Universitetet i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

BACHELOROPPGAVE

Studieprogram/spesialisering: Bachelor i ingeniørfag / Datateknologi	Vårsemesteret 2022 Åpen
Forfatter(e): Henrik Nilsen, Mari Hånsnartredet Bjordal	
Fagansvarlig og veileder: Erlend Tøssebro	
Tittel på bacheloroppgaven: Nettbutikk for IoT-sensorer med kobling til LoRaWAN Engelsk tittel: Webshop for IoT-sensors with connection to LoRaWAN	
Studiepoeng: 20	
Emneord: IoT, ThingPark, LoRaWAN, Nettbutikk, Azure, Vue, Python	Sidetall: 83 + vedlegg/annet: 13 Stavanger 15. mai 2022

Sammendrag

Formålet med oppgaven er å utvikle en nettside som lar brukerene kjøpe IoT-sensorer som automatisk kobler seg til et *Long Range Wide Area Network (LoRaWAN)*. LoRaWAN er et langdistanse nettverk som blir mye benyttet av IoT-produkter. IoT står for *Internet of Things*, som også blir kalt for tingenes internett. Applikasjonens hovedfunksjon er å gjøre Altibox sitt sensornettverk lett tilgjengelig som en kommersiell tjeneste. Da kan flere benytte sensornettverket for egne sensorer. Gruppen ble også oppfordret til å implementere brukervennlige funksjonaliteter på nettbutikken slik at den kan fungere som et *Proof of Concept*. Da kan Lyse bruke nettbutikken som utgangspunkt for videreutvikling av tjenesten.

Nettsiden tar i bruk teknologier fra ThingPark for å koble sensorene til LoRaWAN, og distribueres gjennom Microsoft Azure DevOps. Det blir benyttet Azure Secret Key Vault for å lagre nøkler og passord sikkert, mens annen data blir lagret i Azure SQL Database. Resultatet er et helhetlig produkt som dekker alt fra funksjonalitetene til en nettbutikk til en automatisk prosess som håndterer registrering av enheter opp mot LoRaWAN, nettverksabonnementer og lisenser.

Dropbox link som inneholder videoer av nettsiden og all koden, link: `https://www.dropbox.com/sh/o6ucwqp7bh5ckpd/AACX_9QmS2RdZK5w3jovPazia?dl=0`

Innhold

Forord

Vi ønsker å takke Erlend Tøssebro som har vært fagansvarlig og veileder for oppfølging og samtaler under hele prosjektet.

Vi vil goså takke Ronny Lorentzen som jobber for Lyse for hjelp underveis i oppgaven. Han har vært tilgjengelig på Teams ved små spørsmål og nyttig på digitale og fysiske møter ved større gjennomganger.

Kapittel 1

Innledning

1.1 Problemstilling

Oppgaven går ut på å utvikle en webshop som Lyse AS kan ta i bruk ved salg av sine sensorløsninger for IoT-gjenstander. Hovedformålet til nettbutikken er at kunder automatisk kan få sensorer ferdig satt opp på Altibox sitt sensornettverk etter et kjøp. For selve nettbutikken betyr det at man både må håndtere produktlinjer, som for eksempel sensorer, og mer kompliserte tilleggsfunksjoner som gir kunden abonnement på datatrafikk via sensornettverket. Dette gjøres ved en integrasjon opp mot sensornettverksplattformen ThingPark der eventuelle kundeforhold og abonnement blir aktivert automatisk.

Sensornettverket er en utvidelse av fibernettverket og nettsiden skal gi kunder kommersiell IoT-aksess. Nettsiden vil baseres på ulike Azure løsninger for produksjonsdistribusjon og også oppfylle krav for tilleggsfunksjonaliteter som er forventet av en brukervennlig nettbutikk.

1.2 Mål

Hovedfokuset til oppgaven går ut på å gi kunder IoT-aksess til Altibox sitt sensornettverk, slik at flere kan bruke sensorer til privat bruk. Dette skal oppnås ved å utvikle en plattform hvor kunden kan bestille ulike sensorer og nettverksabonnementer på en enkel og brukervennlig måte. Nettsiden vil også ha flere sekundære mål som dekker ulike krav for administrativ tilgang. Dette omfatter overvåkning av tilkoblede enheter, kontakt med varelager og lagerstatus, og generelle krav for en velfungerende nettside. Oppgaven vil være å lage en *Proof of Concept* (POC) nettside som skal fungere som en slags inspirasjon for Lyse. Den skal vise at det er mulig å gjennomføre en nettbutikk som automatisk registrer sensorer på sensornettverket. Lyse kan bruke dette og bygge videre på det om de finner løsningen interessant og aktuell. Målene for den ferdige oppgaven kan oppsummeres med punktene under.

