Høgskulen på Vestlandet

BACHELOROPPGAVE

Skybasert Automatisering av Windows-Servere: En effektiv løsning for Ambitas Regnskapsavdeling

Cloud-Based Automation of Windows-Servers: An Efficient Solution for Ambita's Accounting Department

Henrik Føllesdal Halvorsen
Marius Bråthen Reikerås
Sivert Åkernes Sæter

Dataingeniør og Informasjonsteknologi
HVL Fakultet for teknologi, miljø og samfunnsvitenskap
Institutt for datateknologi, elektroteknologi og realfag
Veileder: Remy Andre Monsen
13.05.24

Høgskulen på Vestlandet

Fakultet for teknologi, miljø- og samfunnsvitenskap

Institutt for datateknologi, elektroteknologi og realfag

TITTELSIDE FOR HOVEDPROSJEKT

Rapportens tittel: Skybasert Automatisering av Windows-Servere: En effektiv løsning for Ambitas Regnskapsavdeling	*Dato: 13.05.24*
Forfatter(e): Henrik Halvorsen, Marius Bråthen Reikerås og Sivert Åkernes Sæter	*Antall sider u/vedlegg: 62* *Antall sider vedlegg: 1*
Studieretning: Dataingeniør og Informasjonsteknologi	*Antall disketter/CD-er:* 0
Kontaktperson ved studieretning: Remy Andre Monsen	*Gradering:* Ingen
Merknader:	

Oppdragsgiver: Ambita AS	*Oppdragsgivers referanse:*
Oppdragsgivers kontaktperson: Olav Vik	*Telefon:* 913 49 086

Sammendrag:

Denne rapporten beskriver prosessen av å utvikle en automatisert løsning for Windows-servere i skyen hos Ambita. Løsningen automatiserer prosessen for å starte og stoppe gitte Windows-servere. Bakgrunnen for denne oppgaven er å redusere kostnad og ressursbruk. Produktet som er utviklet har resultert i betydelige reduserte driftskostnader for Ambita.

I rapporten blir det beskrevet prosess, drøfting og evalueringer knyttet til oppgaven.

Stikkord:

AWS	Windows-servere	Skyteknologi

Høgskulen på Vestlandet, Fakultet for teknologi, miljø- og samfunnsvitenskap
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 bachelorprosjektet “Skybasert Automatisering av Windows-Servere: En effektiv løsning for Ambitas Regnskapsavdeling". Prosjektet er gjennomført av Henrik Føllesdal Halvorsen, Marius Bråthen Reikerås og Sivert Åkernes Sæter våren 2024.

Først ønskes det å rettes takk til Ambita AS for å tilrettelegge et slikt spennende prosjekt for studentgruppen. Takk til Olav Vik som oppdragsgiver og støtte underveis i prosjektet, og takk til Philippe Haavik for formidabel veiledning.

Takk til Remy Andre Monsen vår interne veileder hos HVL. Han har hjulpet oss med god veiledning, tips og gode råd underveis i prosjektet.

FORORD 3

1 INNLEDNING 5

1.1 KONTEKST 5
1.2 MOTIVASJON 5
1.3 PROSJEKTEIER OG OPPDRAGSGIVER 6
1.4 PROBLEMBESKRIVELSE OG MÅL 6
1.5 OPPBYGGING AV RAPPORTEN 7

2 PROSJEKTBESKRIVELSE 8

2.1 PRAKTISK BAKGRUNN 8
2.1.1 Tidligere arbeid 8
2.1.2 Initielle krav 8
2.1.3 Initiell løsnings-idé 9
2.2 AVGRENSNINGER 9
2.3 RESSURSER 9
2.4 KOSTNADSANALYSE AV WINDOWS-SERVERE 10

3 DESIGN AV PROSJEKTET 15

3.1 FORSLAG TIL LØSNING 15
3.1.1 Alternativ løsning 1 – React, Amazon API Gateway, AWS Lambda 15
3.1.2 Alternativ løsning 2 - Instance Scheduler 16
3.1.3 Alternativ løsning 3 -Amazon Elastic Beanstalk 16
3.2 KOSTNADSANALYSE AV ALTERNATIVER 17
3.2.1 Analyse av kostnad for alternativ nr. 1 17
3.2.2 Analyse av kostnad for alternativ nr. 2 18
3.3 DISKUSJON AV ALTERNATIVER 19
3.4 VALGT LØSNING 21
3.5 VALG AV VERKTØY 21
3.6 PROSJEKTMETODIKK 23
3.6.1 Utviklingsmetodikk 23
3.6.2 Prosjektplan 24
3.7 RISIKOVURDERING 26
3.8 EVALUERINGSPLAN 28

4 DETALJERT LØSNING 29

4.1 AUTENTISERING 30
4.2 SINGLE SIGN-ON 30
4.2.1 SSO sikkerhetsrisikoer 30
4.2.2 Sikkerhetshåndtering 31
4.3 HVORDAN IMPLEMENTERE MSAL.JS? 33
4.4 ARKITEKTUR 34
4.4.1 Webapplikasjonen 34
4.4.2 Design 34
4.5 API-ET 38
4.6 AUTOMATISERINGSPROSESSEN 39
4.7 LAMBDAFUNKSJONER 42
4.8 STEP FUNCTIONS OG TILSTANDSMASKINER 43
4.9 AWS EVENTBRIDGE 44

5 RESULTATER 46

5.1 EVALUERINGSMETODE 46
5.1.1 Utviklingsmetode 46
5.1.2 Brukertesting 46
5.1.3 Testing av produktet 46
5.2 EVALUERINGSRESULTAT 47
5.3 PROSJEKTETS ØKONOMISKE RESULTAT 48
5.4 ØKONOMISK VURDERING AV IMPLEMENTERT LØSNING 51
5.5 PROSJEKTGJENNOMFØRING 51

6 DISKUSJON 54

7 KONKLUSJON OG VIDERE ARBEID 56

8 REFERANSER 58

9 VEDLEGG 63

Figurliste

Figur 2.1: Månedlig kostnad på severne de tre siste årene *11*

Figur 3.1: Alternativ løsning nr.1 arkitektur *15*

Figur 3.2: Alternativ løsning nr.2 arkitektur *16*

Figur 3.3: Priseksempel over kostnader knyttet til Amplify *17*

Figur 3.4: Kostnadseksempel knyttet til lambdafunksjonskall gjennom HTTP-trafikk *18*

Figur 3.5: Gantt-diagram *25*

Figur 4.1: Arkitektur løsningen *29*

Figur 4.2: Eksempel på en tillatelse i AWS *32*

Figur 4.3: Hvordan implementerer Msal.js biblioteket i applikasjonen *33*

Figur 4.4: Skjermbilde som viser veiledning *36*

Figur 4.5: Skjermbilde med serveroversikt mens serverne kjører *36*

Figur 4.6: Skjermbilde etter stopp knappen er presset, som viser stopprossesen *37*

Figur 4.7: Skjermbilde etter servere er stoppet *37*

Figur 4.8: Viser de ulike endepunktene for APIet *38*

Figur 4.9: Tilstandsmaskinene for stopp og start av serverne *44*

Figur 4.10: Konfigurasjonen av tidsplanen som er brukt *45*

Figur 5.1: Figuren illustrerer en oversikt redusering av kostnader etter løsningen er tatt i bruk *49*

Figur 5.2: Kostnad per dag som før løsningen er implementert *50*

Figur 5.3: Kostnad per dag som etter løsningen er implementert *50*

Figur 5.4: Build scriptet til Amplify prosjektet *53*

Tabelliste

Tabell 2.1: Oversikt serverspesifikasjoner, sortert i stigende rekkefølge etter kostnad *13*

Tabell 3.1: Risikoanalysen av risiko nummer 2 ..... *27*

Tabell 3.2: Risikoanalysen av risiko nummer 8 ..... *27*

Tabell 4.1: Oversikt over start- og stopp-tid på serverne *41*

Ordliste

API	Application Programming Interface. Et grensesnitt som gir en koblingen for å utveksle data mellom maskiner og programvare
AWS - Amazon Web Services [1]	Amazon Web Services (AWS) tilbyr skytjenester for databehandling, lagring og nettverk. Disse tjenestene gjør det mulig for brukere å skalere infrastrukturen sin etter behov og betale kun for det de bruker.
AWS Amplify [2]	AWS-verktøy for enklere utvikling av web- og mobilapplikasjoner. Tilbyr hjelp til bygging av sikre og skalerbare applikasjoner.
AWS API Gateway [3]	API Gateway er en administrert tjeneste som gjør det enkelt for utviklere å opprette, distribuere, vedlikeholde og sikre API-er på en skalerbar måte. Det lar deg opprette API-er for tilkobling til serverside-tjenester.
AWS EC2 - Elastic Compute Cloud [4]	EC2-tjenesten fra Amazon Web Services (AWS) tilbyr skalerbare virtuelle servere i skyen. Brukere kan leie disse virtuelle maskinene, kjent som EC2-instanser, og konfigurere dem etter behov.
AWS EventBridge[5]	AWS EventBridge er en administrert tjeneste av AWS som brukere kan sette opp hendelser som utføres når en spesifikk handling skjer. F.eks. bruk av dette er automatisk stopp av EC2 instanser.
AWS Lambda [6]	AWS Lambda er en serverløs databehandlingstjeneste som lar deg kjøre kode uten å måtte administrere servere.

CI/CD pipeline	CI/CD-pipeline (Continuous Integration/Continuous Deployment) er en automatisert prosess for å bygge, teste og distribuere programvare.
Cookies	Cookies er tekstfiler som nettsider lagrer på brukerens enhet for å holde styr på informasjon mellom økter. De brukes ofte for å bevare brukerinnstillinger, innloggingsstatus og handlekurver, noe som forenkler nettbruk.
CSRF [7]	CSRF (Cross-Site Request Forgery) angrep er en type cyberangrep hvor en ondsinnet nettside får en brukers nettleser til å utføre en uønsket handling på et annet nettsted der brukeren er autentisert. Angrepet utnytter tilliten et nettsted har til brukerens nettleser, og gjennomfører handlinger som å endre passord eller overføre midler uten brukerens kunnskap.
DOM	Document Object Model er et plattform- og språknøytralt grensesnitt som gjør det mulig for at programmer og skript kan dynamisk oppdatere innhold og struktur i dokumenter
Elastic Beanstalk (EB) [8]	EB, er en AWS-tjeneste for publisering og administrering av web applikasjoner.
Elastic Load Balancer (ELB) [9]	Et verktøy som fordeler arbeidsflyter. Arbeidsflytene kan bli distribuert til virtuelle servere eller annet oppsett.
Elastiske IP-adresser	En elastisk IP-adresse er en statisk IPv4-adresse tildelt til en AWS EC2-instans i en skyinfrastruktur. Den skiller seg fra vanlige (statisk) IP-adresser ved at den kan tilordnes, frigjøres og omkonfigureres dynamisk til forskjellige EC2-instanser i løpet av driftstiden. Dette gir større fleksibilitet og mulighet for rask gjenoppretting ved feil.
ERP-programvare	ERP-programvare (Enterprise Resource Planning) er et forretningsstyringssystem som tillater organisasjoner å administrere og automatisere ulike forretningsprosesser.

Flerfaktorautentisering	Flerfaktorautentisering er en sikkerhetspraksis som krever to separate autentiseringsfaktorer for å verifisere identiteten til en bruker før de får tilgang til ressursene.
Fullstack	Fullstack er en definisjon på alle teknologilagene i en applikasjon, fra brukergrensesnittet (frontend) til serverlogikk og databehandling (backend).
HTTP-forespørsler	HTTP-forespørsler er klientens meldinger til en server for å få tilgang til spesifikke ressurser på nettet. Disse forespørslene inneholder informasjon om handlingen klienten ønsker å utføre.
IAM (Identity and Access Management)	Er en AWS-tjeneste som hjelper med sikkerhetskontroll av ressurser. Det er muligheter for å lage grupper, brukere og tillatelser for spesifikk tilgang til ressurser.
Infrastruktur som en tjeneste (IaaS, Infrastructure as a Service)	IaaS, eller Infrastruktur som en tjeneste, er en skybasert tjeneste som tilbyr virtuelle maskiner, lagring og nettverk over internett, basert på forbruk. Dette lar brukere kjøre og administrere applikasjoner uten å investere i fysisk infrastruktur, med mulighet til å skalere etter behov.
Microsoft Entra Id [10]	Microsofts skybaserte identitets- og tilgangsstyringstjeneste. Tidligere kjent som Azure Active Directory. Identitetshåndtering for å konfigurere tilgang til tjenester, ressurser til grupper og brukere.
NAT-Gateway	En NAT Gateway (Network Address Translation Gateway) er en administrert tjeneste i en skyinfrastruktur som tillater private nettverkssubnett å få tilgang til Internett mens de opprettholder privat IP-adressering.
React [11]	React er et JavaScript-basert bibliotek for å bygge brukergrensesnitt-komponenter for webapplikasjoner, kjent for sin komponentbaserte tilnærming, bruk av virtuell DOM, og støtte for JSX.
Sesjonskidnapping	Sesjonskidnapping (eng. «Session hijacking») er en type angrep hvor en angriper tilegner seg kontroll over en brukers

	økt (session) etter at brukeren har autentisert seg. Dette kan gjøres ved å stjele eller forfalske en brukers session ID, som ofte lagres i en cookie. Når en angriper har kontroll over denne ID-en, kan de utføre handlinger på vegne av brukeren
Snapshots	Snapshots er statiske øyeblikksbilder av data eller tilstanden til et system på et bestemt tidspunkt. Disse øyeblikksbildene kan brukes til sikkerhetskopiering.
UI	User Interface (brukergrensesnitt), den delen hvor brukeren samhandler med applikasjonen
XSS angrep	XSS (Cross-Site Scripting) angrep er en type sikkerhetsbrudd hvor en angriper injiserer ondsinnet skript i innhold som leveres til en brukers nettleser, vanligvis gjennom en nettside. Disse skriptene kan utføre handlinger på vegne av brukeren, stjele sensitiv informasjon som informasjonskapsler og sesjonsnøkler

1 Innledning

Dette kapittelet forklarer bakgrunnsinformasjon og kontekst til oppgaven. Hvem oppdragsgiver er og motivasjonen til å løse oppgaven. Hva målet er, og problemstillingen vil bli presentert og generell oppbygging av rapporten.

1.1 Kontekst

Ambita benytter i dag et moderne økonomisk regnskapssystem. I samsvar med ulike retningslinjer er Ambita pliktig å beholde informasjon og data fra det tidligere regnskapssystemet. Selskapet er pålagt å oppbevare data og historikk om tidligere kundeforhold over en fastsatt periode i henhold til NOU 2019:9 [12]. Informasjonen er for tiden arkivert i utdaterte systemer, som nå er migrert til virtuelle servere i skyen. Disse serverne kjører hele døgnet, noe som fører til betydelige ressursforbruk og kostnader. Denne arkiveringsplikten strekker seg frem til 1. januar 2025, etter hvilket tidspunkt dataene kan elimineres i henhold til de aktuelle retningslinjene. For å sikre tilgang til data fra de gamle systemene, har Ambita implementert bruk av EC2-instanser fra AWS, som kjører uavbrutt og gir tilgang til data døgnet rundt. Serverne blir sjeldent brukt og det er ikke optimal bruk av ressurser.

1.2 Motivasjon

Regnskapsavdelingen til Ambita må en sjelden gang hente data fra tidligere nevnte servere. Kontinuerlig drift av serverne resulterer i ugunstig utnyttelse av ressurser som energiforbruk og serverkapasitet, inkludert CPU, minne og lagringsplass. Derfor er det av betydelig interesse å optimere kostnadene og ressursbruken. Dette vil ikke bare være økonomisk gunstig for bedriften, men vil også bidra til å redusere miljøavtrykket ved å spare energi.

“Er det mulig å redusere driftskostnadene ved å lage en nettbasert løsning som automatiserer gitte Windows-servere?”

1.3 Prosjekteier og oppdragsgiver

Ambita har alle rettighetene knyttet til sluttresultatet. Løsningen vil være en integrert del av Ambita sine eksisterende AWS-ressurser. Ambita AS ble stiftet i januar 1987 og har hovedkontor i Oslo og en avdeling i Bergen, med omtrent 80 ansatte. Selskapet fokuserer på å utvikle digitale løsninger for eiendomsbransjen. Ambita tilbyr tjenester innenfor eiendomssektoren, slik som digitale kartløsninger og eiendomsinformasjon. Ambita leverer løsninger til ulike sektorer, alt fra offentlige institusjoner til eiendomsutviklere, eiendomsmeglere og privatpersoner.

1.4 Problembeskrivelse og mål

Ambita er lovpålagt å oppbevare historikk fra tidligere kundeforhold frem til 1. januar 2025. For øyeblikket er disse dataene lagret på Windows-servere som kjører på EC2-instanser i skyen. Serverne kjører hele tiden og bidrar til økt kostnad og ressursforbruk. For å kutte utgifter sikter Ambita mot å automatisere driftstidene for serverne, dette er fordi systemet blir sjeldent brukt og det er unødvendig å la de kjøre hele døgnet. Dette må imidlertid balanseres med regnskapsavdelingens behov for tilgang til dataene på serverne.

Ansatte ved regnskapsavdelingen har ikke en bakgrunn innenfor IT derfor er det viktig at webapplikasjonen har et enkelt og intuitivt grensnitt slikt at de enkelt kan starte og stoppe serverne. Dersom de glemmer å stoppe serverne vil dette skje automatisk på ett fast tidspunkt hver dag.

Målet for oppgaven er å utvikle en webapplikasjon som gjennom AWS reduserer kostnadene på serverne, optimaliserer ressursbruk og er en fungerende løsning som Ambita kan ta i bruk. Vedlegget visjonsdokumentasjon beskriver problemstillingen i mer detalj.

Ettersom bakgrunnen for oppgaven er kostnadseffektivisering er det et sentralt mål at selve løsningen klarer å redusere utgifter i forhold til dagens løsning. Det inkluderes derfor en kostnadsanalyse av løsningen.

Dette leder til den aktuelle problemstillingen:

"Hvordan kan en nettbasert applikasjon utvikles for å effektivt automatisere Windows-servere på AWS, slik at kostnader og ressursbruk minimeres?".

1.5 Oppbygging av rapporten

Rapporten er delt opp i 9 deler inkludert vedlegg.

- Kapittel 1 har til hensikt å skissere prosjektets bakgrunn, samt definere mål og problemstillingen det tar for seg.
- Kapittel 2 presenterer en grundigere og mer detaljert beskrivelse av prosjektet.
- Kapittel 3 utforsker mulige alternativer, alternative løsninger og arbeidsmetodikk
- Kapittel 4 tar for seg design og utvikling av prosjektet, beskrevet i detalj.
- Kapittel 5 behandler evalueringen av prosjektet og dets resultater.
- Kapittel 6 diskuterer de oppnådde resultatene og erfaringene fra prosjektet.
- Kapittel 7 konkluderer prosjektet, diskuteres fremtidig arbeid og gir avsluttende tanker.
- Kapitel 8 referanseliste med IEEE format.
- Kapitel 9 list av vedleggs.

2 Prosjektbeskrivelse

Kapittelet beskriver nærmere bakgrunnen til oppgaven og hva den omhandler. Det blir forklart avgrensinger og krav til oppgaven, gruppens initiale løsningside og hvilke ressurser som var til disposisjon. Det er også forklart det økonomiske aspektet ved oppgaven.

2.1 Praktisk bakgrunn

2.1.1 Tidligere arbeid

Tidligere kjørte programmene på fysiske servere i serverhaller hos Ambita. Men gjennom Ambita sin kostnadseffektivisering startet de å bruke infrastruktur som en tjeneste (IaaS) som Amazon Web Services tilbyr. Ambita har migrert sine ressurser og tjenester til skyen gjennom AWS, dette inkluderer også serverne som blir tatt for seg i dette prosjektet.

