endepunkter som var relevant til Rettssjekk-APIet, som førte til at gruppen selv måtte velge relevante endepunkter.

Kundens Profil er bygget opp slik at den har flere sider man kan føre informasjon i, med forskjellige kategorier på hver side. F.eks. kjøretøy på en side og sjøfartøy på en annen. Av denne grunn tenker gruppen at det vil være praktisk om Kundens Profil kunne søke på integrasjonen for én av de implementerte registrene om gangen. Dette vil tillate at informasjonen fra registrene kan legges inn i skjemaene etter hvert som man fyller inn feltene som endepunktene trenger. Det er også en praktisk løsning siden alle integrasjonene bruker forskjellige parametere.

Kjøretøyregisteret har et endepunkt som tar personnummer og gir tilbake informasjon om tilhørende kjøretøy. Dette endepunktet fikk ikke gruppen tilgang til, så gruppen valgte å jobbe med den offentlige informasjonen som Kjøretøyregisteret tilbyr som et alternativ. Løsningen dette førte til, er et API som tar inn registreringsnummeret til et kjøretøy og returnerer merke, førstegangsregistrering, drivstoff og motorvogn-type. Denne løsningen er hjelpsom, men den har mangler, noe som blir fortalt om senere i dette kapittelet.

4.2 Konfigurasjon av APIer

Som nevnt i 4.1, så endte gruppen opp med å bare implementere Kjøretøyregisteret og Rettssjekk. Gruppen fikk ikke vite hvilke konkrete endepunkter eller data som skulle bli hentet ut fra Rettssjekk. Derfor valgte gruppen selv endepunkter i Rettssjekk som de mente ville være relevant. Gruppen fikk ikke tilgang til det ønskede endepunktet i Kjøretøyregisteret og endte opp med å bruke et alternativt endepunkt. Rettssjekk og Kjøretøyregisteret er begge REST Web API som krever autentiseringsnøkler og parametere for å hente ut data. Rettssjekk er brukt til å få ut informasjon angående rettssaker tilknyttet bedrifter, for mer se avsnitt 4.2.2. Dette er blant annet informasjon om dato, partier og sammendrag. Kjøretøyregisteret er vedlikeholdt av statens vegvesen og har all informasjon angående kjøretøy og førerkort tilknyttet privatpersoner.

For at APIene gruppen lager skal kunne hente data, må de ha tilgang til en API-nøkkel slik at de har tilgang til registeret de skal hente data fra. Denne nøkkelen er en unik identifikasjonsnummer som er brukt for å identifisere brukere. Registrenes endepunkter har forskjellige nivåer på tilgang. Gruppen vurderte å ha API-nøkkelen i sin egen klasse som en statisk variabel, men dette fører til ekstra overhead siden koden må forandres for hver bank som bruker koden. Dermed endte det opp med å lage APIene slik at de henter nøklene fra en miljøvariabel i miljøet. Dette legger opp til at ressursbeskrivelsen til tjenestene i Kubernetes, som beskriver oppsettet til tjenestene, legger miljøvariablen inn i containeren APIet kjører på. Gruppen valgte denne måten, fordi det gjør det enklere å forandre på hvilken nøkkel som blir brukt, da hverken koden eller containeren må forandres på. Hvordan dette blir satt opp vil gruppen beskrive i dokumentasjonen som beskriver hvordan man bruker tjenesten.

Sikkerhet i banktjenester er viktig. Om en uønsket bruker får tilgang til APIene kan de få tilgang til sensitiv informasjon fra Rettssjekk. Dette er ikke et problem med Kjøretøyregisteret siden endepunktet som blir brukt er offentlig. Sikkerheten blir ikke håndtert av gruppen, men vil bli håndtert av bankens systemer. De bruker en metode som heter 'Skallsikring', hvor systemet krever en JWT (JSON Web Token) som verifiserer hvem brukeren er og om brukeren skal ha tilgang til systemet (JWT, u.å). Skulle en bruker ikke ha et gyldig JWT, vil han ikke få tilgang til noe i systemet. Siden APIene gruppen lager ligger inne i systemet, så er det kun brukere med gyldige JWT som får tilgang.

4.2.1 Konfigurasjon av Kjøretøyregister-API

Som nevnt i 4.1 Kundens Profil integrasjon fikk ikke gruppen tilgang til det ønskede endepunktet, men endte opp med å bruke et alternativ som tilbyr: merke, førstegangsregistrering, drivstoff og motorvogn-type, og bruker registreringsnummeret til et kjøretøy som parameter. Endepunktet gruppen endte opp med å bruke tilbydde mer informasjon enn det integrasjonen returnerer, men denne informasjonen var irrelevant for Kundens Profil. Gruppen endte derfor opp med å implementere bare henter ut den relevante dataen.

```
string url = string.Format(endPoint + registerPlate);
WebRequest reqObj = WebRequest.Create(url);
reqObj.Method = "GET";
reqObj.Headers.Add("SVV-Authorization:" + apiKey);
try
{
    HttpWebResponse resObj = (HttpWebResponse)reqObj.GetResponse();

    string result = "";
    using (Stream stream = resObj.GetResponseStream())
    {
        StreamReader sr = new StreamReader(stream);
        result = sr.ReadToEnd();
        sr.Close();
    }
    KjoretoyRoot match = JsonConvert.DeserializeObject<KjoretoyRoot>(result.Trim('[').Trim(']'));
    return match;
}
```

