Høgskulen på Vestlandet

Fakultet for teknologi, miljø- og samfunnsvitenskap
Institutt for datateknologi, elektroteknologi og realfag

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

Systemdokumentasjon

Marius Bråthen Reikerås

Sivert Åkernes Sæter

Henrik Halvorsen

Versjon <2.0>

Revisjonshistorie

Dato	Versjon	Beskrivelse	Forfatter
<15.04.2024>	<1.0>	Første iterasjon på dokumentet	Henrik Halvorsen, Marius Reikerås, Sivert Sæter
<12.05.2024>	<2.0>	Siste iterasjon før innlevering	Henrik Halvorsen, Marius Reikerås, Sivert Sæter

Innholdsfortegnelse

1 Innledning

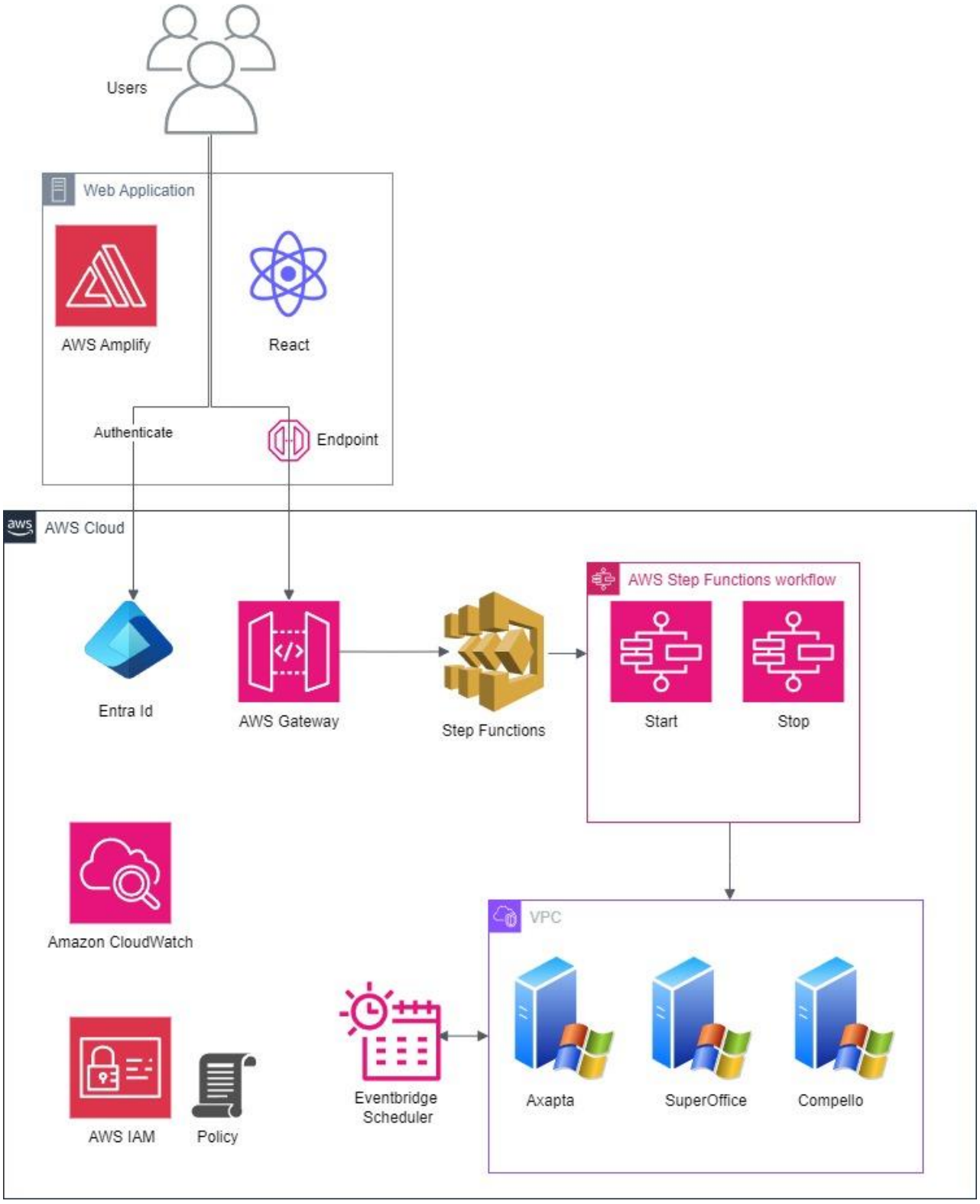
Dette dokumentet utgjør systemdokumentasjonen for applikasjonen utviklet som en del av bachelorprosjektet*: Skybasert Automatisering av Windows-Servere: En effektiv løsning for Ambitas Regnskapsavdeling*

Dokumentet er ment å gi et innblikk i systemet prosjektet har utviklet. Og hvordan det er satt opp og designet. I tillegg inkluderes ulike tekniske aspekter slik som arkitektur, avhengigheter og sikkerhet.