Serverne omtalt i oppgaven inneholder data fra 3 ulike programmer Axapta [13], Compello [14] og SuperOffice [15]. Axapta er det tidligere navnet på Microsoft Dynamics AX, som er en forretningsadministrasjons- og ERP-programvare (Enterprise Resource Planning). Det brukes for å administrere ulike forretningsprosesser som finans, lagerstyring, produksjon, salg, og service. Compello tilbyr programvareløsninger innen elektronisk fakturahåndtering og andre former for dokumentbehandling. De tilbyr verktøy for fakturahåndtering, elektronisk signatur, og mer for bedrifter. SuperOffice tilbyr CRM-programvare (Customer Relationship Management). CRM-programvare hjelper bedrifter med å administrere og analysere kundeinteraksjoner og data gjennom hele kundeforholdets livssyklus, for å forbedre kundeservice, øke salg og markedsføringseffektivitet.

2.1.2 Initielle krav

Ambita har stilt tre krav til prosjektet: automatisk styring av Windows-servere, et brukervennlig grensesnitt, og nødvendige sikkerhetstiltak. For å realisere automatisert styring, vil det bli utviklet en løsning for regnskapsavdelingen som enkelt starter og stopper Windows-servere på AWS-plattformen. Dette tiltaket sikter mot å optimalisere ressursbruk og redusere kostnader. Med tanke på at regnskapsavdelingen vil bruke grensesnittet, er det sentralt at det er intuitivt og brukervennlig.

Sikkerhet er et fundamentalt krav i prosjektet. Sikker oppstart av serverne er direkte koblet til Ambitas krav om nødvendig sikkerhet. Serverne må starte i en bestemt rekkefølge for å oppnå normal funksjonalitet. Videre understreker Ambita viktigheten av å loggføre aktiviteter i grensesnittet for å sikre sporbarhet og etterrettelighet. I tillegg undersøke, vurdere og eventuelt utvikle andre tiltak mot andre sikkerhetsrisikoer enn hva Ambita har beskrevet.

Bakgrunnen for oppgaven er å spare Ambita for unødvendige kostnader, derfor er det et sentralt krav at løsningen klarer å redusere kostnader og ressurser.

2.1.3 Initiell løsnings-idé

Problemstillingen kan løses på flere ulike måter. Løsningen må ha et brukergrensesnitt som lar brukeren starte og stoppe utvalgte servere. Den må ha et innloggingssystem som kun gir tilgang til de som skal ha tilgang. For å kunne starte og stoppe serverne er det behov for å bruke en tjeneste som gjør det mulig å starte serverne gjennom brukergrensesnittet. Brukergrensesnittet burde være enkel og intuitiv.

2.2 Avgrensninger

Det er gjort avgrensinger for prosjektet. Ambita har spesifisert i oppgaven at det gjelder ni Windows-servere på AWS plattformen, løsningen er nødt til å utvikles med tanke på å kun fungere for disse Windows-serverne. Selv om integrasjon med andre servertyper kan være teknisk gjennomførbart ble det valgt i dette prosjektet å ikke utforske det.

I gjennomføringen av prosjektet har Ambita gitt friheten til å velge de mest hensiktsmessige programmeringsspråkene, rammeverk og teknologiske verktøy. For å sikre sømløs integrasjon, må løsningen innlemmes i deres system uten ekstra arbeid. Det ble besluttet å bruke teknologier, rammeverk og programmeringsspråk som er kompatible med Ambita sine eksisterende ressurser.

2.3 Ressurser

I utviklingen av prosjektet er det blitt inkorporert flere teknologiske ressurser. Blant disse er flere biblioteker knyttet til React-rammeverket, som utgjør en sentral del i prosjektet. Bruk av disse bibliotekene gir oss fleksibilitet, økt ytelse og tilgang til funksjonalitet. Ressursene som brukes, blir forklart under.

Plattformen som prosjektet utvikles på er AWS Amplify. Amplify er en utviklingsplattform som gjør det enkelt å utvikle webapplikasjoner på både klient- og serversiden. Det tilbyr integrasjoner

og støtte for alle tjenestene som AWS tilbyr. Denne teknologien bidrar til en optimalisering av prosessen uten at det kreves en omfattende involvering i den underliggende infrastrukturen.

Material UI

Material UI [16] er et åpent bibliotek for kildekode. Biblioteket tilbyr et stort utvalg av gjenbrukbare komponenter som skjemaer, knapper, ikoner, navigasjonsfelt og layouts og mye mer.

Amplify UI

Amplify UI [17] er et bibliotek som er en del av Amplify-plattformen. React-komponentene her er lett å innlemme med AWS-tjenestene som ble brukt.

Microsoft Authentication Library for JavaScript

Et annet bibliotek som er brukt er Microsoft Authentication Library for JavaScript (MSAL.js) [18]. MSAL.js gjør at en enkelt kan implementere autentisering og autorisasjon med Microsofts sin identitetsplattform i JavaScript-applikasjoner. Ambita har en «hybridinfrastruktur». De bruker AWS-infrastruktur for skytjenester, men bruker også Microsoft tjenester for brukerkontoadministrering og Office-programmer. Ambita bruker Entra ID tidligere kalt Azure Active Directory. Det er nyttig å bruke MSAL.js biblioteket til å autentisere og autorisere Ambits brukerkontoer gjennom Entra ID til vår applikasjon.

2.4 Kostnadsanalyse av Windows-servere

I dette kapittelet vil det bli forklart og vist kostnader knyttet til Windows-serverne som oppgaven omhandler. Oppgaven går ut på å redusere kostnaden til serverne, det er da viktig å forstå og sammenligne de økonomiske aspektene blant disse, slik at en kan komme opp med gode alternative løsninger som effektivt reduserer kostnaden.

Service	Service total	December 2021	January 2022	February 2022	March 2022	April 2022	May 2022	June 2022	July 2022	August 2022	September 2022	October 2022	November 2022	December
Total costs	$48,023.25	$478.41	$1,798.72	$1,699.48	$1,769.45	$1,810.45	$1,960.01	$1,929.56	$1,958.10	$1,959.00	$1,930.01	$1,958.07	$1,920.42	$1,936.42
EC2-Other	$26,047.63	$279.32	$1,065.49	$1,037.05	$1,036.21	$1,033.64	$1,043.67	$1,042.43	$1,041.87	$1,042.45	$1,043.16	$1,041.95	$1,033.50	$1,019.93
EC2-Instances	$21,906.44	$198.44	$730.83	$660.03	$730.83	$774.41	$913.94	$884.74	$913.83	$914.16	$884.45	$913.73	$884.52	$914.08
CloudWatch	$60.65	$0.65	$2.40	$2.40	$2.40	$2.40	$2.40	$2.40	$2.40	$2.40	$2.40	$2.40	$2.40	$2.40
Inspector	$8.53	-	-	-	-	-	-	-	-	-	-	-	-	-

Figur 2.1 - Månedlig kostnad på serverne de tre siste årene

Serversystemet ble satt opp i desember 2021. Kostnadsanalysen viser kostnaden de tre siste årene fra nåværende tidspunkt. Kostnadsgrafen viser kostnadene per måned, i dollar. Det er verdt å nevne at Ambita har gitt tillatelse for deling av disse kostnadene.

Se Figur 2.1, som er laget ved hjelp av AWS Cost Explorer [19]. Her viser grafen den gjennomsnittlige kostnaden på 1847,05$ per måned siden desember 2021. I dagens (28.02.24) kronekurs tilsvarer dette omtrent 20 000 kroner i måneden [20]. Disse kostnadene er fordelt på ni ulike servere som kjører tre ulike programmer, Axapta, Compello og SuperOffice.

AWS har en betal etter forbruk (eng: «pay as you go») prismodell [21]. Det vil si at du betaler kun for de ressursene og tjenestene du bruker uten noen forhåndsbetaling eller faste kostnader. I dette tilfellet er det mulig å redusere serverkostnadene betraktelig etter det er blitt utviklet en automatisert løsning som slår av disse når de ikke brukes.

Kostnadene er også delt opp i ulike kategorier, inkludert EC2-instanser, EC2-Other, CloudWatch [22] og Inspector [23].

Kostnadene knyttet til kategorien «EC2-instanser», omfatter kostnader direkte knyttet til kjøring, CPU og minne. Det er hovedsakelig disse kostnadene som blir redusert ved å stoppe serverne. Se tabell 2.1 for eksakte priser for serverne.

Innenfor EC2-Other kategorien er det kostnader knyttet til Elastic Block Store-volumer og snapshots, data-overførings kostnader, elastiske IP-adresser og NAT-Gateway. EBS-volumer er virtuelle harddisker knyttet til EC2-instanser [24]. Snapshots er sikkerhetskopier av EBS-volumer [25]. Disse sikrer data og replikerer volumer, slik at en alltid har sikkerhetskopier av serverne.

CloudWatch omfatter kostnader knyttet til overvåkning, logging og varsler. CloudWatch gir oss muligheten til å overvåke ytelsen til serverne i sanntid og gjør at det er lettere å oppdage eventuelle problemer knyttet til serverne.

Inspector-tjenesten gir en sikkerhets- og sårbarhetsvurdering og gir Ambita tilbakemeldinger hvis det er noe som må utbedres på sikkerhet og oppsett.

Disse kategoriene er nødvendige å ha for å ha et sikkert og pålitelig system i skyen.

Når en setter opp EC2-instansene, har en mulighet til å velge spesifikasjoner for disse instansene, inkludert prosessorkraft, minne, lagringskapasitet og nettverksytelse. Tabell 2.1 viser spesifikasjonene og kostnadene til hver enkelt server.

Tabell 2.1 - Oversikt serverspesifikasjoner, sortert i stigende rekkefølge etter kostnad

Server	Servertype	CPU	Minne	Kostnad per time ($)
1 – (Axapta)	t2.small	1	2GB	$0,036
2 – (SuperOffice)	t2.small	1	2GB	$0,036
3 – (Compello)	t3a.medium	2	4GB	$0,0616
4 – (SuperOffice)	t2.medium	2	4GB	$0,0716
5 – (Axapta)	t2.medium	2	4GB	$0,0716
6 – (Axapta)	t3a.large	2	8GB	$0,114
7 – (SuperOffice)	t3.large	2	8GB	$0,1236
8 – (Axapta)	t3a.xlarge	4	16GB	$0,2464
9 – (Axapta)	r5a.xlarge	4	32GB	$0,458
Total kostnad per time				**$1,2188**
Total kostnad per dag: (24 * kostnad per time)				**$29,2512**
Total kostnad per måned: (30 * kostnad per dag)				**$877,536**

Serverne påfører Ambita for store unødvendige utgifter siden dette systemet brukes sjeldent, maksimalt 3 ganger i måneden. Dette er en stor ineffektivitet i ressursbruken.

Som vist i tabell 2.1, ligger den månedlige kostnaden på $877,536 for kategorien «EC2-instances». Dette er kostnaden som kun er knyttet til datakraft, altså CPU og minne. Dette koster det å ha serverne oppe å kjøre uten noe form for kommunikasjon eller applikasjoner kjørende på disse. Som tidligere beskrevet, kommer kostnader knyttet til EC2-Other oppå denne kostnaden.

Gjennom denne analysen, er det avdekket kostnader siden desember 2021. Den gjennomsnittlige månedlige kostnaden har vært $1847,05, tilsvarende omtrent 20 000 kroner ved dagens kronekurs. Totalt har disse kostet Ambita $48023,25, tilsvarende 507 000 kroner. Disse kostnadene er fordelt på ni ulike servere som kjører tre ulike programmer: Axapta, SuperOffice og Compello.

Store deler av kostnadene kommer av kjøretiden til serverne og tjenestene som er knyttet til disse. Det er tydelig at det er rom for kostnadsredusering, spesielt med tanke på den sjeldne bruken og de kontinuerlige driftskostnadene. Løsningen søker etter å hovedsakelig redusere kjøretiden til serverne til kun ved behov.

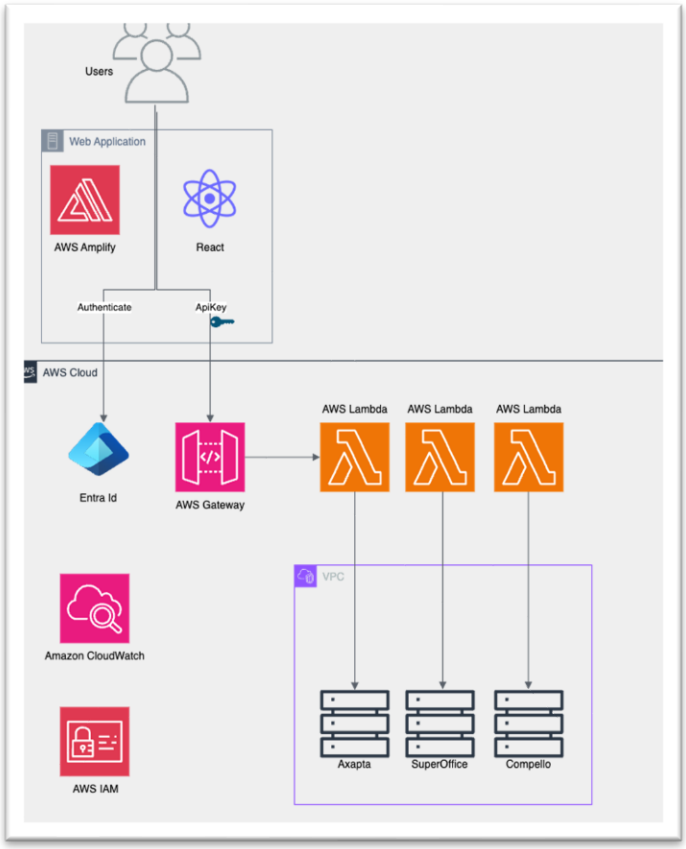
3 Design av prosjektet

Kapittel 3 er en orientering på hvordan planleggingen av prosjektet er blitt gjennomført. Dette innebærer alternative løsninger, diskusjoner av alternativ og hva som er valgt som løsning. Det er også fokus på utviklingsmetode, hvilke verktøy som er brukt under prosjektet og om det økonomiske aspektet. Risikovurdering og evalueringsplan er også inkludert.

3.1 Forslag til løsning

3.1.1 Alternativ løsning 1 – React, Amazon API Gateway, AWS Lambda

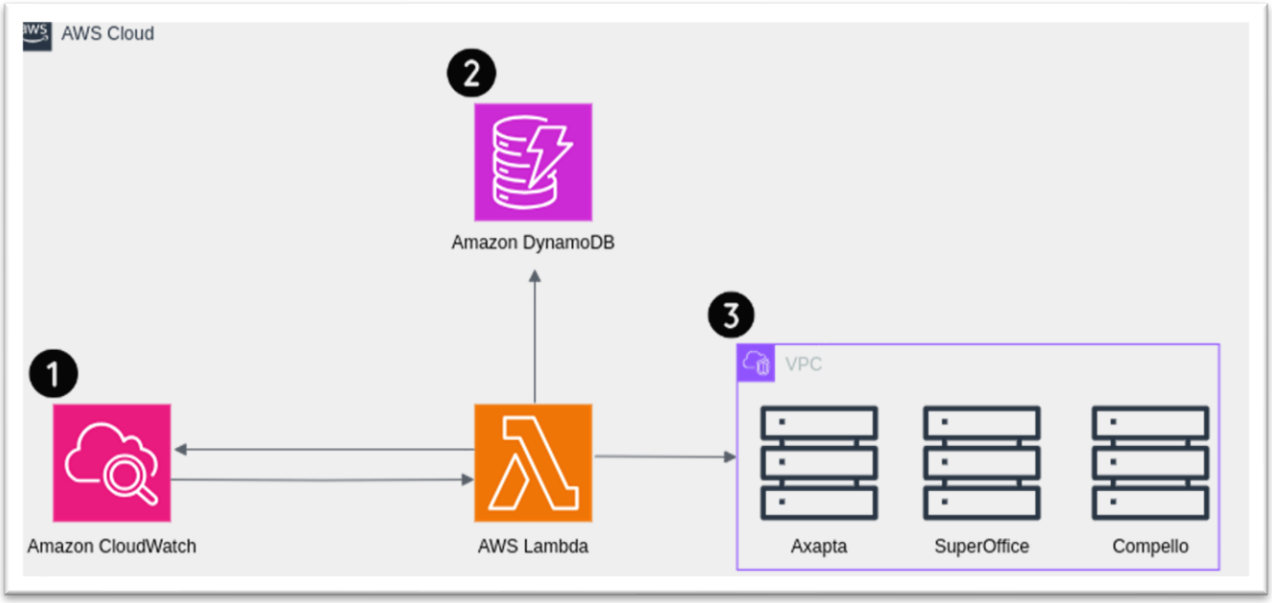
Alternativ nr. 1 er å lage en brukervennlig webapplikasjon med React som kaller ulike funksjoner (AWS Lambda) gjennom å lage et API, (AWS API Gateway). Når en funksjon blir kalt, starter/stopper disse serverne, avhengig om de kjører eller ikke. Applikasjonen blir utviklet på AWS Amplify plattformen som er en serverløs tjeneste. Figur 3.1 illustrer arkitekturen til alternativ nr. 1.



Figur 3.1 - Alternativ løsning nr.1 arkitektur

3.1.2 Alternativ løsning 2 - Instance Scheduler

Denne løsningen automatiserer EC2-instanser. Den starter og stopper instansene automatisk når de ikke er i bruk. Løsningen fungerer ved å sette opp tidsplaner gjennom CloudWatch Scheduler, når en vil at disse instansene skal være oppe. Eksempel på en tidsplan kan være mandag til fredag, kl 08:00 – 16:00, hver uke.



Figur 3.2 - Alternativ løsning nr.2 arkitektur

Denne løsningen lager en CloudWatch-hendelse som kaller på en «Instance Scheduler», en lambda-funksjon. I konfigurasjonen defineres en tilpasset «tag» som «Instance Scheduler» vil bruke til å assosiere de instansene som skal startes og stoppes. Konfigurasjonen blir lagret i Amazon DynamoDB [26] og lambda-funksjonen henter denne konfigurasjonen hver gang den kjøres. En kan da «tagge» de instansene man vil skal startes og stoppes hver gang i den gitte tidsplanen som er definert. Et eksempel på mulig arkitektur av denne løsningen er visualisert i figur 3.2.

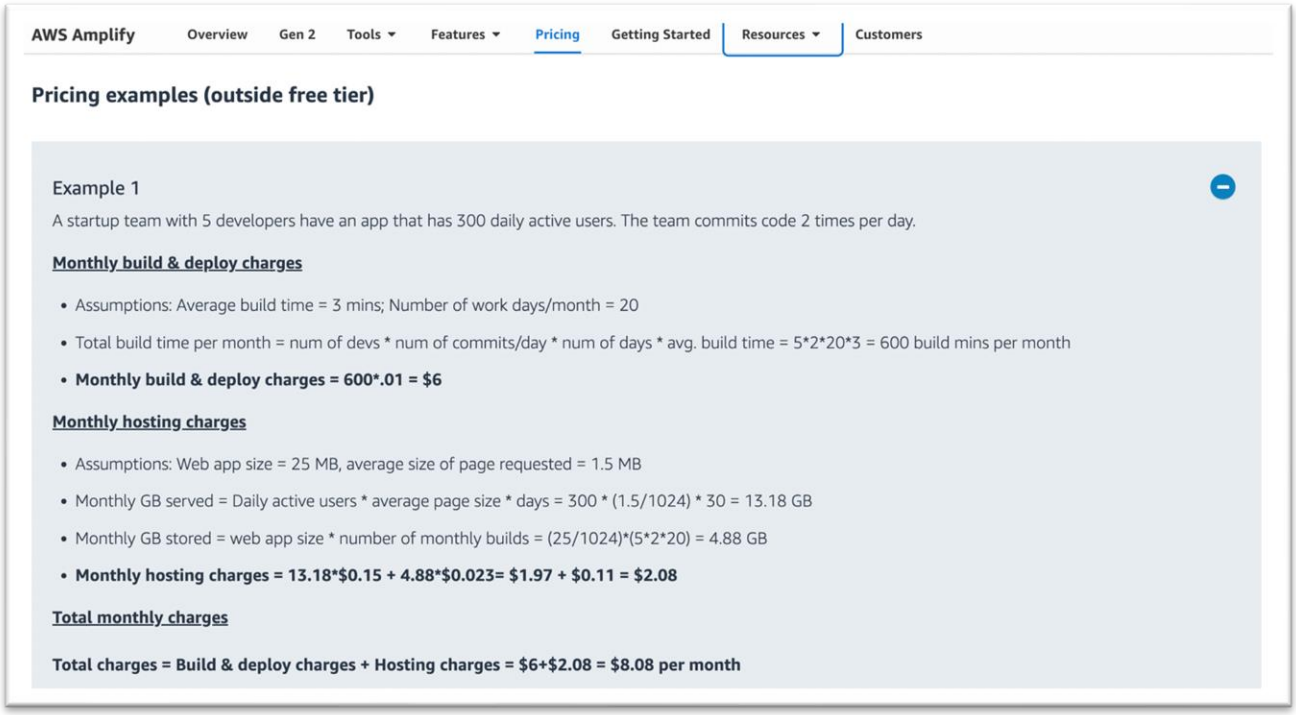
3.1.3 Alternativ løsning 3 -Amazon Elastic Beanstalk

Elastic Beanstalk og Amplify er begge AWS-tjenester som gir mulighet til å publisere og utvikle applikasjoner, og deler mange av de samme grunnprinsippene. Forskjellen ligger i at Elastic Beanstalk legger til rette for mer kontroll over hvilke tjenester og underliggende infrastruktur applikasjonen blir bygd på. I dette tilfellet vil denne løsningen være relativt lik som alternativ nr. 1 men med unntak av alt oppsett av konfigurasjon og infrastruktur som må gjøres manuelt.