Figur 4: Kjøretøyregister endepunkt-kode

I figur 4 vises koden som brukes for å hente data fra endepunktet. Man starter prosessen med å kombinere endepunktet med registreringsnummeret. Deretter hentes dataen med en GET-forespørsel som så konverteres den til et KjoretoyRoot objekt. Objektet er en C# klasse som representerer JSON-objektet endepunktet returnerer. Videre blir den relevante dataen filtrert ut av dette objektet og plassert inn i et nytt objekt som representerer JSON-objektet, som blir returnert.

4.2.2 Konfigurasjon av Rettssjekk-API

Rettssjekk-APIet krever at brukeren autoriserer seg selv ved et API-kall med brukerens API-credentials. API-credentials er kontoens email, passord og API-nøkkel. API-kallet, om vellykket, returnerer et godkjennings-token og informasjon om kontoen. Tokenet og API-nøkkelen blir lagt inn som miljøvariabler i miljøet. Dette tokenet og innholdstypen, i vårt tilfelle “application/json”, må være med i headeren til alle API-kall. De fleste kall krever også parametere.

Gruppen fikk ikke vite hvilke data som Stacc Insight ville ha returnert fra Rettssjekk-APIet, så gruppen måtte selv velge relevante endepunkter. Her kom 3 endepunkter frem:

- Søke om selskapet har konkursdata.
- Søke om selskapet har bevis på at de er betalingsdyktig
- Søke etter rettssaker selskapet har hatt og som er planlagt.

Dataen som kommer fra endepunktene kan være alt fra ingenting til for mye, og dataen måtte derfor kortes ned og begrenses til en overkommelig mengde.

Figur 5: Forkortet klassemodeller til Rettssjekk endepunktene

```
public class Solvent
{
    public string trial_date { get; set; }
    public string trial_summary { get; set; }
    public string trial_parties { get; set; }
    public string trial_type { get; set; }
    public string case_type { get; set; }
    public string role { get; set; }
    public int status { get; set; }
    public string updated_at { get; set; }
}

public class Bankruptcy
{
    public string company_name { get; set; }
    public string reg_date { get; set; }
    public string case_details { get; set; }
}

public class Trial
{
    public string trial_date { get; set; }
    public string trial_summary { get; set; }
    public string trial_parties { get; set; }
    public string trial_reference { get; set; }
    public string trial_type { get; set; }
    public string case_type { get; set; }
    public string role { get; set; }
    public string foretaksreg { get; set; }
```

Store mengder data kommer ofte når man søker etter rettssakene til eldre selskaper, og ingen data kom ofte ut når man søkte på selskaper som ikke nærmet seg konkurs. På figur 5 kan man se informasjonen gruppen valgte å hente ut.

Endepunktene til Rettssjekk-APIet kalles på nesten samme måte som i Kjøretøyregister-APIet. Forskjellen her er et identifikasjonsnummer, vatnum, og headers, som er som sagt en token og innholdstypen. Dataen som blir hentet ut konverteres også til C# klasse modeller som representerer JSON-objektet som blir returnert. Som figur 5 viser, så har de 3 endepunktene hver sin modell.

4.3 Integrering i skyen

Kundens Profil er en tjeneste som blir levert i skyen. Derfor vil det være naturlig å kjøre de nye løsningene samme sted. Skyløsningen Stacc Insight benytter seg av er Kubernetes kjørende i Azure. Det er kun et krav for å kunne kjøre en tjeneste i Kubernetes. Tjenesten må være pakket inn i en container, noe som blir løst ved hjelp av Docker.

Docker-bilde som lages for å kunne kjøre tjenestene vil bygges på Linux, ettersom serveren tjenestene blir hostet på kjører Linux. Linux er også mest hensiktsmessig å bruke for utvikling, da det er mulig å kjøre Linux-containere på Windows ved hjelp av virtualisering. Selv om .NET Core-applikasjoner hovedsakelig er utviklet for Windows, er det fullt mulig å kjøre dem på Linux.