Dokumentet er videre delt opp i 7 deler:

- Arkitektur, et bilde av overordnet arkitektur for prosjektets løsning.
- Prosjektstruktur, pakkestrukturen for prosjektet og intern struktur av prosjektet
- Databasemodell, hvordan har lagring foregått i prosjektet og for løsningen
- Servertjenester, hvilke servertjenester er brukt i løsningen
- Sikkerhet, hvilke sikkerhetstiltak og sikkerhet under utviklingen av produktet
- Installasjon og kjøring, hvilke bibliotek, versjoner og programmer er nødvendig
- Dokumentasjon av kildekode, kildekode for ulike aspekter av frontend-koden.
- Kontinuerlig integrasjon og testing, hvordan har løsningen blitt testet underveis i prosjektet og hvilke tester er laget
- Referanseliste skrevet på IEEE format

2 Arkitektur



Figur 2.1 - viser bilde av arkitekturen til løsningen

Web Application:

- React:

React er ett åpent JavaScript-bibliotek for å utvikle brukergrensesnitt. Det er en av mest populære biblioteketene for front-end utvikling [1]. Dette biblioteket er utnyttet for å utvikle brukergrensesnittet og applikasjonen som den er.

- Github

Github er en plattform bygd for versjonskontroll og som samarbeidsplattform for utviklere. Det er hvor all kode i prosjektet blir lagret [2].

- AWS Amplify

Amplify er en av AWS sine tjenester som gjør det enklere å bygge og deployere applikasjoner til skybaserte systemer. Amplify er koblet opp mot Github-repositoriet hvor koden er lagret [3]. Koden i Amplify oppdateres automatisk når det er en ny versjon av koden lastet opp på GitHub.

- Entra ID (Azure Active Directory)

Entra ID er et skybasert identitets- og tilgangstyringstjeneste fra Microsoft Azure. Det blir brukt til styre tilgang til applikasjonen. Dette blir gjort gjennom tilgangskontroll og flerfaktorautentisering [4].

AWS Cloud:

- AWS API Gateway

Gateway er en av AWS sine tjenester for å lage APIer. Det er satt opp ett REST-api for Amplify plattformen som fungerer som en inngangsdør fra applikasjonen til back-end operasjoner [5]. Dette blir gjort gjennom API-endepunkter koblet opp til forskjellige AWS-tjenester. For prosjektet er det satt opp endepunkter til Step Functions og Lambda.

- AWS Lambda

Lambda er serverless datatjeneste fra AWS som tillater databehandling og kjøring av kode. Lambdafunksjonene er satt opp til å gjøre systemkall til Windows-serverne relatert til automatiseringen [6].

- Amazon CloudWatch

CloudWatch er en monitorerings- og observeringstjeneste tilbudt av AWS. Den henter data, logg filer og hendelser som skjer i AWS ressurser [7].

- AWS IAM

IAM (Identity and Access Management) er en av AWS sine tjenester for å tilordne tilgang til ressurser og ressursene sine tilganger [8]. Det er laget en policy som inneholder alle tilgangsrettigheter nødvendig for å utføre handlinger og tilgang til tjenester. Denne blir lagt til en bruker som tildeles til hver av de nødvendige tjenestene for å gi korrekt tilgang.

- AWS Step Functions

AWS Step Functions er et visuelt verktøy for arbeidsflyt som utviklere kan bruke til å bygge distribuerte applikasjoner, orkestrere mikrotjenester og automatisere prosesser [9].

- Eventbridge Scheduler

Eventbridge Scheduler er en AWS-tjeneste som er laget som en påminningshjelp eller en hendelsesbasert hjelpetjeneste. Enkelt forklart kan denne tjenesten brukes til å utføre en påminning eller gjøre noe vilkårlig på bestemte tidspunkt [10]. I prosjektets tilfelle er det satt opp sjekker på serverstatus i tilfelle serverne ikke er blitt slått av. Slik at det ikke forekommer unødvendige økonomi- og ressurskostnader.