- Utvikle en brukervennlig nettbutikk med nødvendige funksjonaliteter og innlogging.
- Opprette en automatisert prosess som behandler kundeforhold, lisenser og registrering av enheter til LoRaWAN.
- Distribuere nettsiden ved hjelp av Azure DevOps.

1.3 Motivasjon

Hver eneste dag øker antall av IoT-løsninger og fysiske gjenstander som blir koblet opp mot sensornettverket. Lyse sitt arbeid innen IoT handler om å lage tjenester som kan redusere kostnader og øke effektiviteten for innbyggere, kommuner og næringsliv [1]. Nettbutikken vil gjøre disse tjenestene tilgjengelig for folk flest og bidra til økt utvikling av denne typen tjenester. Dersom løsningen av oppgaven virker lovende, vil Lyse kunne bruke denne til å videreutvikle webshopen og i beste utfall ta løsningen i bruk. Hovedmotivasjonen til oppgaven er at løsningen har et potensiale til å bli starten på et nytt system som etterhvert kan bli brukt av flere mennesker internt i Lyse på en ukentlig basis. I beste fall kan dette utvikle seg til å bli en portal

som gir hele den norske befolkningen lettere tilgang til de tjenestene som Lyse tilbyr.

Det er mange eksempler på hvordan konnektivet via ThingPark kan endre fremtidens digitale løsninger. ThingPark er en global leder innenfor IoT-tilkoblingsløsninger. Med industrielle IoT-løsninger, et LoRaWAN-nettverk og ThingPark kan man for eksempel hjelpe med å møte fremtidens byutfordringer. Applikasjonen i oppgaven bruker nettopp disse hjelpemidlene til å distribuere ut disse industrielle midlene og løsninger til både privatpersoner og bedrifter. Løsningene blir dermed tilgjengelig for alle, og det kan gjøre det lettere for initiativtagere å gjennomføre prosjekter som forbedrer samfunnet. Dette gjelder blant annet redusering av forurensing, forbedre trafikkstyring og optimalisering av avfallsinnsamling i en moderne smartby. Kanskje kan nettbutikken være med på å påvirke hvordan man i fremtiden forholder seg til IoT-verdenen og løsningene dette tilbyr. [2]

1.4 Arbeidsmetode

Det populære versjonskontrollprogrammet Git ble brukt sammen med tjenesten GitHub for å forbedre måten gruppen samarbeidet på. Git ble valgt, fordi det har flere funksjonaliteter som blant annet gjør det mulig å lage greiner og forespørsler som andre må godkjenne før en grein kan slås sammen med hovedgreinen. Den andre personen på gruppa må altså lese gjennom koden til den andre og godkjenne den. Dette gjør at feil oppdages raskere, i tillegg til at det øker forståelsen til den andre personen og oppdaterer det andre gruppemedlemmet på hva som er gjort. Git inkluderer også muligheten til å opprette milepæler og en oppgavetavle som gjør det lettere å ha oversikt over mål og arbeidsoppgaver. Det gjør også samarbeid og deling av kode enklere. Gruppen har brukt en utviklingsmetode som baserer seg på Scrum, der ukentlige spurter og møter står sentralt. De ukentlige spurtene gjør det lettere å forholde seg til små, ukentlige frister og gir en god oversikt over hva som bør gjøres og prioriteres. Dette forhindrer også dårlig kommunikasjon og uoversiktlighet.

Når ny funksjonalitet er ferdigstilt, blir koden sendt til et Azure Repos. Azure Repos bruker versjonkontrollprogrammet Git, så derfor funker det bra å ha et repository på Github for utvikling og et repository på Azure Repos

for produksjon. Der er det pipelines som automatisk publiserer koden til både frontend og backend når koden i Azure Repo'et blir endret. Dette gjør det ekstremt effektivt å publisere ny funksjonalitet til produksjonsnettsiden.