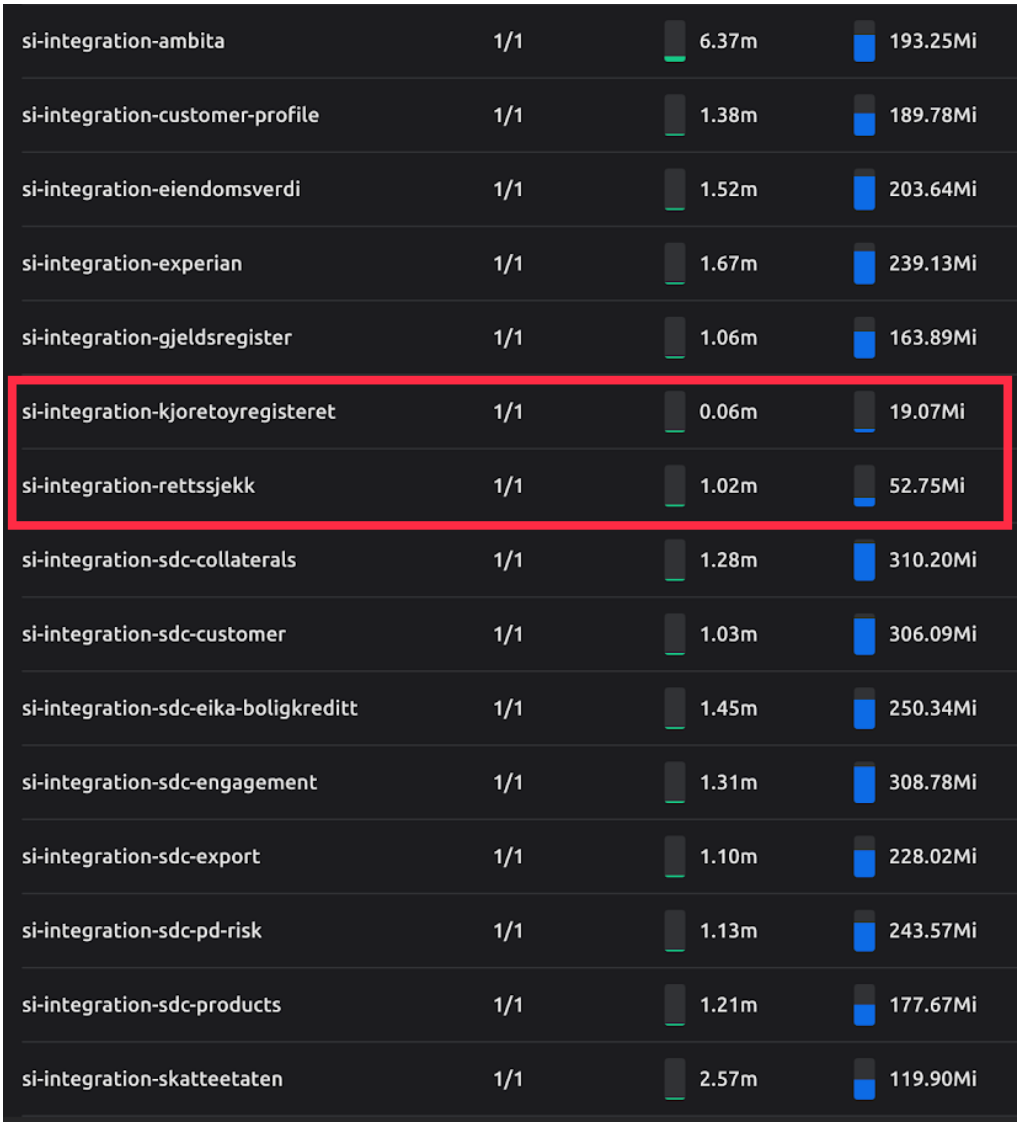
5 Evaluering

5.1 Evalueringsmetode

Etter hver iterasjon blir løsningene gruppen lager evaluert med å se hvor mye ressurser de bruker i forhold til gamle løsninger, N-unit tester og manuel evaluering av hvor godt kvaliteten og resultatene dekker målene som er stilt i prosjektbeskrivelsen.

5.1.1 Ressursevaluering

Stacc Insight ønsker å kunne se forskjell på ressursbruken mellom de allerede kjørende og de nye tjenestene. Den beste måten å sammenligne de på er ved å se på bruken av minne (RAM) og prosessorkraft (CPU). Hvor mye minne som blir brukt er ganske uavhengig av hvilken fysisk RAM-brikke løsningen kjører på, noe som gjør det til en enkel og nøyaktig måte å sammenligne på. Det er også mulig å måle CPU-forbruk, men denne vil være svært avhengig av hvilken CPU-brikke som brukes. Om de gamle og nye tjenestene ble kjørt på to forskjellige typer CPU-er, ville de ikke vært mye nytte i å måle forbruket, men ettersom begge blir plassert i samme skyløsning vil det være interessant å sammenligne CPU-bruk. For å sjekke ressursbruken til tjenestene brukes et overvåkningsverktøy for Kubernetes kalt Infra, som ble nevnt i punkt 3.2.7.