- VPC (Virtual Private Cloud) [11] Dette er kun en visuell ramme for Windows-serverne som angår prosjektet.

3 Prosjektstruktur

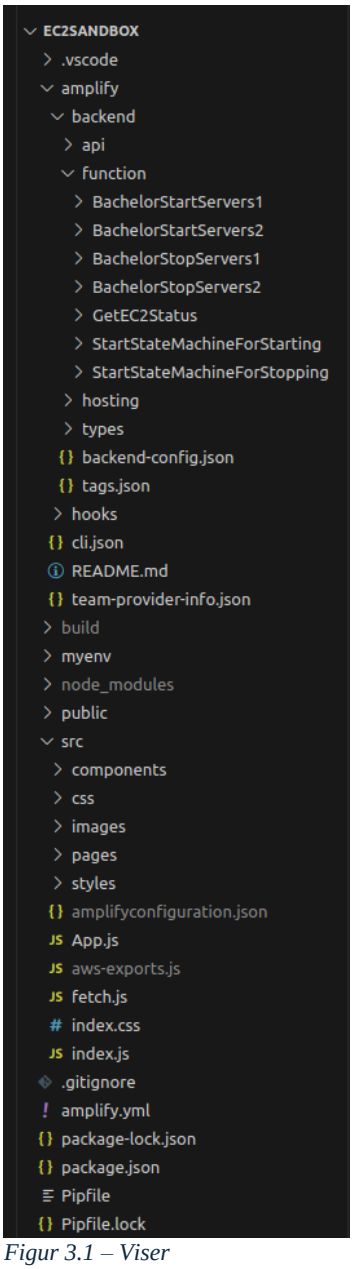
En overordnet katalog kalt “EC2SANDBOX” hvor de mappene som er verdt å nevne er “Amplify” og “src”.

I mappen Amplify er all struktur satt opp ved Amplify kommandolinjeverktøyet. Den inkluderer kode for alt som er utviklet på serverside for applikasjonen. Dette inkluderer API-et for prosjektet og alle lambdafunksjoner.

I mappen src er komponenter, css og sider satt opp i individuelle mapper.

Bibliotekene som er brukt er:

- MUI (Material-UI), åpent React UI bibliotek basert på Google’s Material Design [12].
- Amplify UI, AWS Amplify sitt bibliotek for komponenter til brukergrensesnitt [13].
- Ambita Design, en nettside som tilbyr fargekoder, tekstformat og ulike grafiske elementer relatert til Ambita AS [14].



Figur 3.1 – Viser prosjektstrukturen

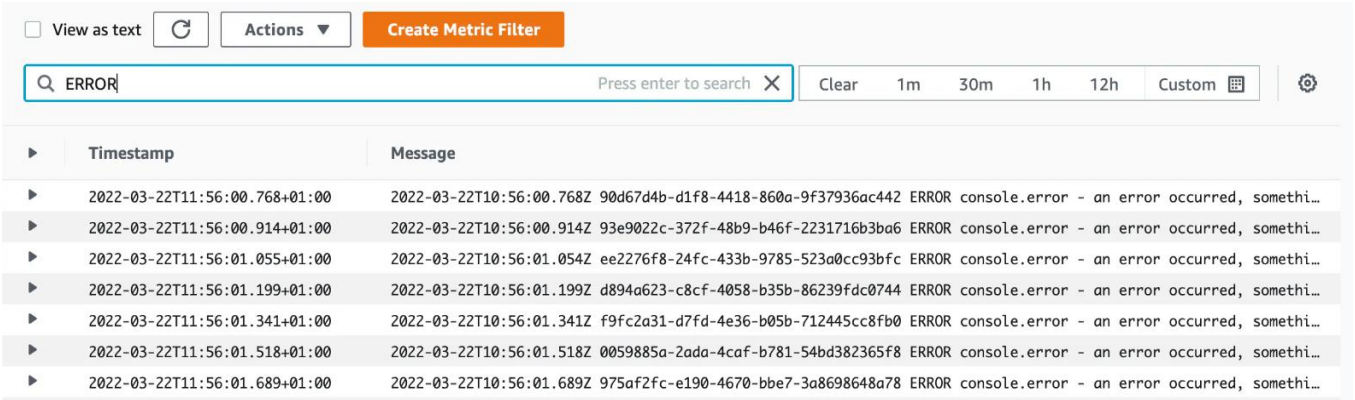
4 CloudWatch Logs og lagring av kode

Det er ikke satt opp en database for prosjektet. Lagring og versjonshåndtering av kildekode er gjort gjennom Github og Git [15]. All kode for prosjektet er lagret på Ambita sin github plattform. Årsaken til at det ikke er brukt noen database for prosjektet er fordi det ikke har vært nødvendig å lagre data for applikasjonen. Det eneste tilfelle av lagring av data er for interaksjoner og bruk av applikasjonen.