3.2 Kostnadsanalyse av alternativer

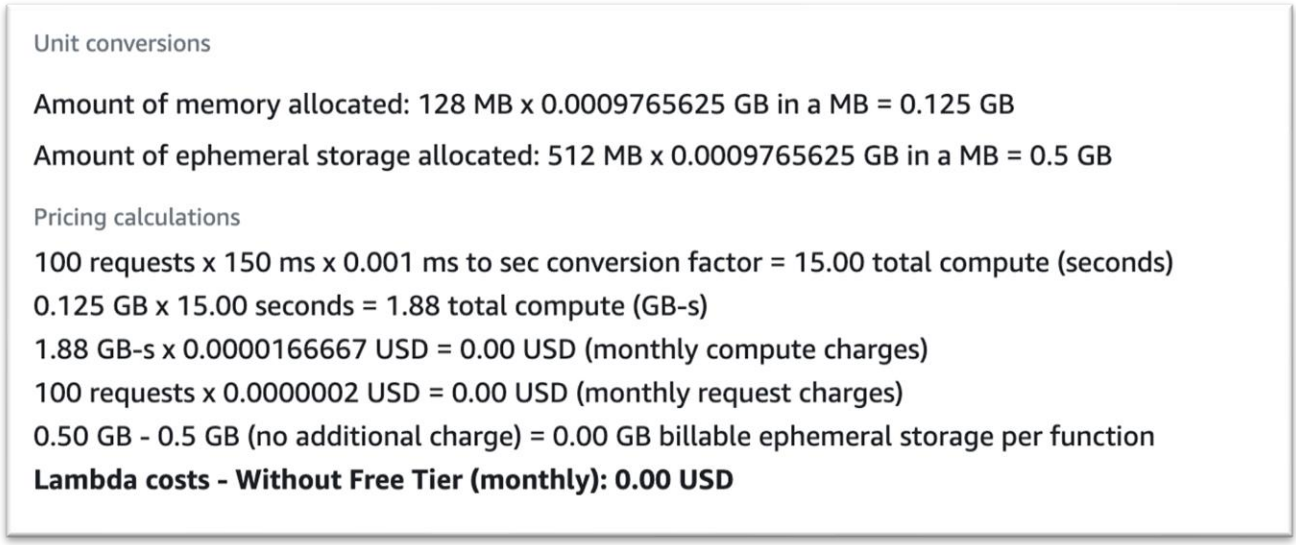
3.2.1 Analyse av kostnad for alternativ nr. 1

For alternativ nr. 1 er det en vesentlig utfordring å estimere kostnadene og i hvor stor grad kostnadsreduksjon kan oppnås. Figur 3.3 er et eksempel som er hentet fra AWS Amplify. Dette viser kostnad knyttet til det å bygge og distribuere applikasjonen samt for publisering av applikasjonen. Eksempelet fra Figur 3.3 er det som best samsvarer med behovene og kriteriene til oppgaven.



Figur 3.3 – Priseksempel over kostnader knyttet til Amplify

Denne løsningen vil redusere kostnadene knyttet til kategorien «EC2-instances» kategorien med mer enn 76%, se neste kapittel 3.2.2. Grunnen for dette er at denne løsningen gjør at serverne er oppe bare når brukeren klikker på start/stopp-knappen.



Figur 3.4 – Kostnadseksempel knyttet til lambdafunksjonskall gjennom HTTP-trafikk

Videre kommer også kostnader knyttet til kall av lambda-funksjoner gjennom HTTP-forespørsler. Figur 3.4 viser et eksempel på kostnad av kjøringen av lambda-funksjoner.

I priskalkulatoren AWS tilbyr [27] er det tatt utgangspunkt i et eksempel med 100 forespørsler i måneden som illustrasjon. Det er viktig å anerkjenne at antallet forespørsler sannsynligvis er mye lavere i virkeligheten. Eksempelet viser at i dette tilfellet vil lambdafunksjonene koste 0.00$. Kostnadene tilhørende lambda-funksjoner er ubetydelige, men det ble besluttet å ta med dette på grunn av at lambdafunksjonene er en viktig del av funksjonaliteten av applikasjonen.

3.2.2 Analyse av kostnad for alternativ nr. 2

I alternativ nr. 2, Instance Scheduler [28], er det utarbeidet en tidsplan i samsvar med vanlige arbeidstider 08:00 – 16:00. Dette valget er basert på bedriftens vanlige arbeidstider og serverne ville da være tilgjengelig i dette tidsrommet. Tidsplanen kan selvsagt endres i henhold til arbeidstidene til de enkelte i regnskapsavdelingen, men det er tatt utgangspunkt i vanlig arbeidstid.

I dag kjører serverne hele døgnet, dette tilsvarer 168 timer i uken. Ved bruk av tidsplaner vil serverne bare kjøre periodevis.

For å vise et eksempel for effektiviteten av dette blir perioden angitt fra 08:00 til 16:00 i hverdager som er innenfor den normale arbeidstiden for ansatte hos bedriften. Med mulighet for å enkelt tilpasse om denne perioden trengs å utvides eller reduseres. Etter tidsplanen vil serverne kjøre totalt 40 timer i uken. Om dette timetallet sammenlignes med dagens løsning gir det dette uttrykket:

$$\frac{40\ timer}{168\ timer} \rightarrow\ 0.238 \rightarrow\ 23.8\%$$

Dette betyr at serverne kjører omtrent 24% av den opprinnelige tiden. Ved å velge alternativ nr. 2 kunne det potensielt ha redusert kostnadskategorien «EC2-instances» med 76%. Dette på grunn av AWS sin betal etter forbruk prismodell. Denne prismodellen vil si at man betaler kun for serverne når de er i bruk, som vil si innenfor denne gitte tidsplanen, ellers vil en kun betale for utgiftene for å ta vare på den lagrede informasjonen på serverne, altså kostnaden knyttet til kategorien «EC2-Other».

3.3 Diskusjon av alternativer

Amazon Web Services tilbyr over 200+ tjenester. Den store variasjonen gjør det mulig å løse problemer på flere ulike måter. Det ble besluttet å prioritere de mest relevante og effektive løsningene som tilfredsstiller behovene til oppgaven, og som samtidig ikke overkompliserer løsningen.

Alternativ nr. 1 er en løsning hvor flere av ressursene og tjenestene som blir brukt automatisk blir konfigurert gjennom oppsett i Amplify. Amplify er en god løsning til raskt å bygge webapplikasjoner. Med denne løsningen fokuseres det på å utvikle applikasjonen uten å måtte bekymre seg for den underliggende infrastrukturen. Dette sparer tiden som blir brukt på utvikling og gir muligheten for å effektivt implementere nye funksjoner raskere. Alternativ nr. 1 er også det rimeligste alternativet blant de ulike alternativene. Dette fordi løsningen optimaliserer tiden serverne kjører til bare når det er behov for det. Den ansatte logger da inn på webapplikasjonen og skrur av og på disse når en trenger tilgang til systemene. På denne måten blir tiden serverne er oppe minimalisert.

AWS Lambda er en serverløs tjeneste som tillater kjøring av kode uten å måtte administrere servere. En serverløs tjeneste, gir muligheten til å kjøre kode uten behov for administrasjon av serverinfrastruktur. Dette gir muligheten til å fokusere på å skrive koden, mens plattformen tar seg av oppskalering, nedskalering og overvåking automatisk. Dette forenkler utviklingsprosessen. Lambda fungerer ved å utføre koden din i respons av hendelser. Disse hendelsene kan være alt fra HTTP-forespørsler via API, til tidsplanlagte hendelser.

Amplify tilbyr støtte for kontinuerlig integrasjon og kontinuerlig leveranse (CI/CD) automasjonsflyt. Amplify gir muligheten for å koble applikasjonskoden til en versjonskontrolltjeneste som GitHub [29] og kan da konfigurere en automasjonsflyt som automatisk tester, bygger og distribuerer endringene i kodebasen.

Alternativ nr. 2, i denne løsningen foregår alt i bakgrunnen. Det vil si at det er ingen brukergrensesnitt eller applikasjon hvor serverne blir styrt fra. Alt blir tatt hånd om av AWS. Løsningen går ut på å sette opp tidsplaner for når tid disse serverne skal kjøre. Denne løsningen har som sagt ingen brukergrensesnitt som er et av kravene i oppgaven. Grunnen for å diskutere dette alternativet er at det reduserer kostnad og er en enkel løsning uten noen form for interaksjon. Denne løsningen ville ha redusert kostnadene med opptil 76%. En behøver ikke å tenke på at en må logge inn på tjenesten og trykke på knapper for å starte serverne før man skal bruke programmene. Løsningen reduserer også muligheten for brukerfeil. I dette løsningsalternativet er serverne oppe gjennom hele arbeidstiden til bedriften, (08:00 – 16:00).

Alternativ nr. 3, Elastic Beanstalk (EB) er en serverbasert AWS løsning for utrulling av applikasjoner. Når du publiserer applikasjonen gjennom EB legger tjenesten opp til hvilke AWS-tjenester som er nødvendig eller relevante for din applikasjon. Tjenesten tilbyr forhåndsdefinerte maler for konfigurering av disse tjenestene, men gir også full tilgang for manuell kontroll av oppsett. Det som gjør EB til en relevant løsning er at den fjerner en del av kompleksiteten bak infrastrukturen til applikasjonen. EB setter opp ELB (Elastic Load Balancer) for applikasjonen, tilrettelegger for skalering av applikasjonen og vil fylle på med nødvendige ressurser som flere EC2 instanser etter forespørsel. ELB distribuerer arbeidsflyter mellom flere beregningsressurser, noe som bidrar til bedre tilgjengelighet og feiltoleranse. Trenger applikasjonen databaser, forslår EB relevante databasetjenester. Noe av det mest fordelaktige med tjenesten er at den setter opp nødvendige IAM (Identity and Access Management) [30] roller for applikasjonen slik at den får tilgang til nødvendige operasjoner mellom AWS tjenester og systemer. Det som kan være negativt er at det gjerne tar lenger tid å sette opp EB som vert for applikasjonen på grunn av en lenger konfigureringsprosess. Konfigureringsprosessen forutsetter også mer utbredt kunnskap om hvilke tjenester og forbindelser som er nødvendig for applikasjonen. En annen negativ side er at applikasjonen bruker serverbaserte løsninger, noe som kan føre til ekstra kostnader.

3.4 Valgt løsning

Etter en grundig evaluering av de ulike alternativene og en detaljert gjennomgang, ble alternativ nr. 1 valgt for å løse oppgaven. Konklusjonen er at alternativ nr.1 er godt egnet for oppgaven og den er gjennomførbar. Denne løsningen reduserer kostnadene mest. Løsningen innebærer å utvikle en webapplikasjon som gjennom et enkelt og intuitivt brukergrensesnitt automatiserer Windows-serverne til Ambitas tidligere programmer Axapta, Compello og SuperOffice. React ble valgt fordi det var av interesse å utbre kunnskapen om rammeverk for klientsideutvikling. React er et av de mest populære rammeverkene for klientsideutvikling for øyeblikket [31], noe som gjorde det til et sentralt alternativ.

3.5 Valg av verktøy

Rapportskriving

Teamet delte Word-tekstdokumenter, Excel-regneark og andre filer gjennom Microsoft 365 - plattformen [32]. Hovedårsakene til dette er muligheten til å synkronisere arbeid på tvers av filer i sanntid gjennom OneDrive, samt muligheten for medlemmer til å se endringer gjort av andre i sanntid.

Prosjekt plan

Gjennom prosjektet ble verktøyet Trello [33] brukt for å holde styr på kommende oppgaver, aktive oppgaver, ferdige oppgaver og ulike frister for innlevering. Trello er en nettside som tilbyr en kanban stil liste over alle oppgavene som er med i et prosjekt. Dette har fungert som en visuell representasjon av prosjektets fremdrift.

Kommunikasjon

Det ble benyttet plattformen Messenger [34] hvor det ble laget en egen kommunikasjonskanal internt i gruppen. Messenger ble benyttet før vi hadde fått brukere på Ambita sine kommunikasjonskanaler. Ambita disponerte Outlook-mail [35] til gruppemedlemmene samt annet innloggingsinformasjon slik at gruppemedlemmene fikk tilgang til ressursene deres. Slack [36] var brukt som kommunikasjonskanal for å kommunisere med ansatte i bedriften ettersom dette var deres foretrukne kommunikasjonskanal.

Design

Det var behov for å lage ulike skisser og trådrammer (eng. «wireframes») for designet av ferdig produkt. Figma [37] er ett designverktøy for å designe skissene og trådrammene. Valget av Figma falt naturlig etter tidligere erfaring.

Utviklingsmiljø

Ambita gav ingen krav til valg av all teknologi, kodespråk og rammeverk. Teksteditoren Visual Studio Code [38] er kjent fra studietiden, det falt naturlig å velge det.

Kodespråk

JavaScript [39] er et programmeringsspråk som primært brukes til dynamisk skripting på klientsiden av nettsider, men kan også brukes på serversiden gjennom verktøy som Node.js. JavaScript brukes først og fremst i nettlesere for å manipulere nettinnhold gjennom DOM. JSON [40] er basert på en underkategori av Javascript og er brukt som ett datautvekslingsspråk. JSON er brukt til å utveksle data mellom serverside og klientside.

Det ble brukt Python 3.8 [41] for å skrive funksjoner i AWS lambda i serversiden. Årsaken til at en eldre versjon som 3.8 er brukt er fordi den var støttet av de forskjellige AWS tjenestene og var mer stabil. Dette sparte oss for kompatibilitetsproblemer.

React

React er et JavaScript-bibliotek for å bygge brukergrensesnitt for nett- og mobilapplikasjoner som er utviklet av Meta. Det fokuserer på modulær utvikling av gjenbrukbare komponenter. React bruker Virtual DOM for å forbedre ytelsen ved oppdatering av grensesnittet. Det har et stort fellesskap og er åpen kildekode.

Versjonskontroll

For å gjøre det enklere for flere personer å jobbe med samme prosjekt uten å overskrive hverandre, og for å få en versjonshistorikk, ble det valgt versjonskontrollsystemet Git [42]. All kode som ble laget i prosjektperioden ble lagt ut på Ambita sitt kodemiljø.

Kunstig intelligens

Gjennom prosjektet har OpenAIs ChatGPT [43] blitt brukt for inspirasjon og feilsøking i kode. Kunstig intelligens er kun brukt som ett hjelpemiddel til læring og kunnskap.

Kontrast og farge

Ambita har en oversikt over hvilke farger som er inkludert i deres visuelle profil [44]. Inkludert i samme nettside har Ambita en design håndbok som informerer om de ulike retningslinjene for kontrast og lesbarhet av hensyn til universell utforming [45]. I tillegg informerer denne håndboken om annet som er relevant for design. Ambita anbefaler også å bruke det digitale verktøyet snook.ca [46] for å sjekke at nettsiden oppnår kravene for kontrast og lesbarhet ifølge WCAG 2.1 [47] prinsippene.

Fysiske ressurser

Oppdragsgiverne har stilt kontorlokalene til dispensasjon for prosjektgruppen. Grunnet sikkerhetsmessige årsaker knyttet til at prosjektet omhandler Ambita sine servere, ble applikasjonen utviklet på Ambita sine PCer. Gjennom perioden har det vært muligheter til å spørre utviklerne ved bedriften om det er noe som måtte avklares med oppgaven eller løsningsidéen.

3.6 Prosjektmetodikk

Dette kapittelet tar for seg utviklingsmetodikk brukt gjennom prosjektet, prosjektplanlegging og risikovurdering. Det er ikke brukt én spesifikk utviklingsmetodikk for prosjektet, men en hybrid tilnærming av Scrumban [48]. Prosjektplanleggingen er gjennomført av gruppen og blir illustrert gjennom ett Gantt-diagram i et underkapittel.

3.6.1 Utviklingsmetodikk

Det ble ikke brukt én enkelt utviklingsmetodikk gjennom prosjektet. I stedet ble ulike aspekter fra de agile utviklingsmetodikkene Scrum [49] og Kanban [50] benyttet. Ved å plukke forskjellige elementer fra disse metodikkene, ble prosjektutviklingen tilpasset gruppens behov og preferanser.

Scrum er et administrativt rammeverk som gir struktur for prosjektgjennomføring. Dette oppnås gjennom tildeling av roller, strukturerte sprinter og tett kommunikasjon med kunde og internt i prosjektgruppen. Aspekter fra Scrum som er brukt inkluderer sprinter, sprint retrospektiv og daglige møter.

Kanban er en utviklingsmetodikk som visualiserer arbeidsflyten. Flyten visualiseres gjennom et Kanban-bord. Et Kanban-bord består av kolonner og kort, hvor kortene inneholder arbeidsoppgaver og kolonnene indikerer hvilket stadium i prosessen kortene befinner seg i.