Figur 6 Ressursforbruk til tjenester kjørende i skyen

Kjøretøyregisteret og Rettssjekk heter på figur 6 si-integration-kjoretoyregisteret og si-integration-rettssjekk, henholdsvis. På figuren er det forskjell på ressursbruken i si-integration-kjoretoyregisteret og si-integration-rettssjekk, ettersom det bare er gjort spørringer mot si-integration-rettssjekk. Minnebruket til si-integration-kjoretoyregisteret ligger bare på 19.07 MB, i forhold til andre integrasjoner (se figur 6) mot andre tjenester som ligger på mellom 119 MB og 308 MB. Disse integrasjonene er skrevet i Tomcat, noe som illustrerer forskjellen på ressursbruk i forskjellige løsninger. CPU-forbruket er målt i millikjerner (m), altså tusendeler av en CPU-kjerne. Også denne er vesentlig mindre enn de andre integrasjonene.

I det skjermbildet ble tatt ble det gjort en spørring mot Rettssjekk, slik at forbruket når tjenesten er aktiv også kan bli målt. CPU-forbruket ligger da på 1.02 m, som er ekvivalent

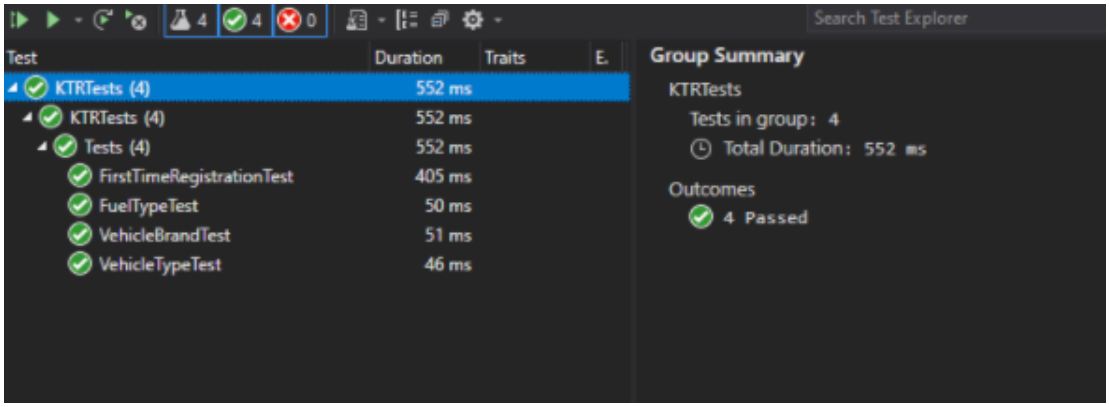
med de andre tjenestene når de er inaktive. Minnebruken ligger for Rettssjekk på 52.75 MB, ettersom det tidligere er gjort flere spørringer mot tjenesten. Manuell testing av tjenesten ved gjentatte spørringer viser at minnebruket øker med omtrent 2 MB per spørring. Minnebruket øker frem til det ligger på 160 MB, når det da faller til 125 MB. Dette skjer trolig på grunn av intern caching for å øke hastigheten på spørringer mot tjenesten.

Ser man på snittet av forbruket til de eldre løsningene ligger det på rundt 250 MB, noe som er vesentlig større enn 160 MB. CPU-forbruket når tjenestene er inaktive er omtrent 20 ganger mindre. Allerede uten noe form for optimalisering ligger forbruket til de nye løsningene langt under de eldre.

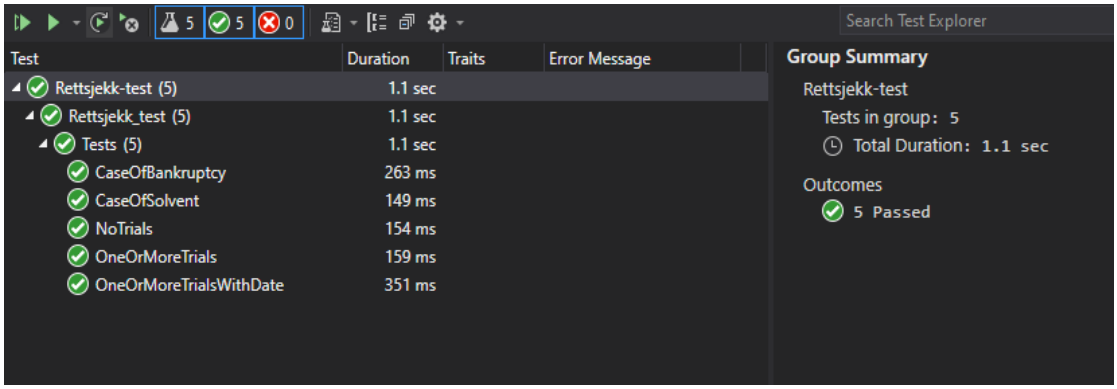
5.1.2 Enhetstesting

Som nevnt over i punkt 3.4.2 så brukes N-unit til å teste de ulike komponenter og funksjoner i implementasjonen ved hjelp av enhetstesting. Dette er noe gruppen bruker kontinuerlig under prosjektet for å teste koden i implementasjonen og validerer det slik at man unngår bugs under konstruksjonsfasen til utgivelsen av tjenesten. Enhetstesting er brukt til å teste enkle funksjoner og ikke hele implementasjonen. Stacc Insight har heller ikke stilt noe krav om dette. Slike tester er derimot viktig for å sjekke om at disse funksjonene får de riktige parametrene når de blir kalt opp og gir også fordeler for kunden som skal bruke tjenesten i å ikke trenge å ende opp med å håndtere en ustabil tjeneste. Samtidig så er disse testene veldig nyttig for forbedret estimering av testtid, noe som sparer mye tid og ressurser.