Dette er håndtert gjennom CloudWatch Logs [16]. CloudWatch Logs er AWS sin løsning på å overvåke, lagre og aksessere logger for AWS-systemer. CloudWatch Logs er aktivt brukt i prosjektet. Det er brukt til følgende:

- Statuskode og tilbakemelding for lambdafunksjoner
- Feilmeldinger til lambdafunksjoner
- Utregning av gjennomsnittlig kjøretid for lambdafunksjoner
- Logger for testing av API-endepunkter
- Logger for testing av State Machines
- Logging av hendelsesrekkefølge

CloudWatch Logs er konfigurert til alle lambdafunksjonene i prosjektet og API-et gjennom Amplify. Her er et bilde av hvordan slike logger kan se ut figur 4.1 [18]. Bilde er ikke relatert til prosjektet av sikkerhetsmessige årsaker.

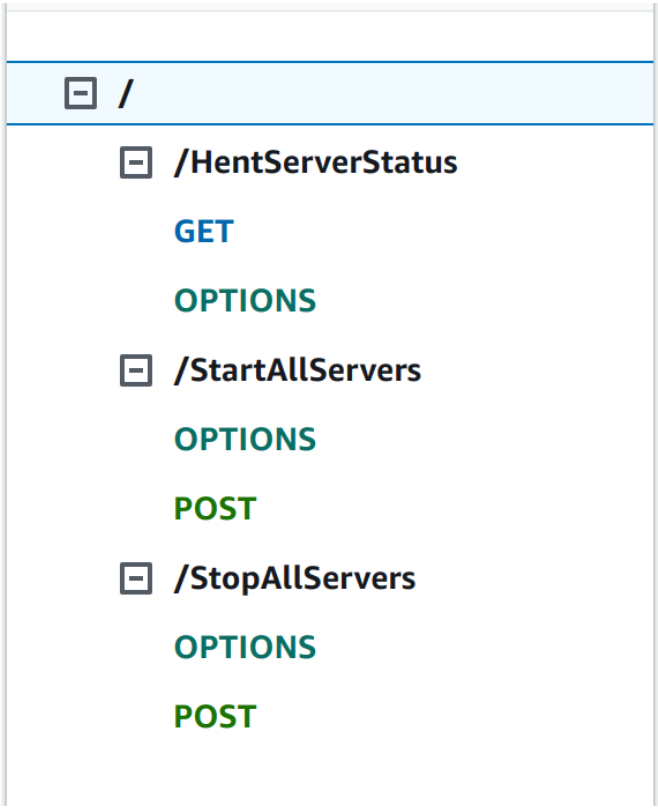


Figur 4.1 - Eksempel over CloudWatch logger

5 Server-tjenester

Det er satt opp ett REST-api gjennom AWS tjenesten API Gateway. Dette API-et er en essensiell del av den tekniske løsningen. API-et er broen mellom applikasjonen og de forskjellige ressursene vi bruker til å starte og stoppe serverne.. Denne broen blir bygget opp av REST-endepunkter.

Det er satt opp tre endepunkter, hvor hvert endepunkt er knyttet opp mot applikasjonen gjennom hendelsesbehandlere. Det ene endepunktet er knyttet til når klient logger inn og autentiseres vellykket. Dette endepunktet bruker REST sin GET metode for å starte en lambdafunksjon som henter status på alle serverne, og leverer status til applikasjonen. De to andre endepunktene er koblet opp mot knapp i applikasjonen som står for stopp eller start av serverne. Endepunktene kjører hver sin State Machine som starter eller stopper serverne. Disse endepunktene bruker REST sin POST metode da det skal gjøres endringer på serverne sin status. Hvert endepunkt er konfigurert med CORS tillatelser til applikasjonen sin URL for å tillate API-kall fra domenet. Nedenfor er ett bilde av API-strukturen figur 5.1.