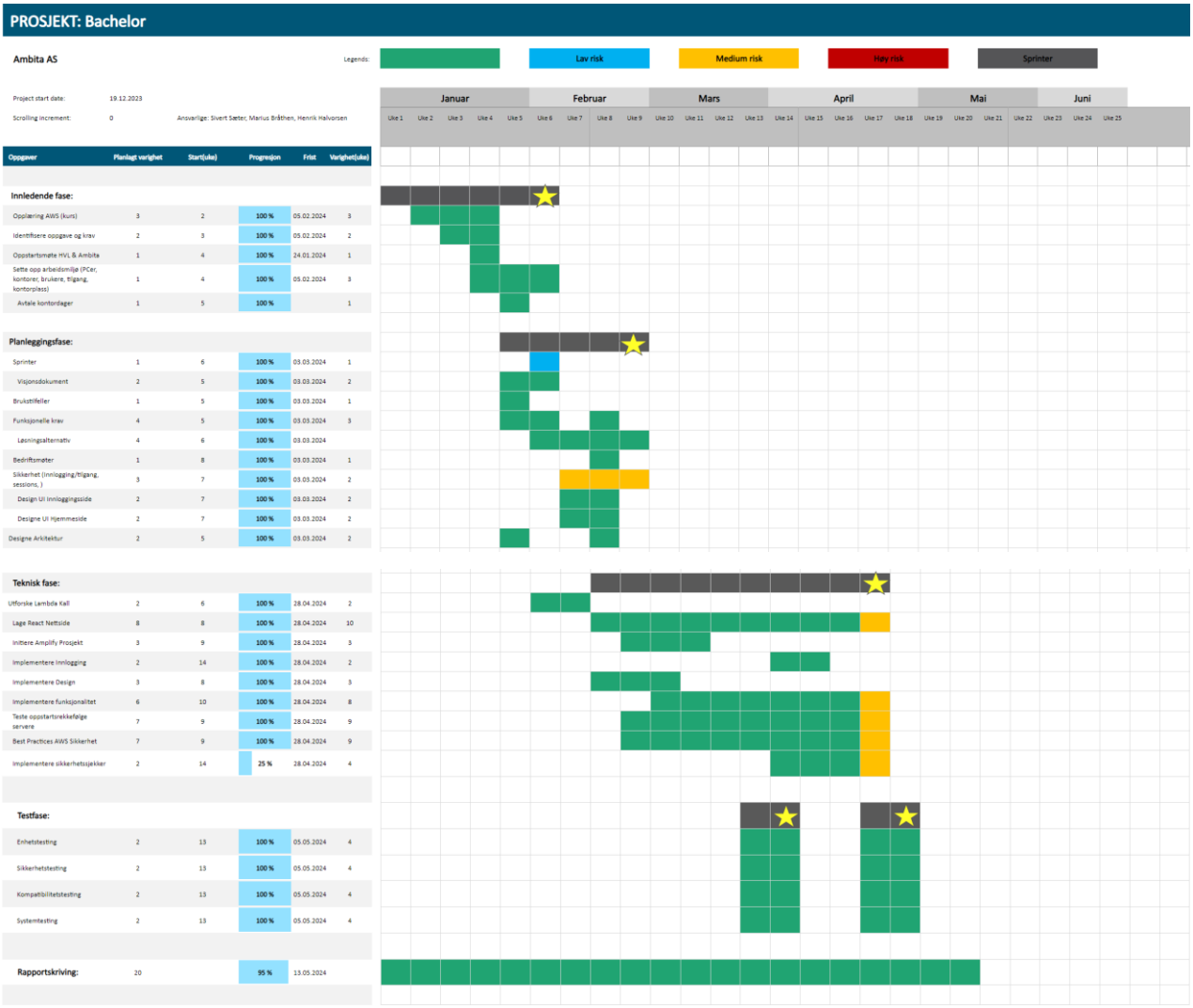
Kolonnene kan være utformet som «Å gjøre», «Under arbeid», «Vurdering» og «Ferdig». Kanban-bordet ble brukt som en del av utviklingsmetodikken.

Ved å integrere elementer fra Scrum og Kanban, ble det skapt en tilpasset utviklingsmetodikk i form av Scrumban. Scrumban har for prosjektet fremmet fleksibilitet, struktur, effektiv kommunikasjon og godt samarbeid med Ambita. Prosessen er delt opp i sprinter, hver med et sett av oppgaver innenfor en spesifisert tidsramme. Disse oppgavene ble organisert på et Kanban-bord. Etter hver sprint er ferdig ble det holdt et møte med Ambita som inkluderte en sprint retrospektiv. Sprint retrospektiv innebærer å reflektere over hva som har blitt oppnådd og identifisere potensielle forbedringer. Disse forbedringene kan relatere seg til selve arbeidet eller måten arbeidet ble utført på. Det ble også avholdt daglige interne gruppemøter for å diskutere gjennomførte oppgaver og fokusområder. Disse gruppemøtene har gjort det mulig for gruppen å analysere hvilke oppgaver som er mest relevant eller pressende, noe som fremmer fleksibilitet i arbeidet.

Valget av utviklingsmetodikk ble tatt på grunnlag av prosjektgruppens preferanser for både den strukturen som Scrum tilbyr, samt fleksibiliteten og den visuelle arbeidsflyten som Kanban gir. Ved å kombinere elementer fra Scrum og Kanban, har prosjektgruppen brukt en tilpasset utviklingsmetodikk. Rollefordelingen fra Scrum ble bevisst utelatt, da gruppen ikke så noen betydelig nytte i den spesifikke rollen som Scrum Master. I stedet har alle gruppemedlemmene delt på ansvaret som tradisjonelt faller under Scrum Master.

3.6.2 Prosjektplan

Planen for prosjektet er illustrert av ett Gantt-diagram. Gantt-diagrammet kartlegger oppgaver og planer gjennom prosjektet. De forskjellige fargekodene brukt i diagrammet er grønn som viser planene utover prosjektet som ikke har noe betydelig risiko, blå, gul og rød beskriver planene våre hvor oppgavene har en risiko og svart viser sprinter. Noen av sprintene overlapper ettersom oppgaver under de forskjellige sprintene ble distribuert i gruppen over samme tid. Gantt-diagrammet er illustrert nedenfor i figur 3.5.



Figur 3.5 – Gantt-diagram

Innledende fase:

Den innledende fasen gikk ut på å bli kjent med bedriften, bacheloroppgaven og sette seg inn i hvordan å utføre et bachelorprosjekt. Her ble AWS utforsket og gruppen tilegnet seg nødvendig kunnskap relatert til oppgaven.

Planleggingsfase:

Planleggingsfasen innebærer arbeidet før det tekniske arbeidet. Dette er design av brukergrensesnittet og arkitekturen til applikasjonen, ønsket funksjonalitet og løsningsalternativ og generelt planleggingen for prosjektet og arbeid.

Teknisk fase:

I den tekniske fasen utføres arbeidet for den tekniske løsningen, som planlagt i tidligere fase. Applikasjonen blir utviklet og satt opp mot Ambita sitt arbeidsmiljø og Amazon sine tjenester i AWS. Ytterlig informasjon om det tekniske aspektet finnes i vedlegget Systemdokumentasjon.

Testfase:

Testfasen er delt opp i to deler. Begge delene inkluderer tester som enhets-, system-, kompatibilitet- og sikkerhetstesting. Årsaken til at det er to deler av denne fasen er fordi under prosjektet har gruppen fått tilgang til en testplattform i form av en "sandbox" konto. Plattformen kunne brukes til å teste AWS og applikasjonen sin funksjonalitet før den ble satt opp på produksjonsplattformen til Ambita. Etter at nødvendige tester var gjennomført og godkjent kunne arbeidet migrere over til produksjonsplattformen. Når løsningen er ferdig blir det også gjennomført testing på lik måte som tidligere.

3.7 Risikovurdering

I oppstartsfasen av prosjektet ble det gjennomført en risikoanalyse av prosjektet. Risikofaren ble vurdert ut ifra sannsynligheten for at risikoen skulle påvirke prosjektet, og i hvor stor grad konsekvensen påvirket prosjektets kvalitet. Risikofaren ble visualisert som risikoprodukt av sannsynligheten og konsekvensen.

Hovedsakelig er risikoen ikke spesielt høy for mange av punktene i prosjektet. De to mest betydelige risikofaktorene involverer imidlertid mangel på kunnskap og misforståelser rundt teknologien som skulle benyttes i løsningen. En av grunnene for disse risikoene var gruppemedlemmenes manglende kunnskap til den sentrale teknologien, AWS. For å håndtere dette ble det gjennomført et introduksjonskurs innen AWS, kalt AWS Cloud Practitioner Essentials [51]. Dette er et kurs som alle nye ansatte hos Ambita må fullføre. Kurset tilbød en grundig introduksjon til AWS og gav et solid grunnlag for videre bruk av tjenesten, samt innsikt i de forskjellige tjenestene AWS tilbyr.

For de andre risikoene ble det laget ulike tiltak for å forhindre eller minimere risikoen. For eksempel en annen høy risiko ved prosjektet er at prosjektet blir unødvendig komplisert, på grunn av feil prioritering av funksjonaliteter. Dette vil resultere i et dårligere produkt eller et uferdig produkt. For å adressere denne risikoen, ble det utarbeidet en prioritetsliste over funksjonalitet.

Dette tiltaket er ment for å minimere risikoen for å bruke for mye tid på aspekter som ikke er kritiske for oppgaven. I stedet fokuseres tid og ressurser på de mest kritiske funksjonene først. Dersom det oppstår muligheter for å utvikle mindre kritiske aspekter vil det bli gjort i slutten av prosjektet. Lignende tiltak ble laget for hver av risikoene som ble kartlagt for prosjektet.

Nedenfor blir risikoanalysen av de to mest kritiske risikoene illustrert, se tabell 3.1 og 3.2 nedenfor. Fullstendig risikoanalyse kan finnes i vedlegget Prosjekthåndboken.

Tabell 3.1 – Risikoanalysen av risiko nummer 2

2	Gruppemedlemmene mangler kunnskap kritisk til oppgaven	Bruk av nye teknologier og prinsipper i oppgaven	Høy (3)	Lav (5)	15	I starten av prosjektet ble det gjennomført et introduksjonskurs for AWS og en workshop for React for å gi en start og grunnleggende kunnskap om sentral teknologi.

Tabell 3.2 – Risikoanalysen av risiko nummer 8

8	Overkompliserering av oppgaven	Utvikler unødvendig funksjonalitet	Høy (4)	Høy (4)	16	Sette opp en liste av de mest vitale funksjonene for applikasjonen og begynn med det mest kritiske for at prosjektet oppnår minste kravet for fullføring. Deretter utvikle neste del av funksjonalitet som er satt opp på prioriteringslisten

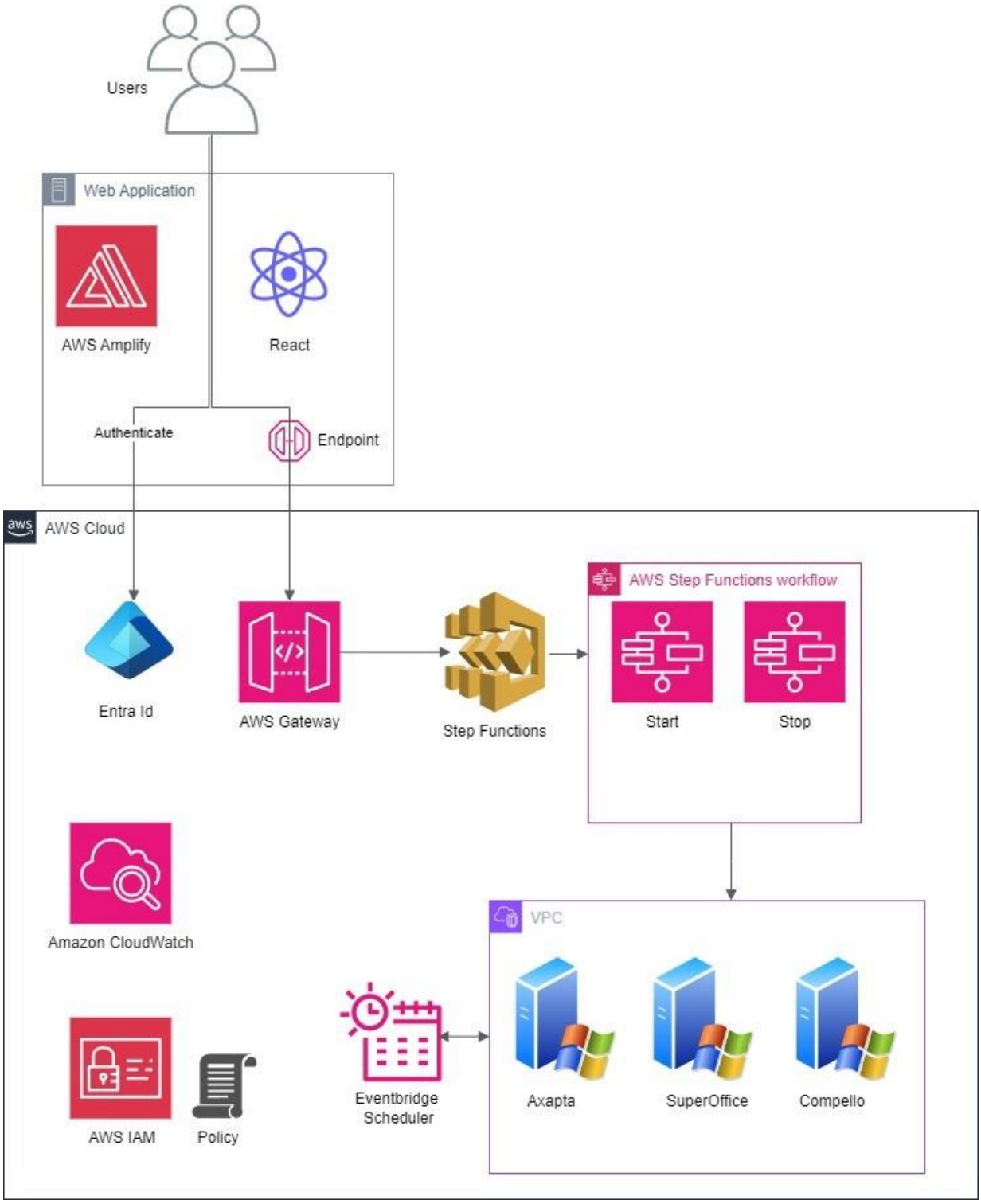
3.8 Evalueringsplan

For å sikre kontinuerlig tilbakemeldinger av brukerne ble det besluttet at i slutten av hver sprint skal det bli gjennomført et møte med arbeidsgiverne for å presentere prosjektets framgang. Her har Ambita muligheten for å komme med sine egne tanker om prosjektets utvikling og for å avklare eventuelle mistolkninger. Tilbakemeldinger fra bedriften og brukerne blir sett på som viktig for å sikre at løsningen dekker behovene og kravene for oppgaven. I tillegg vil det bli gjennomført en del mindre og uhøytidelig møter med ekstern veileder fra Ambita for å sikre at prosjektet er på rett vei.

Utgangspunktet for oppgaven er kostnadsredusering for Ambita. Et stort fokus ligger på evalueringen av oppgavens effektivitet ved å sammenligne serverkostnadene før og etter implementering av løsningen. For å fastslå prosjektets økonomiske effekt, har det vært gjennomført kostnadsanalyser av dagens mangel på løsningen, slik at det er mulig å sammenligne kostnadene med før og etter produktet er tatt i bruk. Kostnader vil bli analysert før og etter utviklingen av produktet er ferdig, slik at når produktet er ferdig er det mulig å da evaluere om løsningen reduserer kostnad og ressursbruk, og om problemstillingen har blitt løst på en god måte.

4 Detaljert løsning

Løsningen som er utviklet for dette prosjektet, er en fullstack-applikasjon hvor klientsidens brukergrensesnitt er utviklet med React, og serversiden er orkestrert gjennom flere tjenester fra AWS. Hvilke AWS-tjenester som benyttes og nærmere detaljer om hvordan de tar del i løsningen forklares i dette kapittelet. En overordnet arkitektur for løsningen som er utviklet vises i figur 4.1.



Figur 4.1 – Arkitektur løsning

4.1 Autentisering

Til autentisering brukes det Microsoft Entra ID, dette på grunn av at Ambita allerede har brukerkontoer på Microsoft plattformen. Entra ID ble tidligere kalt Azure Active Directory. Entra ID muliggjør identitetshåndtering for å administrere tilgang til applikasjoner og ressurser. Applikasjonen er bygget på React og Microsoft har flere ulike biblioteker tilpasset React-rammeverket. Biblioteket heter Msal.js, dette er Microsofts autentiseringsbibliotek for javascriptapplikasjoner. Dette kan brukes til autentisering på klient- og server-siden. I løsningen foregår autentiseringen kun på serversiden.

4.2 Single Sign-On

I dagens teknologiverden, har man mange kontoer og brukere på ulike applikasjoner og det er vanskelig å huske alle brukernavn og passord man bruker i hverdagen. Single Sign-On (SSO) er en effektiv løsning på dette. SSO er en brukerautentiseringsprosess som tillater bruker å få tilgang til flere applikasjoner eller systemer med kun ett sett av påloggingsdetaljer [52]. Det betyr at hvis en bruker har logget inn på et system en gang, vil systemet huske brukeren som er innlogget og vil deretter gi automatisk tilgang til andre integrerte ressurser eller tjenester uten behov for å logge inn på nytt.

SSO øker brukervennligheten i stor grad ettersom brukeren ikke trenger å huske flere ulike brukernavn og passord. I planleggingen av oppgaven, var det viktig at det ble utviklet en applikasjon som var brukervennlig. Innlogging og autentisering er en viktig del av dette. Brukerne trenger ikke å opprette nye brukere selv, de bruker de eksisterende Ambita-kontoene sine til å få tilgang til applikasjonen.

4.2.1 SSO sikkerhetsrisikoer

Selv om SSO tilbyr mange fordeler, er det risikoer knyttet til SSO. Noen av de vanligste sikkerhetsrisikoene med SSO og autentisering generelt, er sesjonskidnapping, skripting på tvers av nettsteder (XSS) og mann-i-midten angrep (MITM). En uautorisert bruker kan utnytte usikrede sesjoner eller kommunikasjonskanaler for å få tilgang til autentiseringsdata. Gjennom XSS angrep kan angripere injisere skadelig kode for å stjele informasjonskapsler (eng. «Cookies») og annen sensitiv informasjon. Ved mann-i-midten angrep, angir angriperen seg for å være et pålitelig ledd i

applikasjonen. Brukeren tror den kommuniserer med applikasjonen, men kommunikasjonen går først gjennom angriperen og deretter til applikasjonen eller tjenesten.

4.2.2 Sikkerhetshåndtering

For å sikre løsningen best mulig mot disse og andre sikkerhetsrisikoer, benytter løsningen flerfaktorautentisering. Dette er i tråd med Ambitas påbud om å benytte flerfaktorautentisering, noe som har blitt nøye fulgt under utviklingen. Dette gjøres gjennom Microsoft Authenticator-appen. Ved å ha flerfaktorautentisering forbedres sikkerheten til brukerkontoen og systemet ved å ha flere former for verifisering før man er gitt tilgang til applikasjonen. Det blir betydelig vanskeligere for uautoriserte brukere å få tilgang til systemer og applikasjoner, når det er flere ledd i verifiseringen.

Msal.js biblioteket som er tatt i bruk for håndtering av autorisering bruker OAuth 2.0 protokollen og er også kompatibel med OpenID protokollen. OpenID og OAuth 2.0 er to åpne standarder for autentisering og autorisasjon på nettet [53]. OAuth 2.0 er en standard som er designet for å tillate autentisering på websider.

Webapplikasjonen har ingen områder som er utsatt for XSS angrep. Den har ingen input-felt hvor en kan injisere ondsinnet skript, dette fordi løsningen ikke har behov for å ha input-felt. Eneste stedet hvor det er mulig å injisere skripts er i Microsoft sin innloggingsportal som kan antas som pålitelig og sikker. Webapplikasjonen har ikke noen databaser knyttet til seg, det er da ikke noen trusler mot SQL-injiseringer i applikasjonen vår.

For å utvikle et sikkert produkt i AWS sitt skymiljø har utviklingen fulgt AWS sine anbefalte praksisprinsipper. Et av kjerneprinsippene i AWS er å følge minste privilegium prinsippet. Dette prinsippet betyr at det skal kun gis bruker tilgang til de ressursene som er nødvendig for å gjøre den oppgaven som skal utføres. AWS Identity and Access Management (IAM) blir brukt til dette. IAM hjelper å administrere tilgang til AWS-ressurser på en sikker måte. Ved å bruke IAM, kan det opprettes og administreres AWS-brukere, grupper og tillatelser til å gi tilgang til ressurser.

Tidlig i prosjektet ble det brukt ferdigkonfigurerte IAM-roller som gav generell tilgang mellom ressurser. Det ble tatt avstand fra disse ettersom det å ha mer tillatelser en nødvendig strider mot god sikkerhet. Dette er grunnen til at det ble opprettet en IAM-rolle med minimale tillatelser.