På figur 7 og 8 under vises resultatet av en gjennomført enhetstest av Kjøretøyregisteret og Rettssjekk som hver inneholder forskjellige metoder. Disse metodene tester de verdiene som er relevant for tjenesten, Kundens Profil, og inneholder de aktuelle objektene som blir evaluert basert på innholdet i parametere, og om de inneholder ekte verdier som allerede ligger i systemet og som er beskrevet i dokumentasjonen.



Figur 7: NUnit testing av kjøretøy programmet



Figur 8: NUnit testing av Rettssjekk programmet

5.1.3 Manuell evaluering

Manuell evaluering var noe gruppen og Stacc Insight hadde planlagt før selve prosjektet, og vil innebærer kontinuerlig kontakt med hverandre angående viktige punkter som kontakt med de ulike tredjeparts-registrene, implementering og filtrering av relevante data, veldokumenterte løsninger og konstruktive tilbakemeldinger om denne løsningen gjennom forskjellige prosjekt iterasjoner.

Ettersom det oppstod uventede problemer underveis med ulike register-administratorene, som ble nevnt i punkt 4.1, så hadde gruppen dessverre ikke nok tid og ressurser til å produsere det ønskede resultatet. Planen var å opprette og evaluere APIer for å hente data fra fire ulike registre, men på grunn av tregheter med Skips- og Løsøreregisteret ble gruppen enige om endring i prosjektplanen og måtte gå videre med alternativer. Gruppen arbeidet videre med de tilgjengelige APIene i fra Kjøretøyregisteret og Rettssjekk, og evaluerte disse i mot de respektive dokumentasjonene.

5.2 Evalueringsresultat

Som skrevet under punkt 1.1 så var gruppens hovedoppgave å kunne videreutvikle Kundens Profil, ved å implementere nye integrasjoner i mot ulike tredjeparts-registre som skal kunne bruke lavere eksekveringstid og mindre ressursbruk, i sammenligning til de eldre tjenestene som i dag kjører på Tomcat. Ved evaluering av resultatet er det tatt hensyn til om problemstillingen er besvart og om besvarelsen gir data som er viktig nok for å danne grunnlag for en konklusjon.

Til tross for det vanskelige arbeidsforholdet og mangler av et par viktige registre som ble nevnt under punkt 6.2, fikk gruppen positive tilbakemeldinger rundt løsningen som ble produsert.

Gruppen har fått utviklet en forståelig og veldokumentert tjeneste som henter ut aktuelle informasjon fra Kjøretøyregisteret og Rettssjekk, og som bruker godt anerkjent spesifikasjon til dokumentasjon. Disse integrasjonene endte opp med å bruke mye mindre ressurser i forhold til de allerede kjørende tjenestene og vil videre bli implementert inn i Kundens Profil.

Resultatene fra evalueringene har gitt grunnlag for noen konklusjoner:

- Tjenesten kan enkelt videreutvikles, ettersom det er mulig å integrere resten av registrene, som gruppen ikke hadde tilgang til, mot Kundens Profil.
- Tjenesten har løsninger som er verdifullt og er relevant for kunden.

6 Diskusjon

6.1 Konsekvenser av valgt løsning og rammeverk

Valget av å bruke Internal API ble gjort med tanke på at APIene bare skulle være tilgjengelig innad i bankene og at endepunktene gruppen hadde tilgang til brukte forskjellige parametere. Om endepunktene brukte samme parameter ville løsningen mest sannsynlig bestått av en Internal API og en Composite API slått sammen, men siden parameterne var ulike, kom gruppen frem til at det var bedre å løse det uten Composite API.

Den valgte måten å løse det på gav ikke gruppen noen problemer, var relativt lett å lære seg og den kan brukes til å raskt lage prototyper. I ressursevalueringen vises det at den nye løsningen bruker mindre ressurser enn de gamle. Gruppen antar at dette hovedsakelig er på grunn av valget å lage en Web API i ASP.NET, men har ikke hatt tid til å undersøke i dybden hva det konkret er som gjør den nye løsningen mer ressurseffektiv.