1.5 Om bedriften

Lyse er en norsk industribedrift med fokus innen telekommunikasjon, infrastruktur og energi. Konsernet er eid av fjorten kommuner, og har som et hovedfokus på å gi tilbake til fellesskapet.[3]

Nasjonalt har konsernet blitt en seriøs aktør innen fiberbasert bredbånd. De leverer også produkter og tjenester under merkevarene Altibox og Smartly. Siden 2018 har Altibox bygget ut IoT-sensornettverk i Norge, og har per i dag en dekning til over 1.000.000 husstander fordelt på over 100 kommuner. [3] [4]

Konsernet har hovedkontor i Stavanger med virksomhet i Norge og i Danmark og med over 1300 ansatte. [3]

Kapittel 2

Bakgrunn

Hvilke teknologier en nettside bruker, har mye å si for arbeidsflyt og brukeropplevelse. Noen valg er blitt gjort for å optimalisere arbeidsflyten under utvikling og vedlikehold av nettsiden, mens andre valg har hovedfokus på brukeropplevelsen. Noen valg har Lyse tatt på forhånd. Teknologiene Lyse har valgt er de samme teknologiene som Lyse bruker på sine prosjekter. Da vil det være lettere for Lyse å videreutvikle og bruke produktet dersom de velger å gjøre dette.

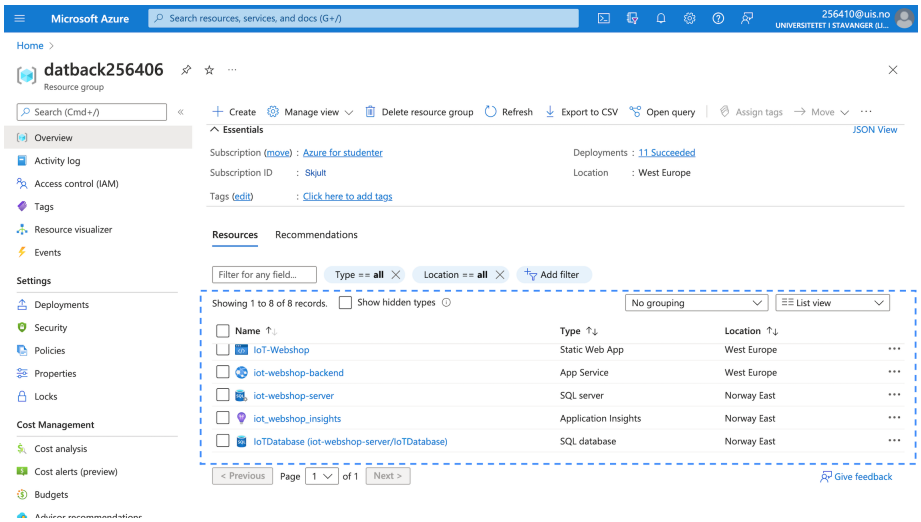
Applikasjonen vil fungere som en live demo som lett visualiserer hvordan man kan utvikle en løsning som automatiserer det manuelle arbeidet som kreves for å registrere sensorer opp mot sensornettverket. Det er derfor viktig å prioritere at brukeropplevelsen skal bli enklest mulig, og at logikken skal fungere på en så effektiv måte som mulig uten for mange innspill fra brukeren. Dette kapittelet vil gi en kort introduksjon til alle de sentrale konseptene og valgene som er gjort i prosjektet, slik at det kan danne et grunnlag for forståelsen for hvordan oppgaven er gjennomført.

2.1 Microsoft Azure

Microsoft Azure er en skyplattform som består av over 200 tjenester [5]. Flere av disse tjenestene bruker Lyse fra før, så det var krav fra Lyse å

bruke Microsoft Azure til blant annet distribusjon av nettsiden. I tillegg er det gunstig å samle flere av tjenestene som blir brukt hos samme leverandør. Da blir det lettere å holde kontroll på tjenestene og eventuelle utgifter. Valgene av andre nødvendige tjenester, som for eksempel nøkkelhvelv og database, er tatt basert på det å kunne samle de ulike tjenestene i samme portal, i stedet for å ha mange forskjellige aktører på samme prosjekt.

Med Microsoft Azure blir alle tjenestene samlet i Azure-portalen, noe som er svært gunstig og effektivt når man skal lage et produkt som skal benytte seg av flere tjenester. Der er det svært enkelt å lage de forskjellige tjenestene ved bruk av portalen. Azure-portalen er en portal som viser alle tjenestene til Azure som man bruker. Den inkluderer også funksjonaliteter som kan endre på tjenestene eller legge til nye tjenester. Figur 2.1 viser ressursgruppen som er opprettet i Azure-portalen. I den blå, stiplede boksen vises fem av de åtte ressursene som tas i bruk. De siste tre er "iotwebshopkeyvault", *App Service*-planen "ASP-datback256406-83d9" og et delt dashbord. De viktigste Azure-produktene som blir brukt vil bli forklart i de kommende delkapitlene.