IAM-rollen i prosjektet gir tillatelsene:

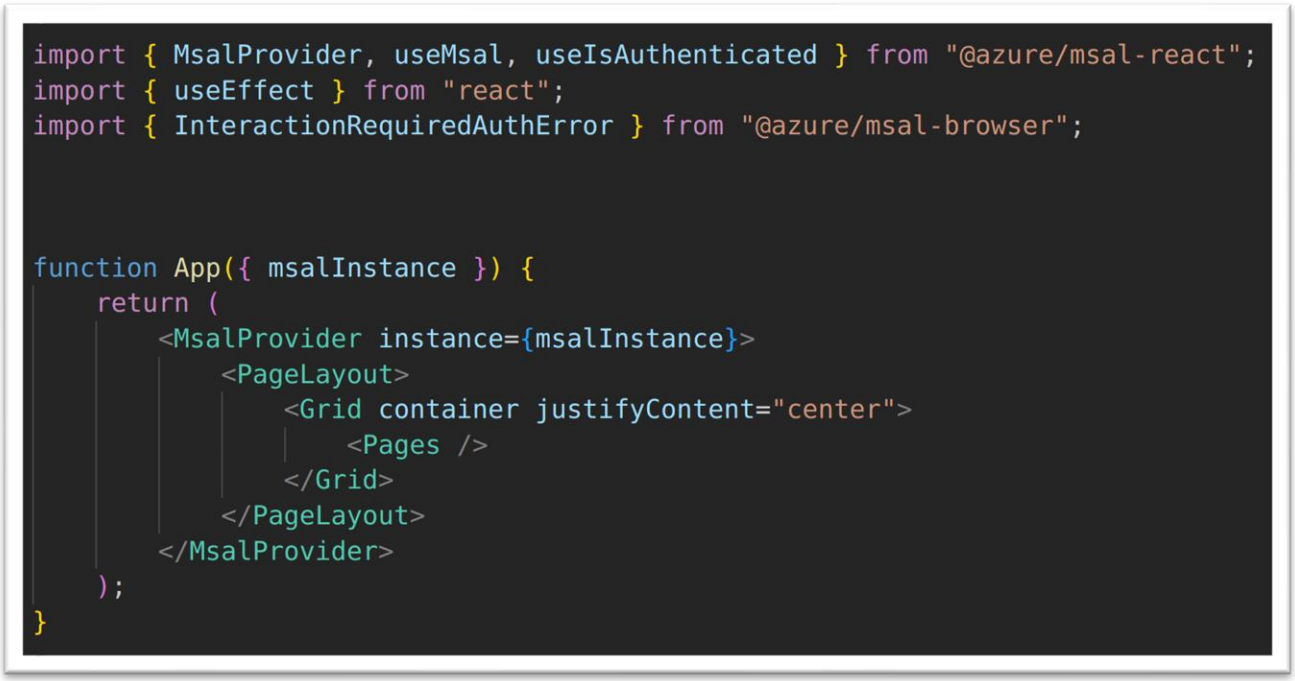
- Lambdafunksjoner kan starte og stoppe EC2-instanser
- Lambdafunksjoner kan lese data fra EC2-instanser
- API kan kalle på lambdafunksjoner og Step Functions
- Step Functions kan kalle på lambdafunksjoner

IAM-rollen ble brukt til alle AWS-tjenestene i prosjektet, på denne måten var de forskjellige tillatelsene gitt oversiktlig, og det var enkelt å gjøre endringer på ett samlet sted. Figur 4.2 viser et eksempel på konfigurasjonen av en tillatelse i IAM [54]. Tillatelsen er for å starte og stoppe EC2-instanser.

Figur 4.2 – Eksempel på en tillatelse i AWS

4.3 Hvordan implementere Msal.js?

Msal.js biblioteket for JavaScript tillater klient-side JavaScript applikasjoner å autentisere brukere ved å bruke Microsoft Entra ID. Denne versjonen av Msal.js bruker Authorization Code Flow som er en av de mest vanlige autentisering og autorisasjonsflyter som er i OAuth 2.0 rammeverket. Det er for det meste brukt av web og serverapplikasjoner. Msal.js biblioteket bruker også en utvidelse til Authorization Code Flow kalt PKCE (Proof Key for Code Exchange) [55]. PKCE hindrer CSRF og autorisasjonskodeangrep. CSRF (Cross-Site Request Forgery) er en type sikkerhetsangrep der en ondsinnet nettside kan sende en forespørsel til en annen nettside der en bruker er autentisert uten at brukeren er klar over det.



```
import { MsalProvider, useMsal, useIsAuthenticated } from "@azure/msal-react";
import { useEffect } from "react";
import { InteractionRequiredAuthError } from "@azure/msal-browser";



function App({ msalInstance }) {
    return (
        <MsalProvider instance={msalInstance}>
            <PageLayout>
                <Grid container justifyContent="center">
                    <Pages />
                </Grid>
            </PageLayout>
        </MsalProvider>
    );
}
```

Figur 4.3 - Hvordan implementere Msal.js biblioteket i applikasjonen

For å implementere Msal.js biblioteket i applikasjonen gjøres det slik det er vist i figur 4.3. Biblioteket fungerer slik at du pakker inn applikasjonen i taggen <MsalProvider>. Applikasjonen vil da bli lagt bak denne taggen som en innloggingsmur.

4.4 Arkitektur

4.4.1 Webapplikasjonen

AWS Amplify er en plattform med verktøy og tjenester fra Amazon Web Services som tilbyr tjenester for utvikling av fullstack applikasjoner og å gjøre dem tilgjengelig over internett. Amplify er spesielt tilpasset webapplikasjoner. Amplify tilbyr flere funksjoner innenfor autentisering, API, notifikasjoner, datalagring som forenkler prosessen å lage fullstack applikasjoner som benytter AWS tjenester. Amplify forenkler prosessen med å koble applikasjonens klientside med AWS-tjenester, og gir utviklerne muligheten til å lage kompleks kode med få linjer. Amplify støtter populære rammeverk og biblioteker slik som React, Angular og Vue.js

Å velge AWS Amplify som plattform for å tilgjengeliggjøre applikasjonen over internett kommer med betydelige fordeler. For det første sikrer integrasjonen med AWS at applikasjoner kan skalere for å møte varierende brukeretterspørsel, mens applikasjonen opprettholder høy ytelse og sikkerhet. Videre har Amplify innebygd mulighet for CI/CD-pipeline som automatiserer bygging og testing av applikasjonen, noe som kan redusere tidsbruk og mulige menneskelige feil under utviklingsprosessen. En annen fordel med Amplify er enkelheten i å starte utviklingen av applikasjoner.

Amplify tilbyr robuste og sikre tjenester for oppstart og utvikling. For å sikre pålitelig og effektiv utveksling av data mellom de ulike tjenestene og forespørsler, brukes et REST API. API-et er koblet gjennom API Gateway, som fungerer som en 'front dør' for innkommende API-forespørsler til serversiden. Forespørslene blir håndtert av AWS Lambda eller Step Functions. De utfører logikk knyttet til forespørsler og opererer med data opp imot Windows-serverne.

4.4.2 Design

Dette avsnittet omhandler designet på den endelige løsningen av programvaren. Programvaren er designet i betraktning ovenfor oppdragsgiverens ønsker, samt gruppens egne idéer for best mulig løsning med tanke på brukervennlighet og design. Hovedfokuset har vært å gjøre designet brukervennlig, noe som var et av kravene for løsningen.

Applikasjonen er designet gjennom flere iterasjoner av gruppen. Generelle aspekter med designet som fargekoder, logoer eller grafiske elementer er brukt i henhold til Ambitas profilhåndbok.

Profilhåndboken sier «Formålet med denne guiden er å formalisere vår visuelle profil og sikre uniform visuelle identitet i alle våre kommunikasjonskanaler!» [56].

Et annet bibliotek som er brukt for grafiske komponenter i webapplikasjonen er Material UI (MUI). MUI er et open-source bibliotek med React-komponenter som tilbyr enkle og justerbare komponenter for å implementere Googles Material Design. Grunnen til at MUI ble brukt er dets gode variasjon av stiler for komponenter, noe som gjorde det enkelt å integrere dem i applikasjonen.

Under designprosessen av applikasjonen var Jakob Nielsens 10 generelle prinsipper for interaksjonsdesign [57] avgjørende. Disse prinsippene bidro til å skape en brukervennlig og intuitiv løsning som oppfylte oppdragsgiverens krav. Spesielt fire av prinsippene var relevante og ble fulgt i prosjektet.

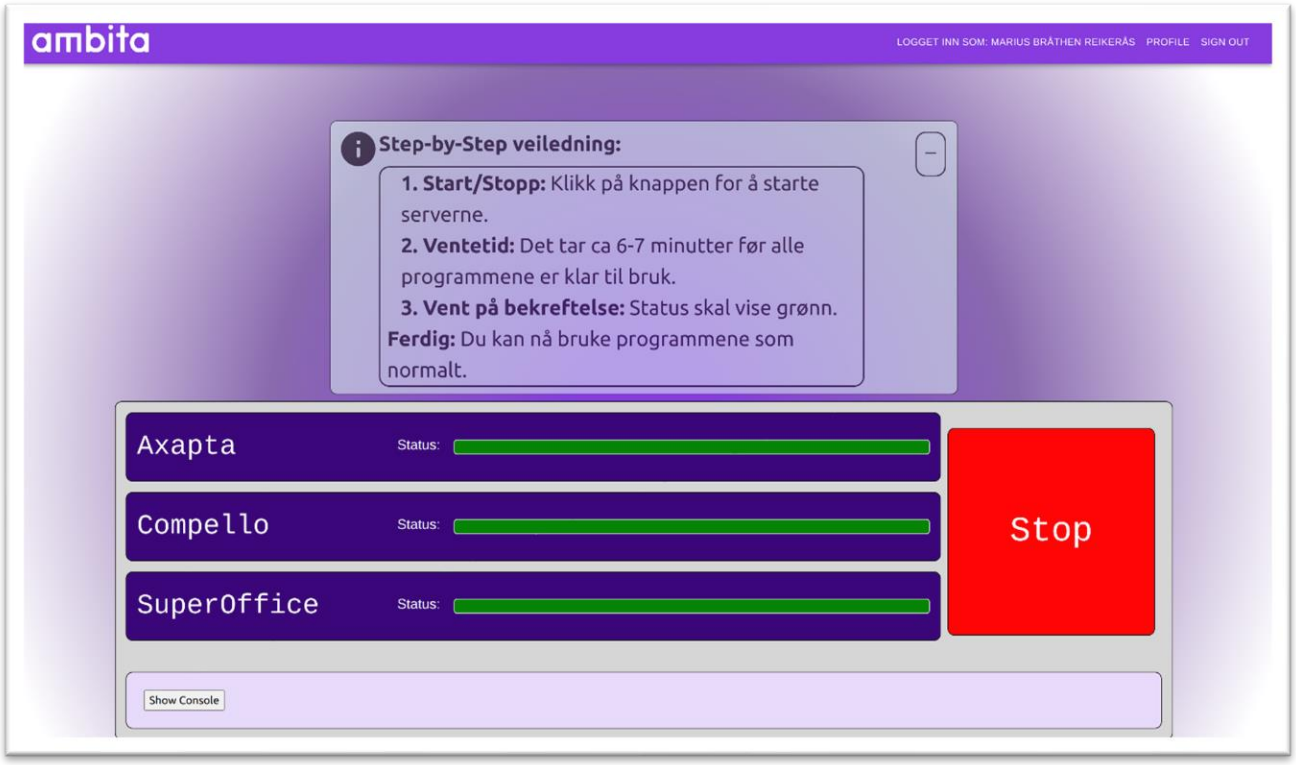
Det første prinsippet, synlighet av systemstatus (eng. «Visibility of system status»), sikret at brukerne alltid var informert om systemets tilstand. Dette var viktig for å gi brukerne kontroll og øke brukeropplevelsen.

Det femte prinsippet feilhåndtering (eng. «Error prevention») hjalp med å minimere feil ved å forhindre dem før de oppsto. Dette bidro til å forbedre brukeropplevelsen og redusere frustrasjon hos brukerne.

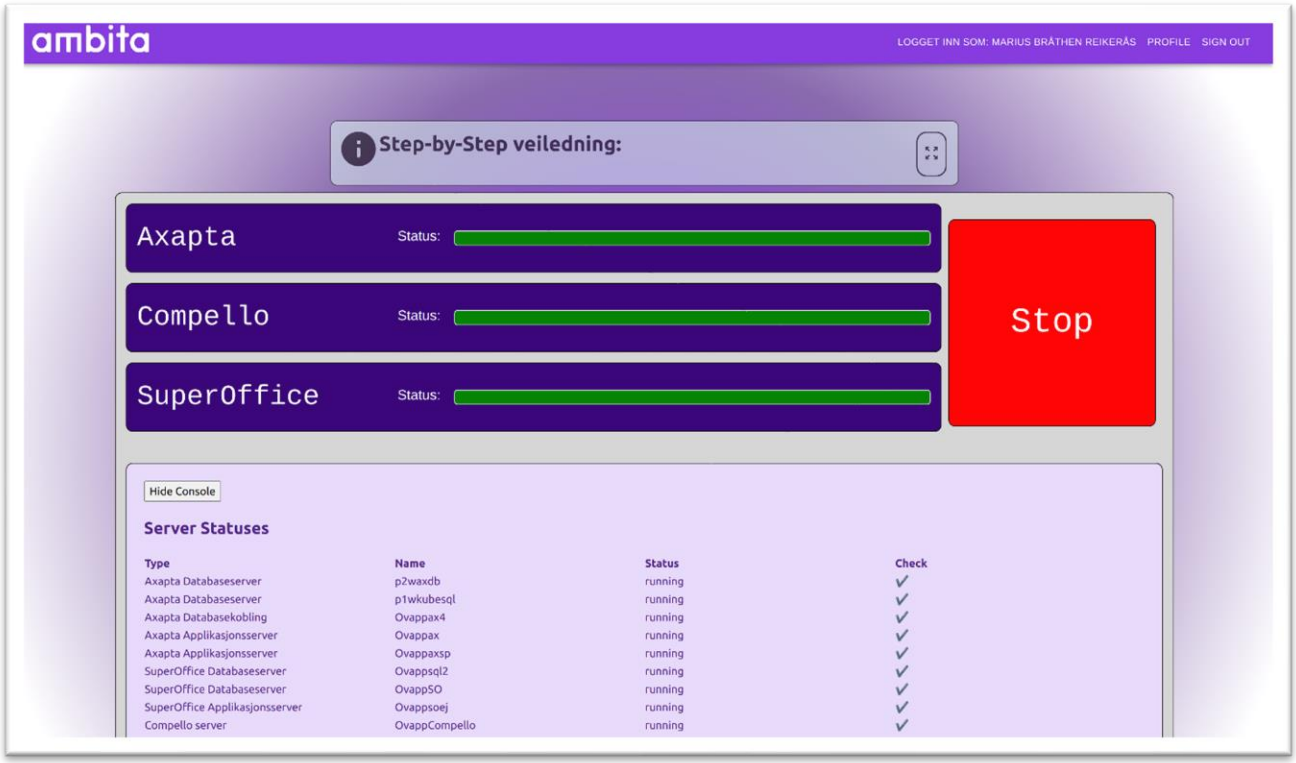
Minimalistisk og estetisk design (eng. «Minimalist and aesthetic design») prinsipp åtte innebar å fjerne unødvendig informasjon for å holde designet ryddig. Dette skapte en mer behagelig og enkel brukeropplevelse.

Til slutt ble hjelp og dokumentasjon (eng. «Help and documentation») det tiende prinsippet implementert gjennom en integrert bruksanvisning som klart beskriver hvordan man bruker applikasjonen. Dette veiledet brukerne og bidro til å sikre riktig bruk av applikasjonen.

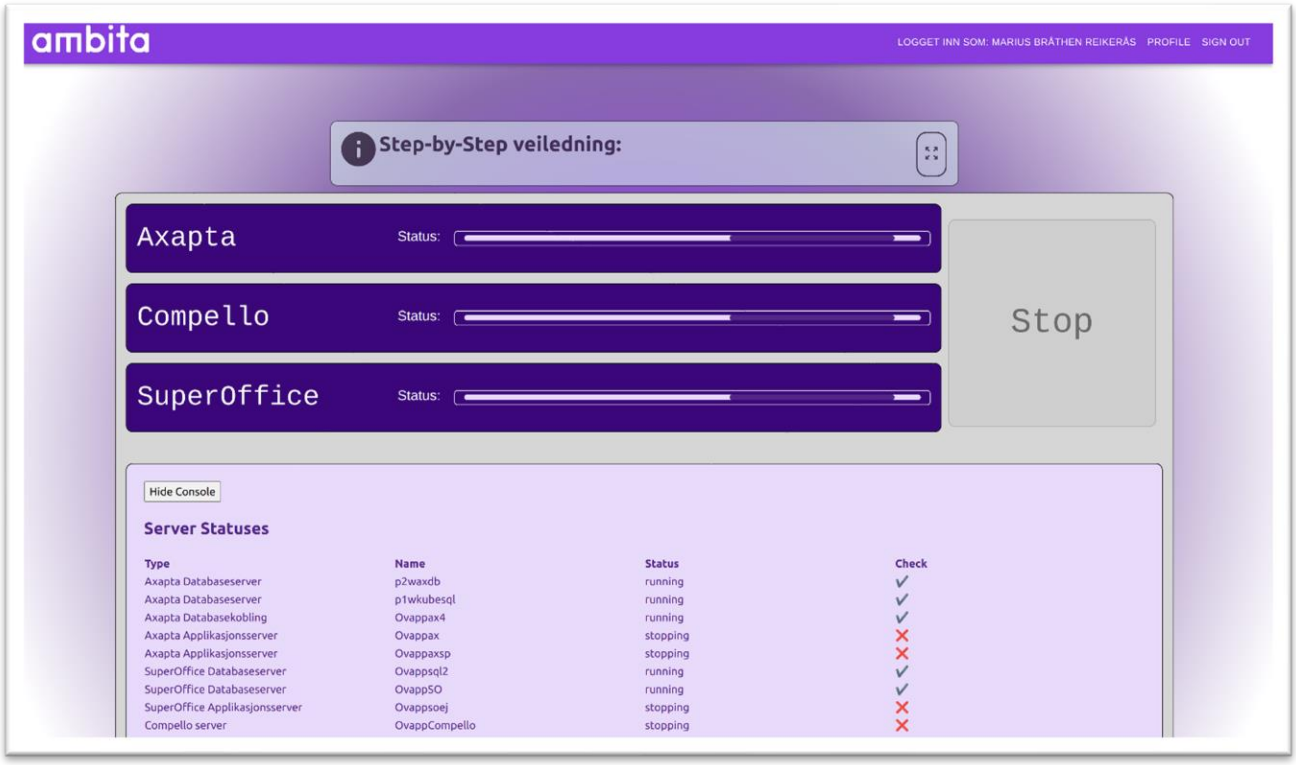
Disse prinsippene var avgjørende for å utvikle og designe en brukervennlig applikasjon, noe som var et av oppdragsgiverens mål. Ved å følge disse prinsippene ble brukeropplevelsen forbedret gjennom en mer intuitiv applikasjon. Figur 4.4– 4.7 viser skjermbilder fra applikasjonen. Det som vises er hjemmesiden før, under og etter start av serverne, samt veiledningen for bruk av applikasjonen.