Figur 5.1 - Oversikt over endpoints for APIet

6 Sikkerhet

En sentral sikkerhetsutfordring i mange applikasjoner er håndtering av tilgang og innloggingssystemer. For å forbedre sikkerheten betraktelig, kan tofaktorautentisering (2FA) implementeres, som krever to ulike former for identitetsbekreftelse ved innlogging, noe som gjør uautorisert tilgang vanskeligere. Ambita bruker allerede Microsoft Entra [1] for dette formålet, noe som forenkler integrasjonen av 2FA i deres eksisterende systemer gjennom Ambitas Microsoft-kontoer.

I tillegg er Identity and Access Management (IAM) en annen sikkerhetsmekanisme som kontrollerer brukertilganger til tjenester på en sikker måte, og sikrer at kun autoriserte brukere får tilgangen til sensitiv informasjon og systemer. Ambita bruker prinsippet om "least responsibility" når det gjelder tillatelser gjennom IAM, dette vil si at hver tjeneste eller bruker kun får tilgang til det minimale de trenger for å oppnå ønsket funksjonalitet.

CORS (Cross-Origin Resource Sharing)-policyer spiller en viktig rolle i å opprettholde sikkerheten i webapplikasjoners arkitektur ved å kontrollere hvilke domener som kan interagere med applikasjonens API. CORS–policyene implementeres igjennom API Gateway, her kan en kontrollere hvilke domener som kan gjøre kall til APIet.. Dette forsterker sikkerheten ved å hindre uautoriserte kryssdomene forespørsler.

Ved å implementere Microsofts autentiseringsbibliotek Msal.js [4] og ved å følge sikkerhetsstandardene OAuth 2.0 [5] og OpenID, beskyttes applikasjonen effektivt mot vanlige trusler som XSS (Cross-Site Scripting) og ‘man-in-the-middle' angrep. Samtidig øker Single Sign-On (SSO) brukervennligheten ved å tillate brukerne å logge inn én gang for tilgang til flere applikasjoner. Sikkerheten styrkes ytterligere med flerfaktorautentisering og nøye tilgangsstyring gjennom IAM i AWS-miljøet.

7 Installasjon og kjøring

Nedenfor er det listet opp en rekke biblioteker og avhengigheter til prosjektet samt hvilken versjon som ble brukt i prosjektet.

NPM, Node Package Manager: er en pakkeadministrator for Javascript kjøretidsmiljøet Node.js
Versjon: 10.2.4
Lisens: Artistic License 2.0
Installasjonskommando: npm install npm@latest -g

Node.js: brukt til å kjøre JavaScript på serversiden.
Versjon: 20.11.0
Lisens: MIT
Installasjonskommando: Download Node.js v20.13.1 (Node.js Offisielle nettside)

React: Et JavaScript-bibliotek for å bygge brukergrensesnitt. Brukes for å bygge den interaktive frontenden av applikasjonen.

Versjon: 18.2.0.
Lisens: MIT
Installasjonskommando: npx create-react-app "my-app"

Python: Brukt for lambda funksjonene
Versjon: 3.8
Lisens: The Python Software Foundation
Installasjonskommando: pip install python==3.8

AWS Amplify: Et sett med verktøy og tjenester for mobil og front-end webutviklere for å bygge fullstakks serverløse applikasjoner på AWS.
Versjon: 6.0.18.
Lisens: Apache 2.0
Installasjonskommando: npm install aws-amplify @aws-amplify/ui-react

Aws-amplify/ui-react: Enkelte React komponenter fra aws sitt react bibliotek
Versjon: 6.1.5.
Lisens: Apache 2.0
Installasjonskommando: npm install @aws-amplify/ui-react

Material UI er et open-source React komponent-bibliotek. Biblioteket tilbyr et stort utvalg av gjenbrukbare komponenter som skjemaer, knapper, ikoner, navigasjonsfelt og layouts og mye mer.
Versjon: 5.15.10.
Lisens: MIT
Installasjonskommando: npm install @mui/material@5.15.10

MSAL.js gjør slik til at en enkelt kan implementere autentisering og autorisasjon med Microsofts sin Identitetsplattform i JavaScript-applikasjoner.
Versjon: 3.14.0
Lisens: MIT
Installasjonskommando: npm install @azure/msal-browser

AWS CLI, et verktøy for å administrere AWS-tjenester direkte fra terminalen eller kommandolinjen.
Versjon: 2.15.16.
Lisens: Apache
Installasjonskommando: `curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64-2.15.16.zip" -o "awscliv2.zip" && unzip awscliv2.zip && sudo ./aws/install`

AWS Amplify CLI er et kommandolinjeverktøy for å lage, integrere og administrere AWS sine skytjenester. Amplify abstraherer bort kompleksiteten for mye av AWS tjenestene.
Version: 12.11.0

Lisens: Apache-2.0

Installasjonskommando: npm install -g @aws-amplify/cli

8 Dokumentasjon av kildekode

I dette kapittelet vil det bli forklart et par kodesnutter som vi har fått lov til å vise av bedriften. Disse kodesnuttene anses ikke som sensitive for bedriften. Kodesnuttene vil hovedsakelig fokusere på klientsiden av løsningen. Målet er å gi et lite innblikk i hvordan komponentene er implementert.