Figur 2.1: Azure-portalen viser ressursgruppen. Fem av de åtte Azure-tjenestene prosjektet bruker er vist i den blå, stiplede boksen.

Alle studenter kan få en bruker med 100 USD på Azure-kontoen dersom de bruker universitetsmailen sin når de registrerer seg. Microsoft er også veldig gode på dokumentasjonen sin, noe som også gjelder Microsoft Azure. Godt

dokumenterte tjenester gjør det lettere å lære nye teknologier, og Microsoft har et godt rykte på seg for kvaliteten og brukbarheten av tjenestene sine. Dette gjør Azure til et logisk valg uansett om det hadde vært et krav fra Lyse eller ei.

2.1.1 Azure Web App

Backenden til nettsiden blir distribuert med Azure Web App. Dette var et krav fra Lyse sin side. Under utvikling av nettsiden brukes gratisversjon av Azure Web App, men dersom nettsiden skal bli brukt senere kan man skalere den med tjenester som dekker behovet for den ønskede bruken. Fordelen med å bruke Azure Web App er at det et skybasert, samtidig som det fungerer godt sammen med andre Azure-tjenester. Appen er enkel å implementere med Azure DevOps ved bruk av kontinuerlig distribusjon. Dette blir diskutert videre i kapittel 2.1.6.

Det er også lett å implementere andre Azure tilleggstjenester slik som secrets og databaser. Det å lage selve nettsiden er veldig enkel når man bruker Azure-portalen. Da vil Azure lage en nettside automatisk for deg. Det vanskeligste er å publisere koden til nettsiden, noe man gjør ved bruk av Azure DevOps som nevnt i delkapittel 2.1.6. Backenden kjører på Python 3.9, som er den nest nyeste Python-versjonen. Grunnen til at backenden kjører Python, er fordi dette var et krav fra Lyse sin side. Det er dette de har erfaring med og ønsker å bruke dersom de skal videreutvikle nettsiden. Hadde backenden kjørt på et annet språk, måtte de ha skrevet om hele koden.

2.1.2 Azure Static Web App

Frontenden til nettsiden blir distribuert med Azure Static Web App. Dette var også et krav fra Lyse. Hovedforskjellen med Azure Static Web App, sammenlignet med Azure Web App, er at den er designet spesielt for frontend med tanke på å levere statiske filer. Det er også designet spesielt til React, Angular og Vue. Sistnevnte er det frontenden benytter seg av. Med Azure Static Web App er det lett å integrere med andre Azure tjenester. Grunnen til at frontend bruker Azure Static Web App foran Azure Web App, er fordi frontenden med Vue hovedsakelig er statiske filer. Da er det mer gunstig å

bruke en statisk side. Her blir det også brukt en gratis utviklingsversjon, men det er fullt mulig å skalere nettsiden til produksjonsnivå med ekstra kostnader. Man lager en statisk nettsiden nesten helt likt som en normal nettside ved hjelp av Azure-portalen. Å publisere kode blir også gjort ved hjelp av Azure DevOps som nevnt i 2.1.6. Frontenden kjører på node 14.18.

2.1.3 Azure Key Vault

Azure Key Vault er en tjeneste av Azure som tilbyr lagring av hemmelige nøkler og secrets. Azure Key Vault blir benyttet for å ikke måtte eksponere innloggingsinformasjon og hemmelige nøkler i koden. Ved hjelp av Azure Key Vault kan man bruke Lyse sine personlige passord og klientider uten at det står i klartekst i koden. Dette reduserer sjansene for at hemmeligheter lekkes ved et uhell. Sikkerheten øker også, siden hvelvet krever riktig autentisering og autorisasjon for at en bruker eller en applikasjon kan få tilgang til nøkkelhvelvet. [6]

Det er enkelt å sette opp Azure Key Vault og legge til nøkler ved hjelp av Azure-portalen. Det viktige her er å huske på å gi alle tjenestene som benytter seg av secret tilgang til Azure Key Vault. For eksempel trenger både frontend og backend tilgang til nøklene. Dersom en av dem ikke har tilstrekkelig tillatelse, vil de ikke kunne hente nøklene og derfor ikke fungere. Det er to tillatelser som applikasjonene trenger for å håndtere nøklene. De trenger *Get* og *List* tillatelse på *Secret Management Operations*. Applikasjonene tregner begge disse tillatelse for å hente ut nøkler.