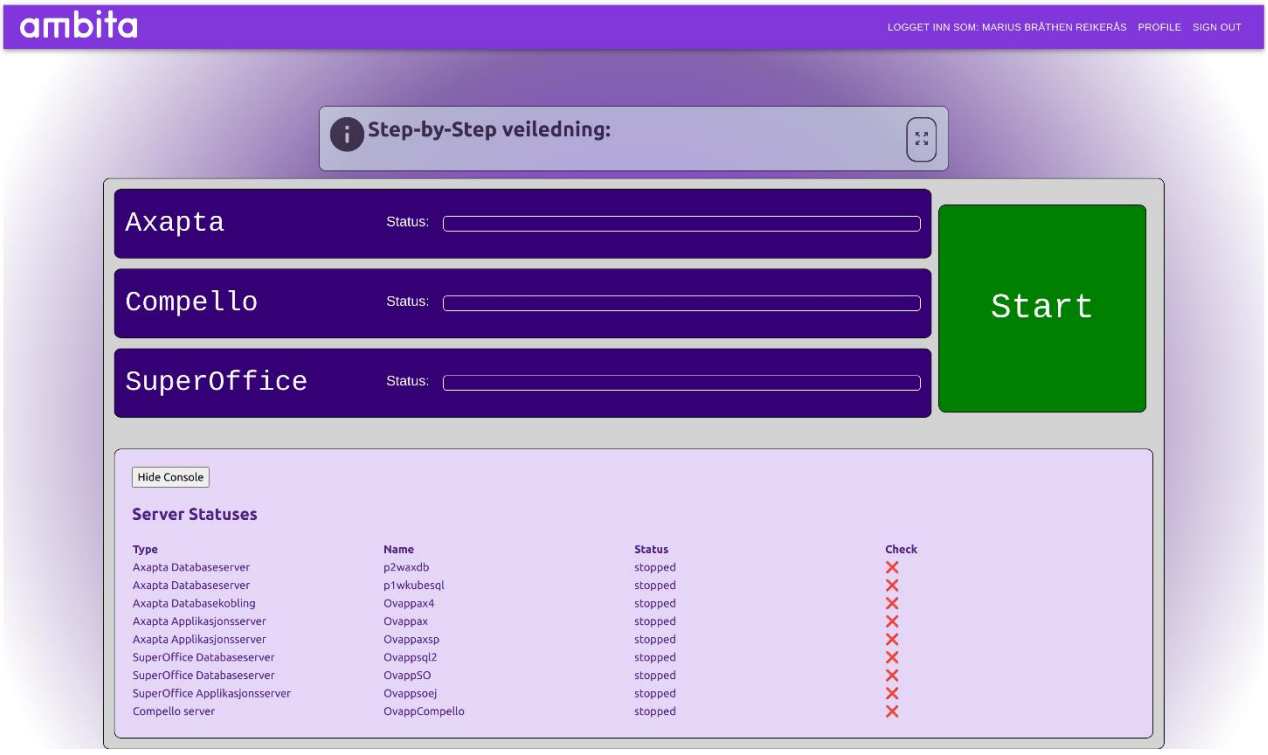
Figur 4.4 - Skjermbilde som viser veiledningen



Figur 4.5 - Skjermbilde med serveroversikt mens serverne kjører



Figur 4.6 - Skjermbilde etter stop knappen er presset, som viser stopprosessen

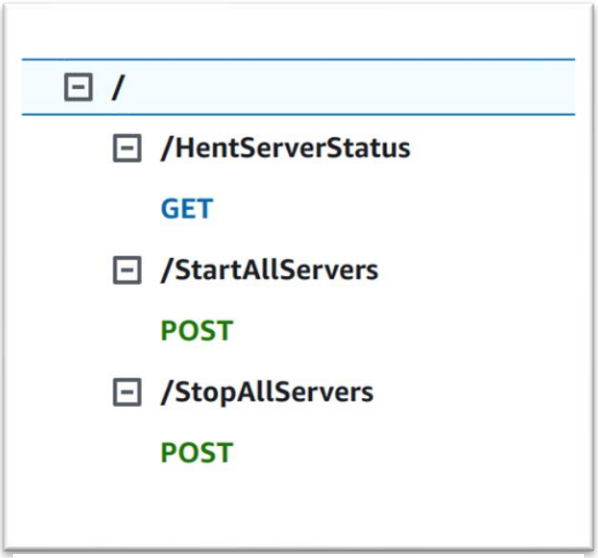


Figur 4.7 - Skjermbilde etter serverne er stoppet

4.5 API-et

Et API er et grensesnitt som gir tilgang til data og ressurser fra datasystemer. Det fremmer god kommunikasjon og samhandling mellom datasystemer. API Gateway er en tjeneste fra AWS som er til for å lage, utgi, drifte eller observere HTTP (REST) eller WebSocket API-er. Spesielt kan API Gateway koble systemer utenfor AWS opp mot tjenester eller ressurser fra AWS Cloud, altså skyen hvor alt er lagret. Et HTTP-api er et tilstandsløst forespørsel-svar (eng. «stateless request-response») api, noe som betyr at all informasjon må være del av forespørsler og svar mellom klient og server. HTTP-api er ikke kommunikasjon i sanntid. WebSocket api er en enkelt sanntids forbindelse mellom klient og server hvor beskjeder kan bli overført. Forskjellen mellom HTTP og WebSocket er at HTTP bruker REST-fulle metoder for å kommunisere mellom klient og server, mens WebSocket bruker sin sanntidsforbindelse.

I prosjektet er det laget et REST API. Dette API-et står for oppkoblingen mellom applikasjon og AWS-tjenester. API-et har 3 endepunkter også kalt URI (Uniform Resource Identifier). To av endepunktene er lenket til hver sin tilstandsmaskin [58], og ett endepunkt er lenket til en Lambda for å hente serverstatus. Det første endepunktet heter /StopAllServers og inkluderer en POST metode som kjører en tilstandsmaskin for å stoppe serverne. Det andre endepunktet /StartAllServers har en POST metode som kjører tilstandsmaskinen for å starte alle serverne. Siste endepunktet /HentServerStatus har en GET metode som henter serverstatus til de 9 forskjellige Windows-serverne. Figur 4.8 viser en oversikt over endepunktene hente fra API Gateway.



Figur 4.8 - viser de ulike endepunktene for apiet

SOP (Same-Origin Policy) [59] er en fundamental sikkerhetsmekanisme som moderne nettlesere har innebygd. Dette går ut på at det ikke kan bli forespurt ressurser fra andre nettsider som ikke er av samme opprinnelse. Dette er relevant for API-et fordi applikasjonen spør etter ressurser fra Ambita sin AWS. Måten dette er ordnet på er gjennom å konfigurere CORS (Cross-Origin Resource Sharing) [60] spesifikt for applikasjonen sin nettadresse. CORS er en sikkerhetsfunksjon som tillater nettsider å få tilgang til ressurser fra servere på ulike domener. CORS fungerer ved å legge til HTTP-headere som lar serveren spesifisere hvilke domener som skal ha tilgang til ressursene den tilbyr.

4.6 Automatiseringsprosessen

En av de viktigste delene av oppgaven er å starte og stoppe serverne i riktig rekkefølge slik at programmene som kjører på serverne blir startet opp riktig og fungerer som normalt. I begynnelsen av prosjektet, ble det fortalt at databaseserverne er nødt til å startes før applikasjonsserverene. Det er totalt ni servere. En server kjører Compello-programvaren, tre servere kjører SuperOffice-programvaren og de resterende fem kjører Axapta-programvaren. Serverne som kjører de tre ulike programmene er ikke avhengig av hverandre, det er kun de serverne som kjører samme program som er avhengig av hverandre. På serverne kjøres det både applikasjons- og databaseservere. Det er viktig at disse starter i riktig rekkefølge, fordi programvaren er avhengig av at de får kontakt med database og SQL-serverne for at programmet skal kjøre som normalt.

Når det gjelder å finne ut den korrekte rekkefølgen, er første skritt å gå dypere inn i infrastrukturen i AWS. Først ble servernes egenskaper og informasjon identifisert gjennom EC2 dashbordet. Dette ga en oversiktlig presentasjon av de relevante serverne. Her er det da mulig å klikke på de ulike serverne og få opp mye informasjon og detaljer av instansene. Ved å analysere serverne ble det funnet et attributt som het «LaunchTime». Attributtet sier noe om når tid denne serveren sist ble startet opp. Dette ble brukt til å finne en sammenheng mellom serverne. For eksempel hvis databaseServer1 hadde en «LaunchTime» på et tidligere tidspunkt enn databaseserver2, kunne det da bli brukt som et utgangspunkt i hvordan rekkefølgen skulle være. Dette ble gjort for samtlige servere og det ble laget alternativ til oppstartsrekkefølge.

Etter det hadde blitt utarbeidet en plan på hvordan teste rekkefølgen, var det tid for å teste og se om programvarene fungerte som forventet. Frem til løsningen til prosjektet er ferdig, vil disse serverne fortsette å kjøre kontinuerlig som før. For å teste rekkefølgen som ble utarbeidet, er det nødvendig å stoppe serverne. Dette ble gjort ved å stoppe serverne i motsatt rekkefølge av testrekkefølgen. Først stoppes applikasjonsserverene, deretter databaseserverne. Av sikkerhetsmessige årsaker ble én og én server stoppet. Når en server stopper, vil det ta litt tid før serveren er stoppet helt, dette varierer veldig fra server til server. I den endelige oppstartsrekkefølgen vil prosessene for å starte database serverne skje samtidig, men i starten ble rekkefølgen testet på en sikker måte slik at prosessen ble gjort ryddig og potensialet for feil i systemene ble minimert. Dette var et viktig tiltak for å finne ut den endelige rekkefølgen.

Automatiseringen av prosessen må helst ta kortest mulig tid. Dette er for at det skal være en brukervennlig løsning og at brukeren har mulighet til å bruke programmene raskest mulig når det er behov for det. Denne prosessen kan være vanskelig å optimalisere tidsbruken på, fordi det manglet tilgang til programmene. Uten tilgang var det ikke mulig å nøyaktig måle når programmene var klar til bruk etter oppstart av serverne. Utgangspunktet som er brukt er hvor lang tid det tar for å starte opp hver enkelt server, som vist i tabell 4.1 nedenfor. Her finnes det tre tider som er blitt brukt for å bestemme hvor lang tid prosessen behøver. Første og andre tid er tiden det tar å starte serveren eller stoppe den, og den tredje tiden er når serveren passerer EC2 sine statussjekker etter oppstart.

Tabell 4.1 - Oversikt over start- og stopp-tid på serverne

Program	Server	Type	Starttid	Statussjekker 2/2	Stopptid
Axapta	1.Databaseserver1	Database	**10 sek**	**1:50 min**	**0:35 min**
	2.Databaseserver2	Database	**10 sek**	**2:55 min**	**0:20 min**
	3.Applikasjonsserver3	Database-kobling	**35 sek**	**3:05 min**	**2:25 min**
	4.Applikasjonsserver4	App	**10 sek**	**3:00 min**	**0:55 min**
	5.Applikasjonsserver5	App	**45 sek**	**2:45 min**	**2:25 min**
SuperOffice	1.Databaseserver1	Database	**10 sek**	**2:00 min**	**0:30 min**
	2.Databaseserver2	Database	**45 sek**	**2:15 min**	**2:20 min**
	3.Applikasjonsserver3	App	**35 sek**	**2:20 min**	**2:30 min**
Compello	1.Applikasjonsserver1	App	**10 sek**	**2:15 min**	**1:00 min**

Gjennom prosessen ble tiden notert på hvor lang tid de ulike serverne brukte på å starte og stoppe. Det ble vist at databaseserverne brukte litt mindre tid på disse prosessene, enn det applikasjonsserverene brukte. Spesielt tiden det tok for å stoppe applikasjonene tok mye lengre tid enn databasene. Kolonnen «Starttid» viser hvor lang tid det tok fra da serveren startet, til det stod at serveren var kjørende. Kolonnen «2/2 Statussjekker» viser hvor lang tid det tar før serveren har fått statussjekkene sine godkjent. Tiden det tok varierte fra 1 minutt og 50 sekund og opptil 3 minutt og 5 sekund. Om programvaren på disse serverne avhenger av disse tidene har ikke vært mulig å si noe om. Kolonnen «Stopptid» viser hvor lang tid det tok fra serverne kjørte og til de var stoppet. Det blir tatt utgangspunkt i at statussjekkene må være fullført før en kan gå videre til neste steg i automatiseringsprosessen.

For å kunne vite om serverne hadde blitt startet opp korrekt, ble den ansatte i regnskapsavdelingen kontaktet. Den ansatte gikk da inn på programmene og testet om de fungerte som normalt. Tilbakemeldingen var at programmene fungerte som vanlig og det var ingen problemer ved bruk. Dette betydde at rekkefølgen som ble utarbeidet var korrekt, og kunne da begynne med å automatisere denne prosessen.

4.7 Lambdafunksjoner

AWS Lambda er en serverløs databehandlingstjeneste fra AWS. Lambda tillater å kjøre kode hvor tjenesten tar seg av det administrative og skalering, noe som gjør det betydelig enklere å fokusere på hva lambdafunksjonene skal gjøre. I prosjektet er det laget fire lambdafunksjoner for start og stopp av serverne, en lambda for å hente serverstatus og to lambdaer for å starte tilstandsmaskinene Disse lambdafunksjonene er skrevet på kodespråket Python versjon 3.8.

De fire lambdafunksjonene for start og stopp av serverne gjennomfører enkle kall til de 9 Windows-serverne. Disse funksjonene blir kjørt gjennom oppsett i tilstandsmaskinen hvor oppstartsrekkefølgen til serverne blir ivaretatt. Tilstandsmaskiner vil bli beskrevet i neste kapittel.

Den ene lambdaen for å hente serverstatus blir kjørt hver gang noen logger inn på siden. Dette er for å oppdatere siden sin informasjon på hvilken status serverne har, slik at bruker av applikasjon har ett oppdatert grensesnitt. Komponenter som tabelloversikt og knappen baserer seg på serverstatus og blir oppdatert i det siden lastes og underveis i applikasjonens bruk.

De to siste lambdaene i prosjektet har som oppgave å starte tilstandsmaskinene. Disse lambdaene er koblet opp mot API-endepunkter som trigger lambdaene hver gang endepunktet blir brukt.

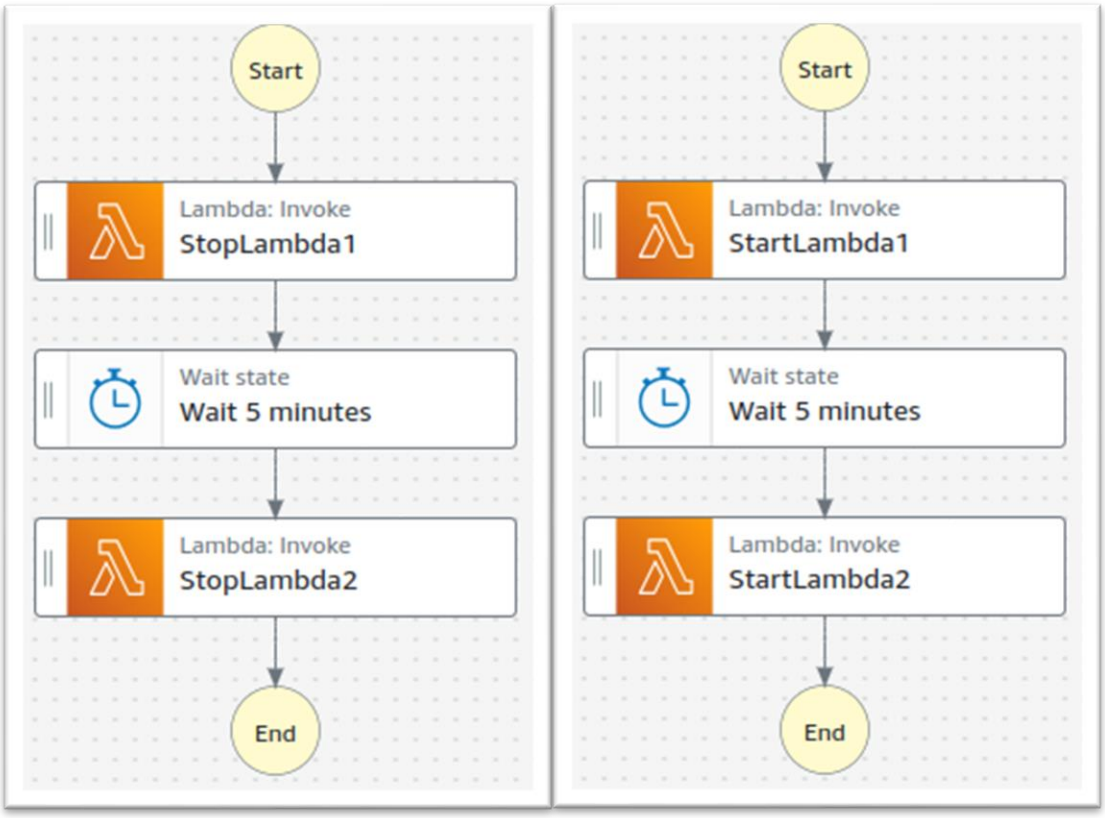
Felles for alle lambdafunksjonene i prosjektet er at de har fått tildelt samme IAM rolle som gir de den rette tilgangen. Som for eksempel trenger de fire start- og stopp-funksjonene-tilgang til å starte og stoppe ec2-instanser. IAM blir brukt som ett sikkerhetstiltak slik at lambdaene og andre AWS-tjenester og ressurser kun har nødvendig tilgang for det funksjonelle.

4.8 Step Functions og tilstandsmaskiner

Step Functions er en serverløs orkestreringstjeneste som gjør at det kan koordineres og lages sekvenser av ulike AWS-tjenester til serverløse arbeidsflyter. Step Functions har flere egenskaper som automatiserer sekvenser av AWS-tjenester. En av egenskapene Step Functions tilbyr er tilstandsmaskiner (eng. «State Machines»).

Tilstandsmaskiner er programmer laget av flere steg. Maskinen utfører et steg og går til neste, helt til alle stegene er utført. I prosjektet er det to tilstandsmaskiner, en for å starte serverne og en for å stoppe serverne. Disse er satt opp til å kjøre tidligere nevnte lambdafunksjoner sekvensielt. Mellom de forskjellige funksjonene er det lagt til en ventetid på 5 minutter. Ventetiden er i forbindelse med oppstartsrekkefølgen av serverne. For å sikre normal funksjonalitet deles serverne inn i applikasjons- og databaseservere. Applikasjonsserverene har en avhengighet til databaseserverne og dermed må databaseservere starte først, og oppnå normal kjøring før applikasjonsserveren kan starte. Grunnen for at ventetiden er satt til 5 minutt er for å sikre at databaseserverne kjører med full funksjonalitet. En visuell arbeidsflyt kan enkelt settes opp ved hjelp av den innebygde designeren i AWS Step Functions. Figur 4.9 er en visuell representasjon av konfigurasjonen.