6.2 Andre konsekvenser

Prosjektet ble preget av mangel på tilgang til registrene, som kom fra forsinkelser og dårlig kommunikasjon med register-administratorene. Dette førte til at gruppen måtte droppe Skipsregisteret og Løsøreregisteret, bruke et mindre ønskelig endepunkt for Kjøretøyregisteret og ikke fikk tid til å utføre delmål mot slutten.

Siden det endte opp med å bli mye venting brukte gruppen mer tid på testing av løsninger innad i ASP.NET. Dette førte til at når gruppen fikk vite hvilke endepunkter de hadde tilgang til gikk utvikling av løsningen raskt.

Om prosjektet skulle utføres på nytt er alle i gruppen enige om at de skulle brukt mindre av ventetiden på testing av løsninger, og heller jobbet med delmål. Gruppen ville også ha vært mer pågående når de prøvde å få tilgang til registrene.

Gruppens utviklingsmetode, Scrum, og risikohåndteringsplanen hjalp til å forsikre at det alltid var noe å gjøre på, selv under ventetiden. På hvert møte snakket hvert medlem om hva de gjorde siden sist og gruppen ble, som en enhet, enige om hva som skulle jobbes med til neste møte.

Alt i alt gikk prosjektarbeidet etter planen til originale GANTT-diagrammet. Det ble satt inn noen uforventede justeringer under utviklingsfasen inntil testfasen, ettersom gruppen ventet lenge for å få tak i de nødvendige API-nøklene, men endte opp med å bare få API-nøkkelen til Rettssjekk og måtte bruke et offentlig endepunkt i Kjøretøyregisteret. Dette var derimot ikke en stor konsekvens for gruppen, ettersom det planlagte tidsbruket for utviklingsfasen viste seg å være rett og gruppen klarte å holde seg innenfor tidsskjemaet.

7 Konklusjon og videre arbeid

Målet med oppgaven var å utvikle en oversiktlig og veldokumentert tjeneste for å hente relevant informasjon fra fire registre. Minimumskravene var å lage en veldokumentert og oversiktlig RESTful Web API som kan hente og levere data fra fire forskjellige registre. Om gruppen ble ferdig med minimumskravene tidlig var det også aktuelt med flere delmål.

Gruppen lagde to RESTful Web APIer som inkluderer oversiktlig kode med kommentarer og bruker en godt anerkjent spesifikasjon med navn 'OpenAPI' til å lage tilhørende dokumentasjon. Målet ble ikke nådd da gruppen kun fikk implementert to registre, og heller ikke fikk gjennomført alle delmålene. Dette skjedde hovedsakelig da gruppen ikke fikk tilgang til de to siste registrene.

Prosjektet er privat eid av Stacc Insight, og skal brukes til å gjøre produktet Kundens Profil mer brukervennlig. Kundens Profil er ment for banker, men Stacc Insight kan fremdeles bruke APIene som er laget i dette prosjektet i andre prosjekter hvor det er relevant å få informasjon om kjøretøy eller om en bedrift er betalingsdyktig, dette er fordi Web APIer er allsidige og kan lett gjenbrukes.

Om Stacc Insight får tilgang til de andre APIene, og gruppen skulle jobbet videre med dette, ville gruppen ha utvidet denne løsningen slik at den inneholder så mange relevante registre som mulig. Det er også et poeng at disse registrene bruker personnummer som parameter slik at brukere trenger å skrive så lite informasjon som mulig inn manuelt, eventuelt kan parameterne være returdata fra et annet register.

Gruppen lærte, i løpet av prosjektet, viktigheten av å være pågående når man prøver å få kontakt med eksterne bedrifter. Om man ikke er det kan man ende opp med å vente for lenge slik at det går utover prosjektet. Det er også viktig å være nøye når man lager risikohåndteringsplanen slik at man alltid vet hva man kan jobbe med.

8 Referat

8.1 Kildeliste

Nes, MS. (2019) *En kort introduksjon til Scrum.* Tilgjengelig fra https://www.visma.no/blogg/en-kort-introduksjon-til-scrum/ (Lest: 4. april 2021)

Stacc. (u.å.) *About Stacc.* Tilgjengelig fra https://stacc.com/about (Lest: 4. april 2021)

Notion (app) (2021). *Wikipedia*. Tilgjengelig fra https://en.wikipedia.org/wiki/Notion_(app) (Lest: 4. april 2021)

Node.js (u.å.) *About Node.js.* Tilgjengelig fra: https://nodejs.org/en/about/ (Lest: 6. april 2021)

Microsoft (u.å.) *What is .NET?.* Tilgjengelig fra: https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet (Lest: 6. april 2021)

Microsoft (u.å.) *ASP.NET* Tilgjengelig fra: https://dotnet.microsoft.com/apps/aspnet (Lest: 6.april 2021)

Wikipedia (2021) *Java.* Tilgjengelig fra: https://en.wikipedia.org/wiki/Java_(programming_language) (Lest: 6. april 2021)