2.1.4 Azure SQL Database

Lyse ga gruppen mulighet til å velge hvilken database som skulle brukes. Det eneste innspillet var at det skylle være en relasjonsdatabase. Da ble Azure SQL Database valgt av flere grunner. Azure SQL Database er en relasjonsdatabase i skyen som også integreres eksepsjonelt med de andre Azure-tjenestene. I prosjektet benyttes det en billig utviklingsversjon, men den har full mulighet til å skalere seg til produksjonsnivå med høyere krav. Dette vil føre til økte kostnader.

Å endre databasen kan gjøres både i Azure-portalen og ved hjelp av Python-kode som blir beskrevet i underkapittelet 2.1.5. Der er viktig å merke seg at Azure SQL Database benytter seg av port 1433. Det er flere kommersielle nettverk som har blokkert denne porten. Det vil forekomme et problem når nettsiden blir kjørt lokalt under utvikling og skal koble seg opp mot databasen. Man vil ikke få tilgang til databasen uten denne porten. Hvis porten er blokkert, må man enten benytte seg av mobilt nettverk, eller VPN, for å unngå dette problemet. På nettverket hjemme vil det fungere uten problemer. Produksjonsnettsidene har ikke problem med porten siden de blir kjørt av en Azure-tjener.

SQLite var et annet alternativ som ble vurdert. Fordelen med SQLite er at den implementerer en rask, liten og fullkompatibel SQL database som blir lagret som en database fil. SQLite er også gratis, noe som er en fordel. Grunnen til at valget falt på Azure SQL Database, og ikke SQLlite, er fordi SQLite er et filbasert databasesystem, mens Azure SQL Database er skybasert. En skybasert database vil være mer effektiv på store databaser enn et filsystem. Det var også svært gunstig at Azure SQL Database passer så bra sammen med de andre Azure-tjenestene.

2.1.5 Pyodbc

For å kunne bruke Python-kode til å benytte seg av databasen, trenger man en tilleggspakke kalt Pyodbc. Pyodbc er en av de enkleste tilleggspakken Python har for å koble seg opp mot databasen. Med denne pakken kan man endre databasen, legge til data og hente ut data.

For at Pyodbc skal virke må også datamaskinen som kjører Python-kode ha en ODBC-driver. ODBC står for *Open Database Connectivity*. Denne nettsiden benytter seg av ODBC versjon 17. ODBC er en driver som tilbyr en standard API for håndtering av databasen, med mål om å være lik på tvers av programmeringsspråk og operativsystemer [7]. Denne driveren ble valgt, fordi den er utviklet av Microsoft til å fungere bra sammen med Azure SQL Database. Det var derfor logisk å ta i bruk løsningen som er utformet spesifikt for dette formålet, og ikke en annen tjeneste fra en annen leverandør.

Det er svært gunstig at det fungerer både lokalt, på tjeneren og på flere

plattformer. Det gjør det lett for Lyse eller andre å eventuelt benytte seg av databasen ved videreutvikling av nettsiden. Dersom man vil endre litt av databasen er det bare å fjerne tabellen. Deretter legge til tabellen med nesten samme Python-kode, men med ønsket modifikasjoner.

2.1.6 Azure DevOps

Bruk av Azure DevOps for kildekodehåndtering var et av kravene til Lyse. Azure DevOps bidrar med kontinuerlig integrering og kontinuerlig distribusjon. Dette bidrar til bedre arbeidsflyt og kort tid for at ny funksjonalitet skal komme ut i produksjon. Her blir det benyttet to forskjellige pipelines for å publisere koden til frontend applikasjonen og backend applikasjonen. Grunnen til dette er fordi det er to helt forskjellige prosjekter som skal på to forskjellige tjenere. Azure DevOps har mal for pipelines, men man må endre dem til å passe til hver enkel applikasjon. Man må f.eks endre versjonen av programmeringsspråket som blir brukt, og hvor nettsiden skal bli distribuert. Azure Secret brukes i Frontend pipeline-en for å skjule API-nøkkelen. Azure DevOps bruker også et eget git repository kalt Azure Repos. Der ligger koden som produksjonsnettsiden benytter seg av.

2.2 Vue

Vue er valgt som frontend JavaScript rammeverk, fordi det er noe gruppemedlemmene er godt kjent med fra før. Ikke bare gir det en fordel siden alle vet hvordan man utnytter teknologien best mulig, men Vue kommer også med CLI- og GUI-verktøy som er ideelt for å starte et prosjekt. Dette inkluderer Vuex-støtte og *material design* biblioteket Vuetify. Vue gjør det også enkelt å implementere en SPA ved hjelp av vue-router. SPA står for *Single Page Application*. En ensideapplikasjon gjør opplevelsen av nettsiden mer brukervennlig og dynamisk, da enkeltsideapplikasjonen oppdaterer den eksisterende siden dynamisk i stedet for å laste helt nye sider fra tjeneren.