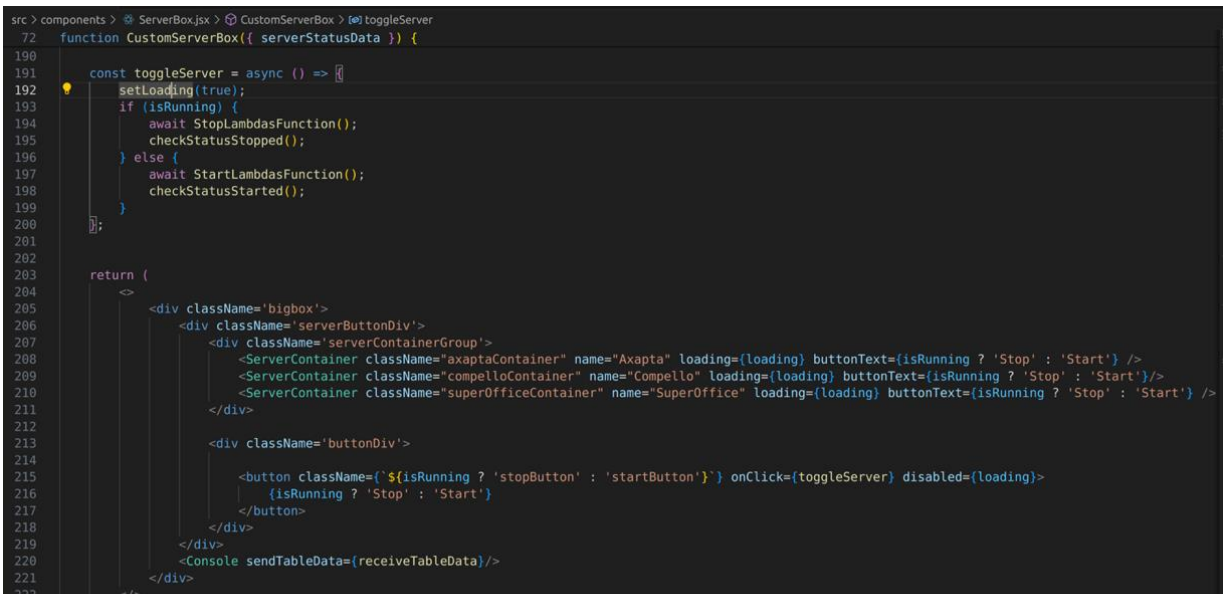
Denne kodesnutten, figur 8.1, viser hvordan den egendefinerte komponenten «ServerContainer» er implementert. Denne komponenten inneholder navnet på serveren, en statustekst og en Loader-komponent som er lagt innenfor en div-tagg. CSS-klassen til komponenten kalt «loaderDivOutside» er standard. CSS-klassen endrer seg dersom teksten på knappen har verdien «Stop» og Loader-komponenten har verdien false, (!loading). Hvis dette er tilfellet er sant vil CSS-klassen endre seg til en CSS-klasse kalt «loading» og ingen klasse hvis tilfellet ikke er sant. Det er implementert tre slike komponenter som representerer serversystemene Axapta, SuperOffice og Compello. I figur 8.1 er det vist hvordan disse tre komponentene er implementert sammen med knappen i hovedkomponenten.