Forte Group (u.å) *The importance of unit testing or how bugs found in time will save you money.* Tilgjengelig fra https://fortegrp.com/the-importance-of-unit-testing/ (Lest: 19. mai 2021)

Hubspire (u.å) *Application programming interface.* Tilgjengelig fra: https://www.hubspire.com/resources/general/application-programming-interface/ (Lest: 19. mai)

Infra (u.å) *The Simplest Kubernetes Desktop Client.* Tilgjengelig fra: https://infra.app/ (Lest: 19.mai 2021)

Stoplight (u.å) *Types of APIs & Popular REST API Protocol. Tilgjengelig fra:* https://stoplight.io/api-types/#:~:text=Discussing different types of APIs,XML-RPC%2C and SOAP.&text=APIs (application programming interfaces) come in many forms. (Lest: 19. mai 2021)

Microsoft (u.å.) *What is Docker?* Tilgjengelig fra: https://docs.microsoft.com/en-us/dotnet/architecture/containerized-lifecycle/what-is-docker (Lest 26. mai 2021)

Kubernetes (u.å.) *Kubernetes.* Tilgjengelig fra: https://kubernetes.io/ (Lest: 26. mai 2021)

JWT (u.å) *Introduction to JSON Web Tokens.* Tilgjengelig fra: https://jwt.io/introduction/ (Lest: 27. mai 2021)

Docker (u.å.) *Docker*. Tilgjengelig fra: https://www.docker.com/ (Lest: 26. mai 2021)

Wikipedia (2021) *Specification (technical standard).* Tilgjengelig fra: https://en.wikipedia.org/wiki/Specification_(technical_standard) (Lest: 27.mai 2021)

Wikipedia (2021) *Representational state transfer*. Tilgjengelig fra:

https://en.wikipedia.org/wiki/Representational_state_transfer (Lest: 3 juni 2021)