En av prioriteringene i oppgaven er at det skal være lett for brukere å ta i bruk produktet. Da er det viktig at valget av rammeverket forbedrer brukeropplevelsen, og at funksjonalitetene innenfor rammeverket støtter behovene

applikasjonen har for brukervennlighet. En SPA var et av behovene som blir ansett som viktig, siden applikasjonen blir raskere. Vue er også et av de mer populære rammeverkene, noe som gjør at dokumentasjonen er bra. Rammeverket blir også vedlikeholdt jevnlig, slik at den inneholder få til ingen feil.

Det finnes også andre kandidater som var aktuelle ved valg av Vue som rammeverk. Både Angular og React er populære rammeverk som er kjent for sin gode ytelse. Selv om alle rammeverkene er basert på komponenter og tillater lett opprettelse av nye egenskaper på nettsiden, har alle forskjellig arkitektur og struktur. Applikasjonen har ingen spesielle behov som kun kan tilfredsstilles av det ene eller det andre rammeverket, så valget ble til slutt basert på at Vue var kjent fra før og gruppen visste at rammeverket ville dekke alle behovene.

2.2.1 Vue CLI

Vue CLI fungerer som en standard verktøygrunnlinje for Vue-økosystemet. Dette sørger for at de ulike verktøyene som blir brukt sammen vil fungere uten problemer ved hjelp av fornuftige standardinnstillinger som blir laget av verktøygrunnlinjen. Fordelen med dette er at man slipper å ta hensyn til konfigurasjonene, da dette blir fikset for deg ved hjelp av konfigurasjonsmulighetene som Vue CLI tilbyr. Vue CLI inneholder også en stor samling av offisielle tilleggsmoduler som inkluderer gode verktøy i frontend-økosystemet. [8]

Tidligere hadde gruppemedlemmene erfaring med å sette opp Vue-ruteren, Vue-applikasjonen og det meste av konfigureringene selv. Da det ble bestemt at frontenden skulle inkludere ulike rammeverk som hadde behov for mange nye konfigureringer, fant gruppen fort ut at Vue CLI var ideelt for å knytte alle verktøyene sammen uten feil. I det man oppretter et prosjekt med Vue CLI får man et valg om å spesialtilpasse prosjektet manuelt etter behov, i stedet for å velge de vanlige innstillingene. Dette er mer tilpasset for et produksjonorientert prosjekt slik som denne oppgaven. Fleksibiliteten til Vue CLI lot oss velge versjonen av Vue, velge å inkludere TypeScript, velge å inkludere Vue-ruter, velge å inkludere Vuex-butikk og velge formatering. Hele appen blir da tilpasset vårt behov med fungerende konfigurasjonsfiler.

Det var viktig å kunne velge versjonen av Vue, da Vuetify-rammeverket kun er kompatibel med versjon 2 per i dag. Det finnes en betaversjon av Vuetify 3, men den er kun for ment for testformål, og er ikke ment for produksjons-applikasjoner. Dette er derfor ikke aktuelt for dette prosjektet. [9]

Selv om Vue 3 er raskere og lettere enn Vue 2, i tillegg til at det inneholder en forbedret støtte til TypeScript og nye funksjoner, var det fortsatt viktigere å kunne bruke Vuetify. Dette skyldes fokuset på brukeropplevelsen. Det ble antatt at Vue 3 ikke ville gi merkbar forskjell i hastighet for denne typen applikasjon. Ved en eventuell overgang til Vue 3, vil fortsatt alt på nettsiden fungere etter overgangen uten at man trenger å gjøre noen endringer. Det er dermed ingen direkte negative sider ved å ikke bruke Vue 3. Da har man fortsatt muligheten til å oppgradere enkelt til Vue 3 når Vuetify får støtte for dette. [10]

2.2.2 Vuetify

Vuetify er et komplett UI-rammeverk som er bygget på toppen av Vue. Grunnen til at Vuetify tas i bruk, er fordi det er et godt verktøy for å bygge detaljerte og engasjerende brukeropplevelser.[11] Rammeverket tar inspirasjon fra Vue sin bruk av ulike komponenter, og gir utvikleren tilgang til flere hundre, ulike varianter av forskjellige komponenter. Disse gir et godt helhetsinntrykk for brukeren.