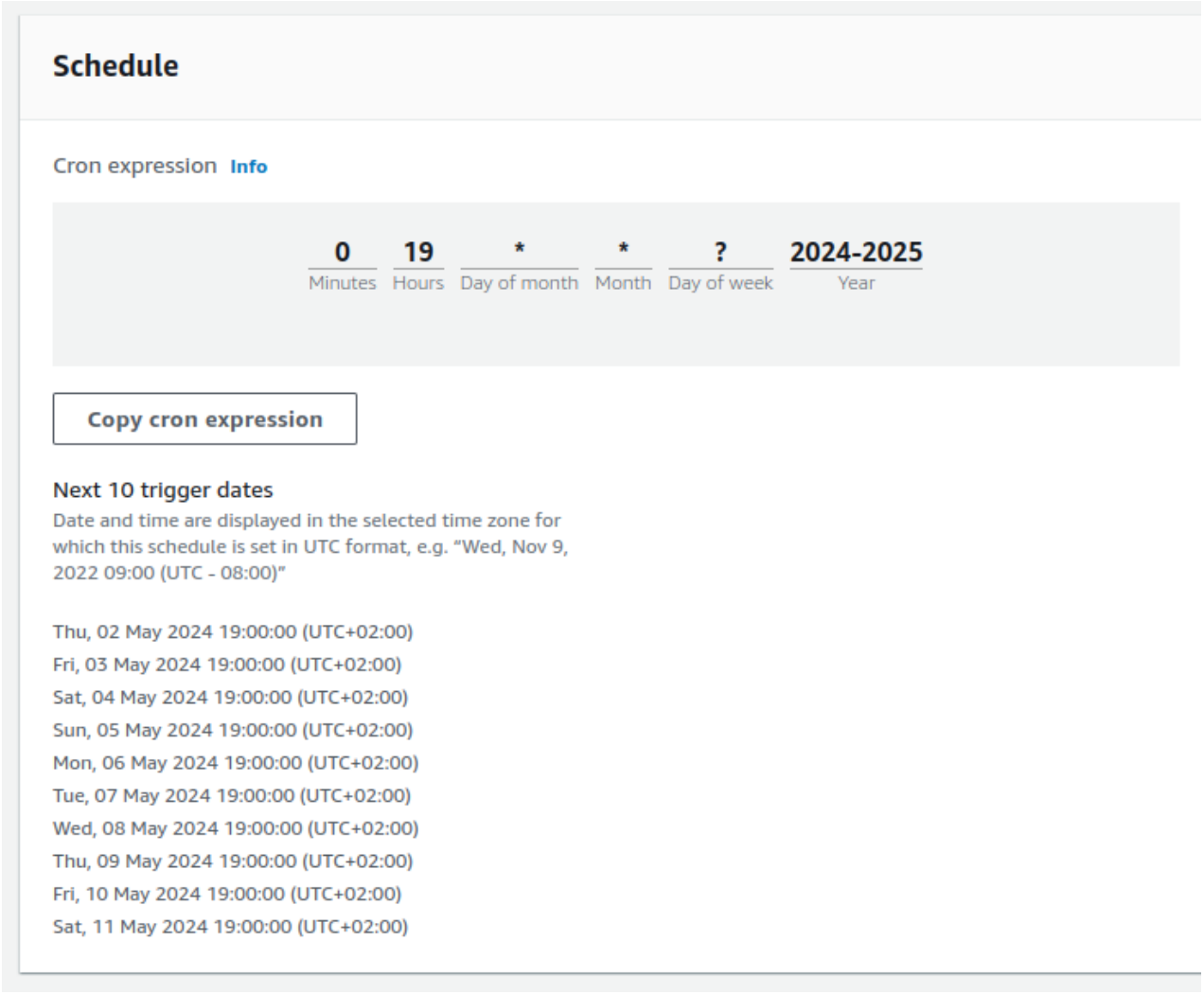
Selve oppgaven tilstandsmaskiner utfører kunne også blitt løst med bruk av lambda. Grunnen for at lambda ikke var en passende løsning på dette er at den vil bruke kjøretid mens den venter de 5 minuttene. Som nevnt tidligere bruker AWS en betal etter forbruk modell som ville føre til at lambdaene kostet betydelig mer. AWS tar kun betalt for hvert steg i tilstandsmaskinene noe som fjerner de store ekstrakostnadene for å vente 5 minutt. Nedenfor vises et bilde av arkitekturen og sammenhengen for tilstandsmaskinene som ble brukt. Hvor den til venstre beskriver prosessen for oppstart av serverne, mens tilstandsmaskinen til høyre beskriver den motsatte prosessen. Hver prosess starter med at tilstandsmaskinen starter databaseserverne, venter fem minutter for å sikre at de er kjørende før den starter de resterende applikasjonsserverene.



Figur 4.9 – Figuren viser tilstandsmaskinene for stopp og start av servere

4.9 AWS Eventbridge

Ettersom applikasjonens design forutsetter at brukeren manuelt stopper serverne etter bruk igjennom nettsiden, er det viktig å ha en sikkerhetsplan for tilfeller der dette glemmes. En slik plan kan implementeres ved hjelp av AWS EventBridge. Denne tjenesten muliggjør opprettelsen av en automatisert tidsplan for å stoppe serverne. Det ble implementert en tidsplan for dette scenarioet som aktiveres hver dag klokken 19:00. Dette tidspunktet ble valgt etter en dialog med intern veileder. For å sikre at tidsplanen gir rom for eventuell overtid uten å forstyrre brukerne. Den normale arbeidstiden er mellom 07:00 – 17:00 vil dette tidspunktet sikre at brukerne ikke vil bli forstyrret selv med overtidsarbeid. I tillegg er det også mulig å manuelt stoppe eller endre denne tidsplanen dersom behøvet. Tidsplanen fungerer ved dersom serverne fortsatt er aktive på det angitte tidspunktet. Da vil systemet automatisk utføre de nødvendige Lambda-kallene for å stoppe dem, på samme måte som ved bruk av stopp-knappen på nettsiden. Konfigurasjonen av denne tidsplanen er illustrert i figur 4.10.



Figur 4.10 – Figuren viser konfigurasjonen av tidsplanen som er brukt

5 Resultater

5.1 Evalueringsmetode

For evaluering av løsningen er det brukt Scrum underveis, løsningen er testet av brukere og løsningen er blitt tatt i bruk av Ambita for å samle data for økonomiske og ressursbesparelser. Det er gjennom disse metodene det er kommet frem til et prosjektresultat og et resultat for økonomi og ressurser. Det er også et vurdert et resultat av gjennomføringen av prosjektet.

5.1.1 Utviklingsmetode

Tidligere forklart er det brukt en utviklingsmetode tilnærmet Scrumban. Det er vurdert arbeid, fremgang og resultater underveis for hver sprint gjennom møter internt i gruppen og i lag med bedriften. Utviklingsmetoden har vært en viktig del av evalueringen gjennom hele prosjektet. Eksempelvis med hvordan IAM rollen for løsningen ble dannet. Under møte med oppdragsgiver og ekstern veileder for planleggingsfasen ble det kommentert sikkerheten for prosjektet. Nærmere ble det kommentert IAM roller og hvor viktig det var at lambdafunksjoner og andre tjenester ikke hadde mer tillatelser enn nødvendig. Metodikken har bidratt til avsluttende evaluering av løsningen, gjennom ett møte med Ambita.

5.1.2 Brukertesting

Under utviklingsprosessen ble løsningen vist fram til den interne veilederen i flere uformelle møter. I tillegg ble det gjennomført tre større møter med ansvarlige fra Ambita etter avsluttet sprint. Her ble det presentert løsningsidéen og designet av brukergrensesnittet. Gjennom møtene fikk bedriften muligheten til å komme med innspill og eventuelt oppklare om det hadde skjedd en mistolkning av behovet.

5.1.3 Testing av produktet

Løsningen ble 18.April tatt i bruk av bedriften. Løsningen hadde ønsket funksjonalitet og nødvendigheter for å bli tatt i bruk på en sikker måte. Den ble tatt i bruk for å samle data relatert til økonomiske besparelser og ressursforbedringer. Samtidig oppfylte løsningen ønsket til Ambita for automatiseringen av serverne gjennom et enkelt brukergrensesnitt. Der var mindre mangler på dette tidspunktet som senere har blitt utbedret.

5.2 Evalueringsresultat

Evalueringsmetodene som er brukt har dannet grunnlag for å besvare problemstillingen og vurderes opp imot krav for løsning. Ambita har også gjennomgått en vurdering av løsningen. Ut ifra testingen av produktet kan det fastslås at løsningen oppfyller kravene gitt for oppgaven. Dette kan begrunnes med disse funksjonelle og ikke-funksjonelle egenskapene for løsningen:

Det er opprettet automatisk styring av Windows-serverne

Brukergrensesnittet er intuitivt og brukervennlig på den måten at mulighetene for å gjøre feil er minimale.

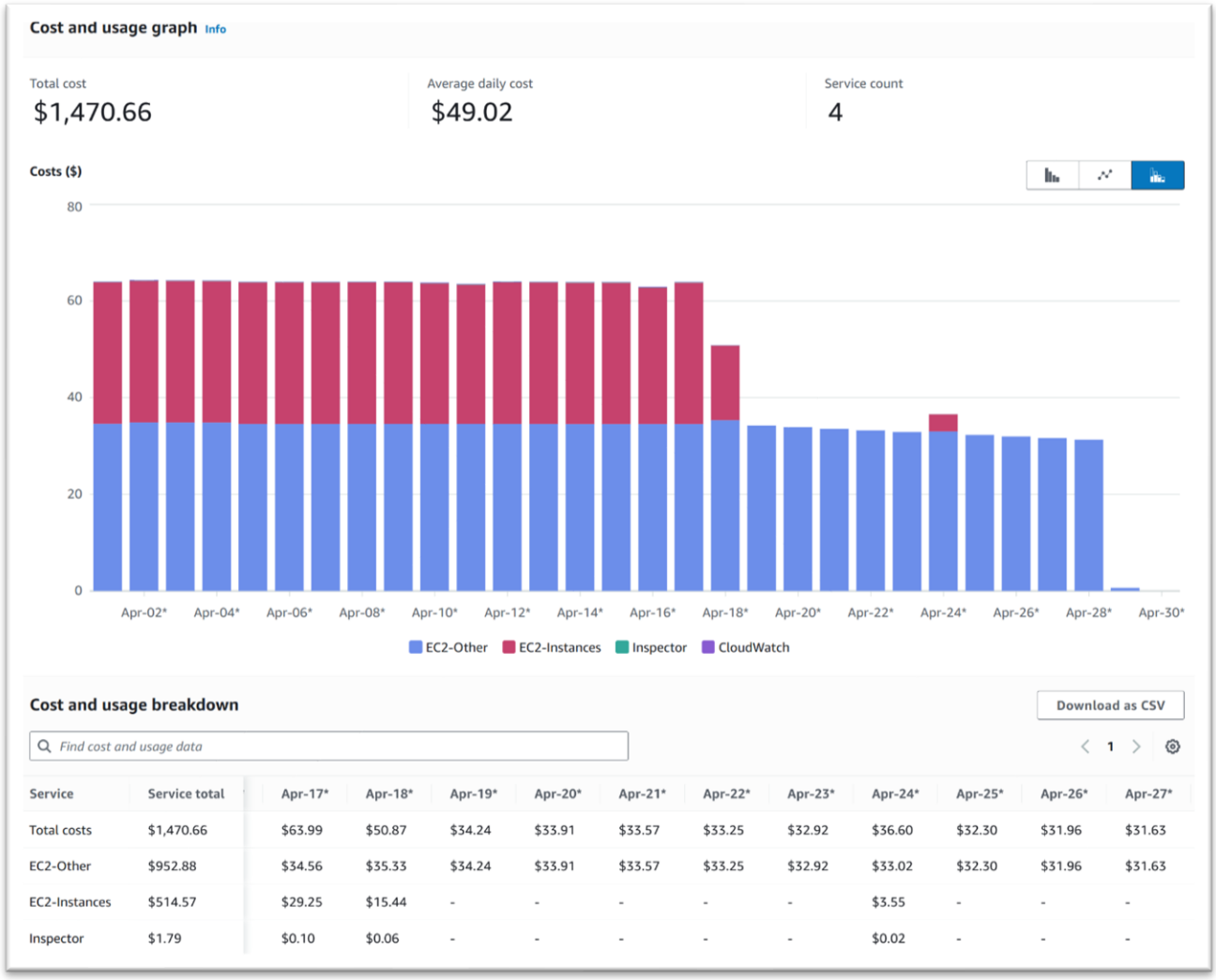
- Nødvendige sikkerhetstiltak er implementert. Applikasjonen gir kun tilgang til funksjonalitet dersom bruker er logget inn og autentisert.
- Applikasjonen gir et oversiktlig bilde av prosessen for bruker.
- Applikasjonen viser instruksjoner for bruk
- Løsningen sparer økonomiske utgifter og ressurser for Ambita.

Ambita har gjort en evaluering av løsningen. I evalueringen kom det frem at løsningen utfyller ønsket funksjonalitet på en god måte. Relevante punkter som økonomibesparelser og ressursoptimalisering er trukket frem som noe av det bedre med løsningen. Alle unødvendige driftskostnader er kuttet. Det er mindre serverbelastning, ettersom de ikke kjører kontinuerlig og det blir spart ressurser som lagring. Et punkt som stod frem for Ambita var mulighetene for å bruke innloggingsløsningen fra prosjektet til andre interne prosjekter. Applikasjonen har enkelte feil som bør utbedres. Den ene feilen er relatert til innlastingen av applikasjonen. Etter en bruker logger inn hentes informasjon fra Windows-serverne som er med å forme utseende til nettsiden. Denne prosessen tar cirka ett sekund, noe som gjør at knappen ikke er riktig innstilt og viser muligens feil det første sekundet. En annen feil er at noen komponenter som viser prosessen for at serverne slår seg av eller på kun er koblet opp imot handlinger fra klientside. Dette vil si at dersom siden lastes inn på nytt under oppstartsprosessen, mister komponentene forbindelse til prosessen og viser ikke lenger animasjon for lasting. Selv om siden lastes inn på nytt midt i oppstartsprosess eller stopping vil de fortsette feilfritt.

5.3 Prosjektets økonomiske resultat

Oppgavens problemstilling er å effektivisere ressursbruk og redusere kostnaden til de omtalte serverne. Den ferdig implementerte løsningen har automatisert Windows-serverne og tilfredsstilt de tre hovedkravene løsningen. I denne delen blir det presentert hvor mye løsningen har redusert kostnadene og hvor mye den kommer til å spare Ambita for frem til 1.Januar 2025.

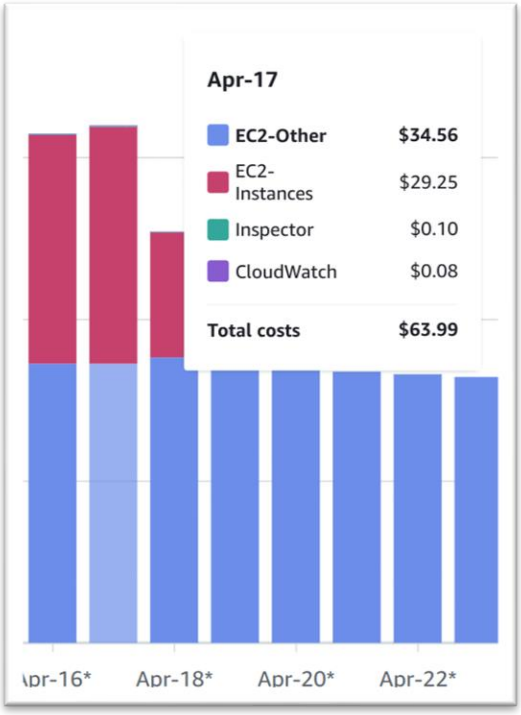
Før utviklingen av løsningen startet, ble kostnadene til serverne analysert. Disse serverne har ulike tjenester knyttet til seg for å kunne fungere som normalt, se kapittel 2.4. Kostnadene til de ulike tjenestene som er knyttet til serverne ble analysert og det ble gitt et overblikk over hvordan kostnadene henger sammen med hverandre. Kostnaden knyttet til kategorien EC2-instances hadde en månedlig kostnad på $877, omtrent 10 000 kroner, se tabell 2.1. Disse tallene er hentet fra AWS sin prismodell. I tabell 2.1, kan en se at denne kostnaden varierer fra måned til måned, men dette er lite ubetydelig for analysen av det økonomiske resultatet. I tillegg er det en kostnad som kommer oppå dette. Denne kategorien er EC2-Other, dette er kostnader knyttet til lagring av EBS-volumer og snapshots, dataoverføringskostnader, NAT-Gateways, elastiske- og offentlige ip-adresser. Denne kostnaden per måned er på $1040, omtrent 11 500 kroner. Disse to kostnadene har holdt seg relativt jevnt siden desember 2021.



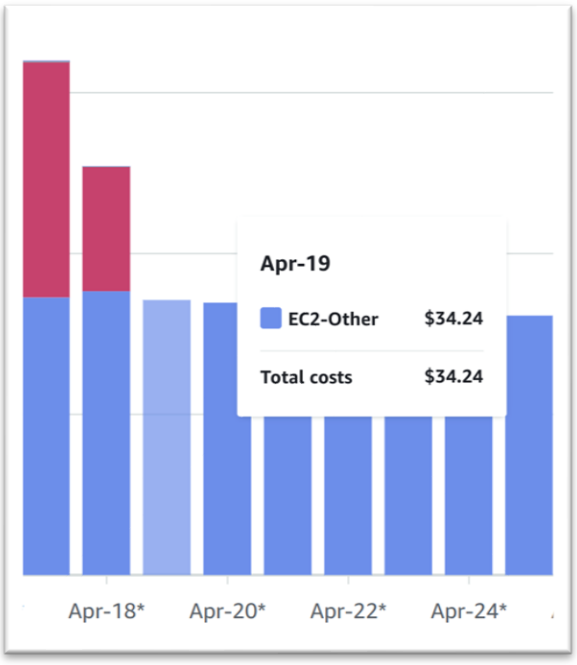
Figur 5.1 – Figuren illustrer en oversikt over redusering av kostnader etter løsningen er tatt i bruk

Etter at løsningen ble implementert 18.april, observeres det en betraktelig reduksjon i totale kostnader. Som presentert i figur 5.1, kan en se redusering av kostnaden til kategorien EC2-instances (markert med rød farge) er eliminert. Årsaken til dette er at serverne har blitt slått av og da påløper det ingen kostnader. Videre viser kostnadskategorien «EC2-Other» en gradvis nedgang. Grunnen for dette er når serverne er slått av, vil kostnaden til EBS-volumene og snapshots reduseres fordi disse ikke behøver å hele tiden lagre og ta nye kopier av volumene ettersom at serverne er slått av. De andre kostnadene innenfor denne kategorien vil heller ikke påløpe siden serverne er slått av. Den 24. april ser vi at det er en liten rød kolonne, dette viser at serverne ble slått på i en liten periode og deretter slått av igjen.

Figur 5.2 illustrerer at den daglige kostnaden den 17.april, var på $63.99, omtrent 700 kroner. Dette kostnadsnivået har vært konstant siden desember 2021. Figur 5.3 viser kostnaden etter at serverne ble slått av og viser hvor mye kostnaden har blitt redusert med etter løsningen ble tatt i bruk. Den grafiske fremstillingen indikerer en nedgang på omtrent $30, tilsvarende omtrent 330 kroner. Denne visualiseringen gir en klar indikasjon på de økonomiske fordelene som følge av den implementerte løsningen.



Figur 5.2- Figuren viser kostnad per dag før løsningen er implementert



Figur 5.3 - Figuren viser kostnad per dag etter løsningen er implementert

Dette vil si at løsningen har klart å redusere kostnaden med omtrent 330 kroner per dag. For å finne ut hvor mye løsningen vil spare Ambita for frem til 1.januar 2025, er det tatt utgangspunkt i hvor mange dager det er igjen av året og ganger med beløpet vi sparer per dag:

$$245\ dager * 330\ kroner = 80\ 850\ kroner$$

Den implementerte løsningen vil kunne potensielt spare Ambita for opptil 80 850 kroner frem til 1.januar 2025. Dette tar utgangspunkt i at serverne er slått av hver dag, men i praksis er et slikt scenario urealistisk. Trolig vil besparelsen være lavere enn hva estimatet antyder.

Målet for prosjektet som ble presentert i kapittel 1.4, er å implementere en løsning som både reduserer kostnader og optimaliserer ressursbruken til serverne. Den implementerte løsningens resultat, har demonstrert sin evne til å oppnå disse målene. En kvantitativ analyse presenteres for å illustrere den økonomiske innvirkningen løsningen har hatt, inkludert en prognose av forventede

besparelser frem til 1. januar 2025. Det er tydelig at dette økonomiske målet er det fremste for prosjektet, ettersom det står sentralt i å oppfylle hovedmålet om kostnadsreduksjon. Denne suksessen kan i stor grad tilskrives den løsningen som er utviklet i løpet av prosjektet.

5.4 Økonomisk vurdering av implementert løsning

Den implementerte løsningen benytter seg av flere tjenester og ressurser som AWS tar betalt for, dette må selvfølgelig analyseres og presenteres. Selv om kostnaden knyttet til dette er ubetydelig lav, vil det bli forklart i et kort avsnitt.

I den økonomiske utregningen av tjenester og ressurser som løsningen benytter seg av er det kommet frem til at å driftskostnadene for publisering av et Amplify-prosjektet er estimert til $0.01 månedlig, omtrent 0,11 kroner. Videre er det identifisert at kostnader knyttet til kall av lambdafunksjoner, HTTP-forespørsler og stegene i tilstandsmaskinen, koster $0.1010716, tilsvarende 1,12 kroner. Dette impliserer at den totale kostnaden forbundet med en komplett kjøringsprosess fra stoppet, til startet og deretter stoppet, beløper seg til 1,12 kroner. Så det vil si at en prosess koster 1,12 kroner.