8.2 Figurliste

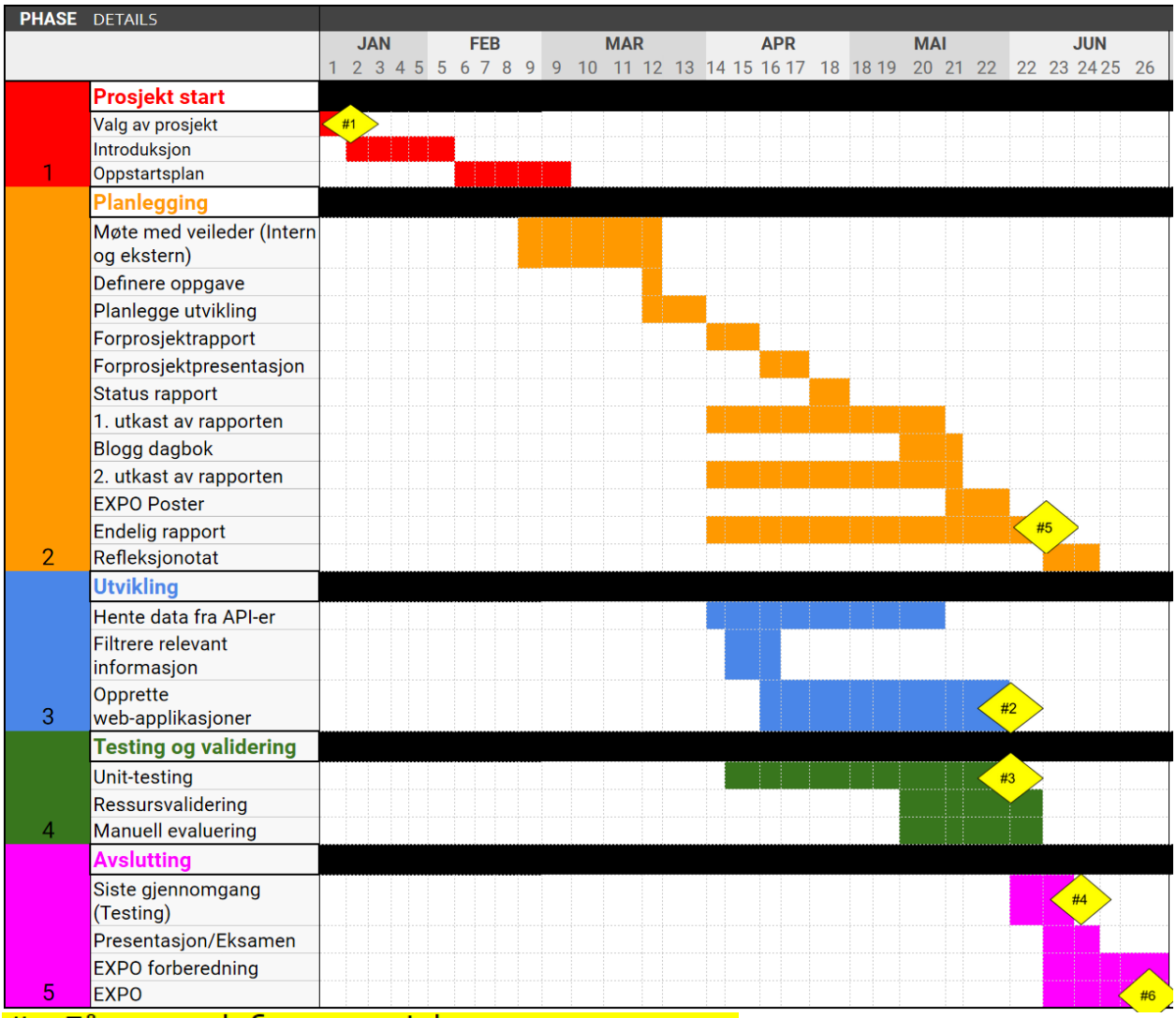
10 Vedlegg

10.1 Risikohåndteringsplan

Suksessfaktor	S	K	RF	Tiltak	FASE
Systemproblemer (Bugs, system crash, etc…)	6	3	18	Test flittig og pushe ofte til Bitbucket med kommentarer (!).	Utvikling/Testing og validering
Ikke oppnådd ønsket produkt/tjeneste pga. prioritering av tid	4	4	16	Være mer bevisst på hva hovedmålet for sluttresultatet er og prioritere det viktigste etter. Hold milepælene.	Utvikling/Testing og validering/Avslutning
Gruppemedlemmer blir syke/Langtidssykdom	2	6	12	Komme frem med en back-up plan for nye fordelinger av arbeid og ha oversikt over forventet tidsbruk.	Planlegging/Utvikling /Testing og validering
Tidsproblemer/ Forsinkelser	2	4	8	Gå gjennom den opprinnelige planen og forhold til det som er viktig til resultatet.	Utvikling/Testing og validering/Avslutning
Misforståelser i forhold til kravspesifikasjon	2	4	8	Ha tett kommunikasjon med arbeidsgiveren.	Utvikling
COVID-19/Restriksjoner	6	1	6	Finne andre kommunisering kanaler som discord, teams, osv...	Utvikling/Testing og validering

Dårlig kommunikasjon innad gruppen	**3**	**2**	**6**	Daglige møter. Være mer aktiv og bruke mer tid på kommunikasjon.	Utvikling
Problemer med utstyr	**1**	**5**	**5**	Bruke utstyret riktig.	Utvikling/Testing og validering
Endringer i kravspesifikasjon	**1**	**3**	**3**	Ha en god dialog med arbeidsgiveren og programmere løsninger som det lett å integrere til nye ting, om det trengs.	Utvikling/Testing og validering

10.2 GANTT-skjema



#1: Fått egendefinert prosjekt

#2 Opprettet et utkast av produkt/tjeneste

#3 Gjennomført første test

#4 Ferdiglagd tjenesten

#5 Skrevet ferdig en endelig rapport

#6 EXPO presentasjon

10.3 Detaljert GANTT-skjema

PHASE	DETAILS	Planned start (Week)	Planned duration (Week)	Actual start (Week) (Week)	Actual Duration (Week)	Percent Complete	
	Prosjekt start						
	Valg av prosjekt	1	1	1	1	100%	
	Introduksjon	2	5	2	5	100%	
1	Oppstartsplan	6	5	6	5	100%	
	Planlegging						
	Møte med veileder (Intern og ekstern)	9	3	10	3	100%	
	Definere oppgave	12	1	12	1	100%	
	Planlegge utvikling	12	2	12	2	100%	
	Forprosjektrapport	14	2	14	2	100%	
	Forprosjektpresentasjon	16	2	16	2	100%	
	Status rapport	18	1	18	1	100%	
	1. utkast av rapporten	14	8	14	8	100%	
	Blogg dagbok	20	2	20	2	0%	
	2. utkast av rapporten	14	9	14	9	30%	
	EXPO Poster	21	2	21	2	0%	
	Endelig rapport	14	11	14	11	0%	
2	Refleksjonotat	23	2	23	2	0%	
	Utvikling						
	Hente data fra API-er	14	5	18	2	100%	
	Filtrere relevant informasjon	15	2	19	2	100%	
3	Opprette web-applikasjoner	15	8	15	8	100%	
	Testing og validering						
	Unit-testing	15	3	15	9	100%	
	Ressursvalidering	20	4	20	4	30%	
4	Manuell evaluering	20	4	20	4	30%	
	Avslutting						
	Siste gjennomgang (Testing)	22	3	22	3	0%	
	Presentasjon/Eksamen	23	3	24	3	0%	
	EXPO forberedning	23	4	24	4	0%	
5	EXPO	23	4	24	4	0%	

10.4 Link til Koden

10.4.1 Kjøretøy Integrasjon

https://github.com/AuStien/si-integration-kjoretoyregisteret

10.4.2 Rettssjekk Integrasjon

https://github.com/KnutAaboe/si_integration_rettssjekk_fixs_headers