Alle disse komponentene kan tilpasses etter eget behov, samt settes sammen for å skape unike design. Vuetify gjør det også lett å tilpasse de ulike komponentene til ulike størrelser på ulike skjermer, slik at det lett fungerer ut av boksen på forskjellige skjermstørrelser. Man kan også oppnå et bra resultat raskere ved bruk av dette rammeverket. Mange av komponentene som tas i bruk er standard for flere typer applikasjoner, og når de fungerer bra, så er det ikke nødvendig å skulle lage noe helt likt på nytt selv. Det er heller ikke like nødvendig med ekstremt gode designferdigheter. Rammeverket er også lettere og mindre tidkrevende enn å bare bruke HTML og CSS.

Det finnes andre UI-rammeverk for Vue som var aktuelle. Et av disse rammeverkene er BootstrapVue. Dette rammeverket baserer seg mye på versjon 4 av Bootstrap og har tilgjengelig komponenter som bygges opp i Vue-stil med Bootstrap grunnlag [12] Selv om Bootstrap er kjent fra før, og kom-

ponentene virket lovende, følte gruppen at Vuetify var et bedre valg. Dokumentasjonen til Vuetify var bedre og mer oversiktlig. I tillegg inneholder dokumentasjonen til Vuetify en sandkasse-funksjonalitet som lar deg endre på de ulike komponentene i sanntid inne i dokumentasjonen. Dette gjorde det lettere å visualisere mulighetene og forstå hvordan man best utnyttet funksjonene som rammeverket hadde å tilby. Det virket også lettere å avvike fra det standard utseende til komponentene, slik at applikasjonen kunne bli mer personlig enn det man hadde oppnådd med BootstrapVue.

Et annet alternativ var Buefy. Dette rammeverket hadde lignende sandkasse-funksjonaliteter som det var i Vuetify dokumentasjonen, men mulighetene for å endre på komponentene var begrenset og antall tilgjengelige komponenter var mye færre enn med Vuetify. Buefy er også basert på Bulma, noe som gruppen ikke hadde hørt om før. Det gjorde det også mindre atraktivt enn Vuetify. [13]

Element UI, et annet rammeverk, hadde litt av det samme utgangspunktet som Buefy. Det var ikke like stort og det var også begrensninger for design. Selv om Vuetify baserer seg på versjon to av Vue, er det også noe Element UI gjør. [11][14] Fordelene av å velge Vuetify ble derfor mange, og begrensningene ved å måtte bruke Vue-versjon to vil ikke ha noen konsekvenser for opptredenen til programmet.

2.2.3 Vuex

Den største fordelen Vuex-verktøyet gir, er muligheten til å implementere en ”butikk”. Hovedfunksjonen til butikken er at den beholder tilstanden din. En Vuex-butikk skiller seg fra et vanlig globalt objekt ved at en Vuex-butikk er reaktiv. Når en Vue-komponent henter tilstand fra butikken, vil komponentene raskt oppdateres hvis butikkens tilstand endres. Vuex tillater også organisering av data i ulike moduler slik at man kan holde alt ryddig og vedlikeholdbart. [15] Dette er for eksempel perfekt å bruke for å dele informasjon som skal være reaktivt mellom ulike komponenter. For eksempel til autentisering av brukere og deres tilganger.

Vuex er også skreddersydd for Vue og det er et lett å legge til via Vue CLI. Vuex har også støtte for Typescript, og siden det passet så godt inn med de andre valgene ble det ikke naturlig å se etter andre alternativer.

2.3 Typescript

TypeScript er et statisk programmeringsspråk som bygger på det kjente språket JavaScript, og som har fokus på typen til variablene. Typen kan for eksempel være streng eller heltall. En av de største fordelene med TypeScript er at man må definere type til variablene. Det gjør feilsøking lettere, og koden blir fort bedre strukturert. Når variabler ikke har en definert type kan man fort få uventet resultat av forskjellige operasjoner. Dersom man ikke vet hvilken type en variable er når man definere den, kan man benytte seg av typen *any*. *Any*-variabelen kan være alle TypeScript typer.

En annen fordel med TypeScript er at man må definere hva en funksjon returnerer. Dersom funksjonen ikke returnerer noe, må man presisere at den returnerer *void*. Dette vil også forebygge uventende feil, fordi man vet hva en funksjon returnerer. Koden vil ikke kompilere dersom man prøver å gi en variabelen en verdi fra en funksjonen som returnerer feil type. I JavaScript vil koden kompilere, men man vil plutselig få en uventet feil en annen plass i koden som opprinnelig stammer fra at variabelen har fått en verdi med feil type.