src > components > ServerContainer.jsx > ServerContainer

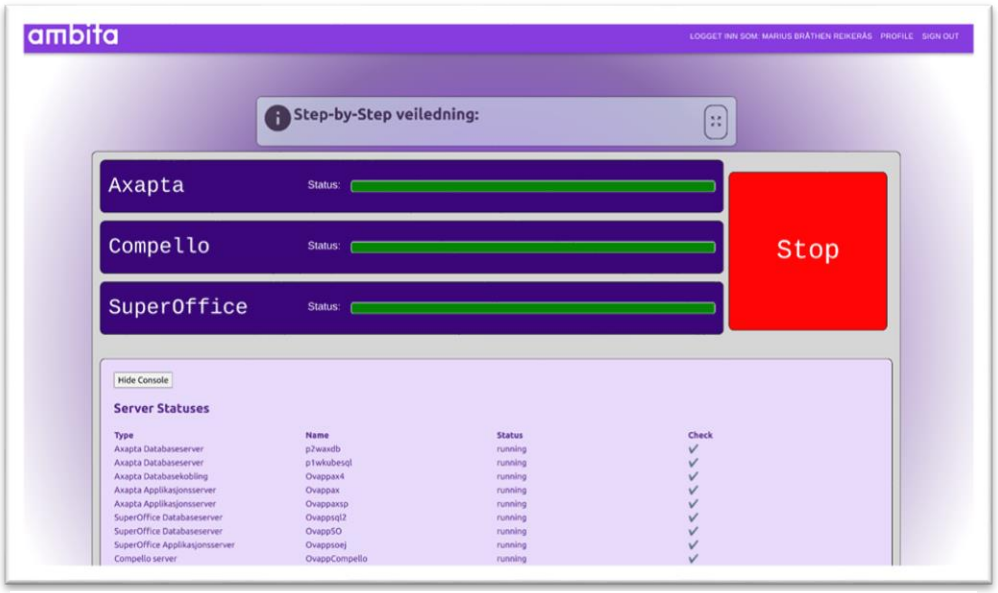
```jsx
import React from 'react';
import '../css/ServerBox.css';
import Loader from './Loader';

function ServerContainer({ name, loading, buttonText }) {

  return (
    <>
      <div className="serverContainer">

        <span className='serverTxt'>{name}</span>
        <span className='statusTxt'>Status:</span>
        <div className={`loaderDivOutside ${buttonText === 'Stop' && !loading ? 'loading' : ''}`}>

          <div className='loaderDiv'>
            {loading && <Loader />}
          </div>
        </div>

      </div>
    </>
  );
}
export default ServerContainer;
```

Figur 0.1 - Egendefinert komponent, ServerContainer

I figur 8.2 blir den egendefinerte hovedkomponenten «CustomServerBox» implementert. Den inneholder tre av de egendefinerte komponentene «ServerContainer». Som forklart i tidligere kodesnutt, tar disse komponentene inn parameteren «navn» og statusen til komponenten Loader. Denne hovedkomponenten inneholder de tre serverkomponentene og en felles knapp-komponent. Denne knapp-komponenten har en funksjon kalt «toggleServer» som blir kalt når knappen registrerer en onClick-hendelse. Denne funksjonen er definert over denne hovedkomponenten. Funksjonen setter verdien (setLoading) til true, som sørger for at Loader-komponenten begynner å laste. Hvis (isRunning) er tilfellet, blir «StopLambdasFunction» kallet på. Denne kaller da på tilstandsmaskinen som sørger for å stoppe serverene. Gjennom denne prosessen blir «checkStatusStarted()» kallet på. Denne er ansvarlig for å hente statusen på serverne gjennom oppstart og avslåing for å kunne gi en visuell bekreftelse gjennom brukergrensesnittet. Statusfeltet blir så oppdatert hvis alle serverne har startet opp eller blitt skrudd av. Komponenten «Console» implementert nederst i bildet, har ansvaret for å vise statusen til serverne i form av en tabell. Brukergrensesnittet er vist i figur 8.3 for å gi en lettere forståelse av hvordan komponentene henger sammen.



Figur 0.2 - Implementasjon av hovedkomponenten



Figur 0.3 - Bilde av frontend-løsning

```
export const Home = () => {

    return (
        <>
            <AuthenticatedTemplate>
                <InfoDiv />
                <AuthenticatedContent />
            </AuthenticatedTemplate>


            <UnauthenticatedTemplate>
                <Typography variant="h4">Logg inn for å få tilgang til serverportalen for Axapta, Compello og SuperOffice.
            </UnauthenticatedTemplate>

        </>
    );
}
```

Figur 0.4 - Funksjonalitet til implementasjonen av innlogging

Figur 8.4 viser hvordan MSAL.JS biblioteket fungerer. Home-komponenten har ansvaret for å kun sørge for å vise innhold hvis man er logget inn eller ikke. Hvis man er logget inn, vil komponentene <InfoDiv> og <AuthenticatedContent> vises. Hvis man ikke er logget inn, vises det en tekst som sier at du må logge inn for å tilgang.