5.5 Prosjektgjennomføring

Gjennomføringen av prosjektet har i hovedsakelig fulgt Gantt-diagrammet vist tidligere.

En innledende fase hvor grunnmuren for prosjektet er bygget på at gruppen tilegnet seg kunnskap om AWS og systemer som kunne brukes i prosjektet. Gruppen ble kjent med Ambita som bedrift og ble kjent med hvordan et bachelorprosjekt foregikk.

I planleggingsfasen har det blitt fokusert på å planlegge prosjektets gang, alternative løsninger og design og arkitektur. Tidlig var det utfordringer om nøyaktig løsning på prosjektet. AWS har over 200 tjenester, hvor flere tilbyr mulige måter å oppnå et lignende resultat på som det endte opp med for prosjektet. Dette løste seg etter hvert som kunnskapsnivået økte. Det ble designet flere alternative brukergrensesnitt som ble samlet til ett felles design og senere presentert for bedriften. Den største utfordringen med denne fasen var langtidsplanlegging. Gruppen hadde ikke satt opp en Gantt-diagram ordentlig tidligere. Det var vanskelig fastslå hvor lang tid ulike faser, milepæler

eller oppgaver skulle ta. Det har vært endringer på Gantt-diagrammet flere ganger etter det ble laget.

Den tekniske fasen av prosjektet er den fasen som har gitt mest læringsutbytte og det er her flest utfordringer har oppstått. Ambita laget et eget utviklingsmiljø for bachelorprosjektet hvor gruppen kunne utforske AWS-tjenester, hvordan de samhandlet og utføre testing underveis. Her lagde gruppen ett førsteutkast av løsningen. Opprettet et Amplify prosjekt, API, lambdafunksjoner og testet samhandling for å slå av noen EC2 instanser satt opp for testing. Når løsningen først hadde fått innlogging og nødvendig sikkerhet ble løsningen migrert til Ambita sin hovedplattform. Her har resterende funksjonalitet og tilkobling mellom klient og server satt opp. Utfordringer som ble møtt underveis var hvordan å koble applikasjonen til API-et og å gjøre Amplify til vertsplattform for applikasjonen. API-et ble koblet til applikasjonen med endepunkter, men forespørselen ble ikke innvilget. Dette var på grunn av SOP som forklart tidligere og det ble fikset med CORS. Utfordringen med Amplify som vertsplattform, var å gi all nødvendig data og ressurser som var behøvd for å kjøre applikasjonen. Figur 5.4 viser skriptet for å bygge applikasjonen på internett.

File: amplify.yml

```
version: 1
frontend:
  phases:
    prebuild:
      - cd ec2sandbox
      - npm i
    build:
      commands:
        - npm install
        - npm run build
  artifacts:
    baseDirectory: ./ec2sandbox/build
    files:
      - '**/*'
  cache:
    paths:
      - node_modules/**/*
backend:
  phases:
    build:
      commands:
        - cd ec2sandbox
        - pip install pipenv
        - pipenv install --deploy --ignore-pipfile
    artifacts:
      files:
        - '**/*'
```

Figur 5.4 – Build scriptet til Amplify prosjektet

Det totale læringsutbytte og resultatet av å gjennomføre prosjektet har vært stort. Det er bygget en fullstack-applikasjon med flere koblinger mot AWS og Ambita sine systemer. Det er brukt mange ulike AWS-tjenester som i en helhet utfører oppgaven på en god og effektiv måte. Majoriteten av prosjektet har inkludert nye og ukjente områder, noe som har resultert i ny lærdom og kunnskap. Selve prosjektgjennomføringen har gruppen sett på som en suksess med forebedringspotensiale. En av forbedringene er en mer konkret arbeidsfordeling på daglig basis og gjerne konkrete arbeidsfordelinger for fremtidig arbeid.

6 Diskusjon

I dette kapittelet presenteres en analyse av konsekvensene knyttet til de tilnærmingsmetodene som er valgt. Videre vil en undersøkelse av hvordan ulike begrensninger, metodikker, verktøy og strategiske valg har influert prosjektets utfall.

Konsekvensene av gruppens beslutning om å implementere løsningsalternativ nr.1, fremlagt i delkapittel 3.1.1, har i stor grad resultert i at løsningen på problemstillingen har blitt innfridd. Løsningen har oppnådd de etablerte målene om nedskjæring av kostnader og forbedring av ressursutnyttelsen som det redegjøres for i kapittel 5. Funksjonalitet og brukervennligheten til brukergrensesnittet er vurdert som både intuitivt og sikkert. Implementering av disse egenskapene bidrar til en helhetlig løsning hvor produktet innfrir målene og kravene til problemstillingen. Ettersom at produktet er blitt tatt i bruk av bedriften, er det forventet at den vil være i kontinuerlig bruk frem til 1.januar 2025.

Gjennom utviklingen har løsningen også indirekte bidratt til å løse en annen funksjonell utfordring for bedriften. Utviklingen av vårt prosjekt som benyttet Amplify for klientsidens utvikling, ble det implementert single sign-on (SSO) autentisering som en del av løsningen. Denne løsningen ble anerkjent internt i bedriften, da et spørsmål om implementering av SSO i Amplify-prosjekter ble stilt i bedriftens felles Slack-kanal. Det ble fremhevet av den interne veilederen at løsningen hadde realisert SSO-integrasjonen i prosjektets Amplify miljø. Dette støttet ikke bare opp under av at det ble valgt riktig løsning for autentisering og sikkerhet, men fungerte også som en verdifull referanse som potensielt kunne hjelpe andre team som stod ovenfor lignende problemstilling. Ved å ha delt hverandres kunnskap og erfaringer, kunne vi dermed bidra til en bredere forståelse av løsninger innenfor bedriftens prosjektportefølje.

I den innledende fasen av prosjektet ble det utviklet skisser for det planlagte brukergrensesnittet, dette var inspirert av Ambita sin hjemmeside [61] og OpenAI sin oversikt over tjenestestatus [62]. I vedlegget Kravdokument finnes det prototyper for designet, sekvensdiagram og annet arbeid fra den innledende fasen.

Selv om prosjektet ble offisielt tatt i bruk den 18. april, er det klart at løsningen har potensiale for forbedringer på enkelte områder. En av områdene for forbedringer er oppstartsrekkefølgen. Utfordringen skyldes hovedsakelig en ufullstendig forståelse ovenfor hvordan serverne samhandler og avhengigheter mellom dem. I tillegg mangel på klare signaler om når programvaren på serverne opererer normalt etter start av serverne. Gjennom en dialog med serveransvarlig i Ambita ble det forklart at det var en avhengighet mellom databaseserverne og

applikasjonsserverene. Imidlertid ble det ikke klart om avhengigheten var gjensidig eller ikke, noe som førte til en antagelse om en gjensidig avhengighet mellom disse.

Den implementerte løsningen involverer at en tilstandsmaskin venter i fem minutter mellom oppstart av databaseserverne og applikasjonsserverene. Med ytterligere testing av serverne og den kjørende programvaren kunne det vært mulig å utvikle en mer effektiv oppstarts- og nedleggingsprosess av serverne, men dette ble ikke prioritert i prosjektet. Det er også verdt å punktere at denne løsningen var bare en av flere mulige, og det kan godt hende at det finnes en bedre løsning enn hva som har blitt drøftet og utforsket i rapporten.

7 Konklusjon og videre arbeid

Under dette prosjektet har det blitt arbeidet frem en løsning på denne problemstillingen: *"Hvordan kan en nettbasert applikasjon utvikles for å effektivt automatisere Windows-servere på AWS, slik at kostnader og ressursbruk minimeres?"*.

Hvor målene for oppgaven gitt av Ambita har vært:

- Automatisert styring av Windows Server-instanser: Vi vil gi regnskapsavdelingen muligheten til å raskt og enkelt starte og stoppe Windows Servere på AWS-plattformen. Dette vil bidra til å optimalisere bruk av ressurser og redusere kostnader.
- Brukervennlig grensesnitt: Den nettbaserte løsningen bør være enkel å bruke, og den vil gi regnskapsavdelingen full kontroll over serverinstansene de har behov for. Et enkelt knappetrykk vil tillate dem å skru av og skru på disse serverne.
- Sikkerhet: Det BØR inkludere funksjoner for sikker start av disse Windows Server-instansene og registrere aktiviteten som blir gjort på disse, hvis dette ikke allerede er inkorporert

Se vedlegg Bacheloroppgave, som er utgitt av Ambita.

Basert på evalueringen i denne rapporten kan det konkluderes med at løsningen har vært vellykket. Det har blitt utviklet en nettbasert fullstack-applikasjon som automatiserer Windows-servere via et brukervennlig grensesnitt. Denne løsningen har bidratt til betydelige kostnads- og ressursbesparelser.

Sikkerheten i løsningen har vært en høy prioritet, og det har blitt implementert tofaktorautentisering og SSO for å sikre at bare autoriserte brukere får tilgang. Videre har det blitt fulgt anbefalt praksis for å minimere feil og sikre en sikker løsning. Dette inkluderer prinsippet om minst mulig rettigheter når det gjelder tjenestetilganger og tillatelser, samt håndtering av andre identifiserte sikkerhetsrisikoer.

Det ble presentert løsningen for Ambita-representanter, og det ble gitt en gjennomgang av applikasjonen og prosessen bak. Det ble demonstrerte funksjonaliteten og gitt forklaring på valgene som hadde blitt tatt og begrunnelsen bak dem. Under presentasjonen fikk representantene komme med innspill og spørsmål, og dette resulterte i at alle involverte parter betraktet prosjektet som vellykket.

Applikasjonen ble tatt i bruk 18. april, og i tillegg til å hjelpe Ambita med automatisering av Windows-servere har prosjektet også løst et annet problem. Gjennom prosjektets utvikling ble det funnet en måte å koble SSO-innloggingssystemer til Amplify-portalen, som var et ønske fra andre team internt i Ambita. Dette har vært et positivt utfall av prosjektet, da det har bidratt til å løse ytterligere utfordringer hos Ambita.

Samlet sett er prosjektet en suksess sett fra Ambitas perspektiv. Vi har levert en løsning som innfrir kravene de stilte til prosjektet.

8 Referanser

[1] AWS «Amazon Web Services» hentet fra https://aws.amazon.com/. (Lastet ned: 16. Februar 2024)

[2] AWS «Amazon Amplify», hentet fra https://aws.amazon.com/amplify/ (Lastet ned: 20. Februar 2024)

[3] Amazon «Amazon API Gateway» hentet fra https://aws.amazon.com/api-gateway/ (Lastet ned: 20. Februar 2024) Microsoft (2024).

[4] AWS «Amazon Elastic Compute Cloud», hentet fra https://aws.amazon.com/ec2/. (Lastet ned: 16. Februar 2024)

[5] AWS «AWS Amazon EventBridge », hentet fra https://aws.amazon.com/eventbridge/ (Lastet ned: 24. Februar 2024)

[6] AWS «Amazon Lambda» hentet fra https://aws.amazon.com/lambda/ (Lastet ned: 20. Februar 2024)

[7] OWASP. (2021). "Cross-Site Request Forgery (CSRF)." Hentet fra https://owasp.org/www-community/attacks/csrf (Lastet ned 26. April 2024).

[8] AWS «AWS Elastic Beanstalk» hentet fra https://aws.amazon.com/elasticbeanstalk/ (Lastet ned: 26. Februar 2024)

[9] AWS «AWS Elastic Load Balancer» hentet fra https://docs.aws.amazon.com/elasticloadbalancing/latest/userguide/what-is-load-balancing.html (Lastet ned: 02. Mai 2024)

[10] Microsoft «Microsoft Entra ID» hentet fra https://www.microsoft.com/en-us/security/business/identity-access/microsoft-entra-id (Lastet ned: 26. Februar 2024)

[11] Meta "React - En JavaScript-bibliotek for å bygge brukergrensesnitt," hentet fra https://reactjs.org (Lastet ned: 5. Mars 2024)

[12] Regjeringen «Fra kalveskinn til datasjø — Ny lov om samfunnsdokumentasjon og arkiver» hentet fra https://www.regjeringen.no/no/dokumenter/nou-2019-9/id2639106/ (Lastet ned: 30. April 2024)

[13] Microsoft «Microsoft Dynamics AX» hentet fra https://dynamics.microsoft.com/en-us/ax-overview/ (Lastet ned 5. Mars 2024)

[14] Compello «Compello» hentet fra https://www.compello.com/ (Lastet ned 5. Mars 2024)

[15] SuperOffice «SuperOffice CRM» hentet fra https://www.superoffice.com/crm/ (Lastet ned 5. Mars 2024)

[16] Google «Material-UI: A popular React UI framework» hentet fra https://material-ui.com// (Lastet ned 26. feb8ruar 2024)

[17] AWS «AWS Amplify UI» hentet fra https://docs.amplify.aws/ui (Lastet ned: 26. Februar 2024)

[18] Microsoft «MSAL.js Documentation» hentet fra https://github.com/AzureAD/microsoft-authentication-library-for-js/tree/dev/lib/msal-browser (Lastet ned: 26. Februar 2024)

[19] AWS, «AWS Cost Explorer» hentet fra: https://aws.amazon.com/aws-cost-management/aws-cost-explorer/ (Lastet ned 28.02.24)

[20] Norges Bank, «Valutakurser» hentet fra: https://www.norges-bank.no/tema/Statistikk/Valutakurser/?tab=currency&id=USD (Lastet ned: 28. Februar 2024)

[21] AWS, «AWS Pricing» hentet fra: https://aws.amazon.com/pricing/?aws-products-pricing.sort-by=item.additionalFields.productNameLowercase&aws-products-pricing.sort-order=asc&awsf.Free%20Tier%20Type=*all&awsf.tech-category=*all (Lastet ned: 28. Februar 2024)

[22] AWS «AWS Cloudwatch» hentet fra https://aws.amazon.com/cloudwatch/ (Lastet ned: 5. Mars 2024)

[23] AWS «AWS Inspector» hentet fra https://aws.amazon.com/inspector/ (Lastet ned: 5. Mars 2024)

[24] AWS, «Amazon EBS Volumes» hentet fra: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-volumes.html (Lastet ned: 28. Februar 2024)

[25] AWS, «Amazon EBS Snapshots» hentet fra: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSSnapshots.html (Lastet ned: 28. Februar 2024)

[26] AWS, «Amazon DyamoDB» hentet fra: https://aws.amazon.com/dynamodb/ (Lastet ned: 28. Februar 2024)

[27] AWS «Amplify pricing» hentet fra https://aws.amazon.com/amplify/pricing/ (Lastet ned: 5. Mars 2024)

[28] AWS «AWS instance scheduler» hentet fra https://aws.amazon.com/solutions/implementations/instance-scheduler-on-aws/ (Lastet ned: 5. Mars 2024)

[29] GitHub «GitHub» hentet fra https://github.com/ (lastet ned: 26. Februar 2024)

[20] AWS «AWS Identity and Access Management» hentet fra https://aws.amazon.com/iam/ (lastet ned: 26. Februar 2024)

[31] Statista. «Most used web frameworks among developers worldwide, as of 2023» Hentet fra https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/ (Lastet ned: 26. April 2024)

[32] Mircosoft «Office 365 Essentials» hentet fra https://www.microsoft.com/en-us/office/365/office-365-essentials (Lastet ned: 26. Februar 2024)

[33] Trello «Trello - Organize Anything» hentet fra https://trello.com/ (Lastet ned: 26. Februar 2024)

[34] Meta «Messenger» hentet fra https://www.messenger.com/ (Lastet ned: 26. Februar 2024)

[35] Microsoft «Outlook» hentet fra https://www.microsoft.com/en-us/outlook-com/ (Lastet ned: 26. Februar 2024)

[36] Slack Technologies «Slack» hentet fra https://slack.com/ (Lastet ned: 26. Februar 2024)

[37] Figma «Figma» hentet fra https://www.figma.com/ (Lastet ned: 26. Februar 2024)

[38] Microsoft «Visual Studio Code» hentet fra https://code.visualstudio.com/ (Lastet ned: 26. Februar 2024)

[39] Mozilla «JavaScript» hentet fra https://developer.mozilla.org/en-US/docs/Web/JavaScript (Lastet ned: 26. Februar 2024)

[40] JSON «Introducing JSON» hentet fra https://www.json.org/json-en.html (Lastet ned: 16. April 2024)

[41] Python «Python 3.8.0» hentet fra https://www.python.org/downloads/release/python-380/ (Lastet ned: 16. April 2024)

[42] Git. «About Version Control» hentet fra https://git-scm.com/ (Lastet ned 26. Februar 2024)

[43] OpenAI «ChatGPT» hentet fra https://chat.openai.com/ (Lastet ned: 15. Mars 2024)

[44] Ambita «Farger» hentet fra https://design.ambita.com/ambita/#/colors (Lastet ned: 28.Februar, 2024)

[45] Uutilsynet «Websites» hentet fra https://www.uutilsynet.no/english/websites/906 (Lastet ned 28.Februar, 2024)

[46] Snook, J. «Colour Contrast Check» hentet fra https://snook.ca/technical/colour_contrast/colour.html#fg=FFFFFF,bg=3600766 (Lastet ned 28. Februar 2024)

[47] World Wide Web Consortium «Web Content Accessibility Guidelines» hentet fra https://www.w3.org/WAI/standards-guidelines/wcag/ (Lastet ned: 28. Februar 2024)

[48] Planview «What is Scrumban?» hentet fra https://www.planview.com/resources/guide/what-is-scrum/lkdc-what-is-scrumban/ (Lastet ned: 10. Mai 2024)

[49] Scrum.org «What is Scrum?» hentet fra https://www.scrum.org/resources/what-is-scrum (Lastet ned: 28. Februar 2024)

[50] Atlassian «Kanban» hentet fra https://www.atlassian.com/agile/kanban (Lastet ned: 10. Main 2024)

[51] AWS «AWS Cloud Practitioner Essentials» hentet fra https://aws.amazon.com/training/classroom/aws-cloud-practitioner-essentials/ (Lastet ned: 16. Februar 2024)

[52] OneLogin «How Does Single Sign-On Work» hentet fra https://www.onelogin.com/learn/how-single-sign-on-works (Lastet ned 25.April 2024)

[53] Auth0 “What is OAuth 2.0?” hentet fra https://auth0.com/intro-to-iam/what-is-oauth-2 (Lastet ned: 03.April 2024)

[54] AWS «AWS Identity and Access Management» hentet fra https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_examples_ec2_tag-owner.html (Lastet ned: 10. April 2024)

[55] Microsoft «Microsoft Authentication Library for JavaScript (MSAL.js)» hentet fra https://learn.microsoft.com/en-us/javascript/api/overview/msal-overview?view=msal-js-latest (Lastet ned: 10. Mai 2024)

[56] Ambita «Profilhåndboken» hentet fra https://design.ambita.com/ambita/#/ (Lastet ned: 28. April 2024)

[57] J. Nielsen, Nielsen Norman Group. «Ten Usability Heuristics» Hentet fra https://www.nngroup.com/articles/ten-usability-heuristics/ (Lastet ned 16. April 2024)

[58] AWS «AWS State Machines» hentet fra https://docs.aws.amazon.com/step-functions/latest/dg/amazon-states-language-state-machine-structure.html (Lastet ned: 12. April 2024)

[59] Mozilla «Same-origin policy» hentet fra https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy (Lastet ned 12. April 2024)

[60] Mozilla «Cross-Origin Resource Sharing (CORS)» hentet fra https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS (Lastet ned 16. April 2024)

[61] Ambita «Ambita» hentet fra https://www.ambita.com/ (Lastet ned: 26. April 2024)

[62] OpenAI «OpenAi status» hentet fra https://status.openai.com/ (Lastet ned: 26. April 2024)

9 Vedlegg

- Vedlegg 1. Prosjekthåndbok
- Vedlegg 2. Kravdokument
- Vedlegg 3. Visjonsdokument
- Vedlegg 4. Systemdokumentasjon
- Vedlegg 5. Oppgavebeskrivelse fra Ambita