9 Kontinuerlig integrasjon og testing

Prosjektet benytter seg av AWS Amplify for å håndtere kontinuerlig integrasjon og kontinuerlig leveranse (CI/CD) av koden. CI/CD betyr "Continuous Integration og Continuous Deployment", som er fleksible metoder og verktøy som bygger bro mellom utvikling og drift av programvare. Amplify er koblet opp med Github repostory til prosjektet og hver endring som gjøres i prosjektets Github-repository automatisk blir bygget og testet hver gang det blir pushet eller merget til hovedgrenen.

AWS Amplify samarbeider med Github-repositoryet, slik at hver endring automatisk utløser en bygg- og testprosess. Når byggeprosessen starter, utfører Amplify en rekke tester på koden for å sikre at den oppfyller kvalitetsstandardene våre før den blir distribuert. Denne automatiserte testprosessen bidrar til å oppdage og rette feil tidlig i utviklingsprosessen, noe som bidrar til å forbedre kvaliteten på koden og øke hastigheten på leveransene våre.

Gjennom utviklingsprosessen testet vi hver enkelt komponent og tjeneste separat for å bekrefte deres funksjonalitet før de ble integrert og testet på nytt som en helhet i PROD miljøet til Ambita. Dette ble gjort igjennom et eget miljø hos Ambita´s AWS-kontoer kalt "sandbox". Her ble det testet ut enkelte AWS tjenester og eksperimenterte med dem i trygge omgivelser uten fare for å ødelegge for Ambita sine eksisterende ressurser. Her ble hver komponent testet for seg selv, før de ble kombinert og testet på nytt. Etter hvert ble det laget en fullstendig applikasjon som ble testet før den ble gjenskapt i produksjonsmiljøet til Ambita.

Denne prosessen var spesielt viktig siden AWS-tjenestene og miljøet var et nytt område for gruppen, og det var viktig å sikre at alt fungerte som det skulle, selv med mindre erfaringer.

10 Referanser

- [1] React «React» hentet fra https://react.dev/ (Lastet ned: 15.Mars 2024)
- [2] Github «Github» hentet fra https://github.com/ (Lastet ned: 15.Mars 2024)
- [3] Amplify «AWS Amplify» hentet fra https://aws.amazon.com/amplify/ (Lastet ned: 15.Mars 2024)
- [4] Microsoft Entra ID «Entra ID» https://www.microsoft.com/nb-no/security/business/identity-access/microsoft-entra-id (Lastet ned: 15.Mars 2024)
- [5] AWS API Gateway «API Gateway» hentet fra https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html (Lastet ned: 15.Mars 2024)
- [6] AWS Lambda «Lambda» hentet fra https://aws.amazon.com/lambda/ (Lastet ned: 15.Mars 2024)
- [7] AWS CloudWatch «CloudWatch» hentet fra https://aws.amazon.com/cloudwatch/ (Lastet ned: 15.Mars 2024)
- [8] AWS IAM «Identity and Access Management» hentet fra https://aws.amazon.com/iam/ (Lastet ned: 15.Mars 2024)
- [9] Step Functions «AWS Step Functions» Hentet fra https://aws.amazon.com/step-functions/ (Lastet ned: 23.April 2024)
- [10] Eventbridge Scheduler «Introducing Amazon Eventbridge Scheduler» hentet fra https://aws.amazon.com/blogs/compute/introducing-amazon-eventbridge-scheduler/ (Lastet ned: 23.April 2024)
- [11] Amazon Virtual Private Cloud «Amazon Virtual Private Cloud» hentet fra https://aws.amazon.com/bl[ogs/compute/introducing-amazon-eventbridge-scheduler/ (Lastet ned: 23.April 2024)
- [12] MUI «Material UI» hentet fra https://mui.com/material-ui/react-progress/ (Lastet ned: 15. Mars 2024)
- [13] Amplify UI «Amplify UI» hentet fra https://ui.docs.amplify.aws/ (Lastet ned: 15.Mars 2024)
- [14] Ambita AS «Ambita Profilhåndboken» hentet fra https://design.ambita.com/ambita/#/logo/ambita (Lastet ned: 15.Mars 2024)
- [15] Git. «About Version Control» hentet fra https://git-scm.com/ (Lastet ned 26. Februar 2024)

- [16] 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)
- [17] Auth0 “What is OAuth 2.0?” hentet fra https://auth0.com/intro-to-iam/what-is-oauth-2 (Lastet ned: 03.April 2024)
- [18] CloudAsh«Use log levels in CloudWatch Logs to generate filter-friendly logs» hentet fra https://cloudash.dev/blog/cloudwatch-log-levels