2.4 Python Flask

Et av kravene Lyse kom med innebar bruken av Python Flask. Dette ble da valgt slik at applikasjonen lettere kan videreutvikles med hensyn til bedriftens kunnskaper og systemer. Dersom applikasjonen skal integreres med deres systemer, er det en fordel at denne applikasjonen støtter de teknologiene som bedriften bruker. Dette er også noe gruppen er kjent med fra før. En annen grunn til at Flask ble valgt, var fordi det er et lite og lett Python-nettverk som gir nyttige funksjoner og verktøy som gjør det enklere å lage nettapplikasjoner ved hjelp av Python. Flask gjør det også enkelt å utvide applikasjonen og det krever ingen komplisert standardkode for å komme i gang med prosjektet. [16]

2.5 IoT-aksess

LoRaWAN-sensornettverket som Altibox har bygget ut i Norge, og som sensorene skal bli koblet opp mot, har i dag dekning til over 1.000.000 husstander i 100 kommuner. Dette nettverket blir fortsatt utvidet den dag i dag. LoRaWAN står for "Long Range Wide Area Network" og er et nettverk som er designet for å kommunisere over lange avstander samtidig som det bruker lite energi. Det bruker radiosignaler til å kommunisere med, og i EU bruker det 863-873 Mhz.

Siden informasjonen blir sendt med radiosignaler har alle mulighet til å få tak i informasjonen som blir sendt. Derfor blir denne informasjonen kryptert med algoritmen AES128. AES128, også kjent som *Advanced Encryption Standard*, er en symmetrisk krypertingsalgoritme som krypterer til 128 bits. Det finnes forskjellige versjoner av AES, der AES128 bruker 128 bits nøkler. AES er veldig populær, og grunnen til dette er fordi den er både rask og sikker. Nettverket er i tillegg dupleks, har god sikkerhet og enhetene kan være mobile samtidig som det tilbyr lokasjonstjenester. Dette gjør LoRaWAN ekstremt gunstig for ett IoT-nettverk. [17] [18]

Nå som nettverket blir tilbudt som en kommersiell tjeneste, må kunder ha muligheten til å koble seg til dette nettverket. IoT-aksessen i denne oppgaven er en kombinasjon av sensorkommunikasjon, en IoT-plattform og sensornettverket. [1] Oppgaven fokuserer på å gjør dette lettest mulig for kunden ved å ordne alle de tekniske kravene i bakgrunnen slik at kunden ikke trenger å fokusere på dette.

For å håndtere logistikken med lisenser, tilgang og oppkobling brukes IoT-plattformen ThingPark sammen med API-en ThingPark DX API. DX API plattformen inneholder blant annet DX API-ene DX Core og DX Admin. Disse blir aktivt brukt i oppgaven. Applikasjonen skal altså bruke en tredjepart for å automatisere prosessen som skjer fra man velger en sensor til den får IoT-aksess.

2.5.1 ThingPark

Lyse bruker ThingPark-plattformen for å kontrollere og observere abonnentene, IoT-enhetene og abonnementene. ThingPark er en tilkoblingsadministrasjonsløsning som gir mulighet til å utgi og kontrollere et sikkert LoRaWAN nettverk. [19]

Thingpark er en tredjepart som brukes av Lyse for å koble sine løsninger opp mot LoRaWAN-økosystemet. Dette fungerer altså som et mellomledd mellom nettbutikken og selve sensornettverket som enhetene skal bli koblet opp mot. Nettverket, og ThingPark som en tjeneste, har stort fokus på potensialet til industrielle IoT-løsninger, og bruker den nyeste ThingPark-teknologien til å yte et best mulig resultat for dem som tar i mot tjenesten.

Tjenesten ThingPark samler alle komponentene, slik som sensorer, porter og data, og blir styrt av et unikt og brukervennlig grensesnitt. ThingPark er veldig fleksibelt og kan brukes til både *Proof of Concept* og til produksjonsnivå. Dette er svært gunstig for denne oppgaven med tanke på at nettsiden fungerer som en *Proof of Concept*, men skal ha muligheten til å skalere til et ferdig produkt i framtiden. [19]

2.5.2 ThingPark DX API Platform

ThingPark DX API Platform er et sentralt tilgangspunkt for å kunne bruke de ulike ThingPark DX API-ene. En DX API gjør det mulig å bruke funksjonene til ThingPark OS og ThingPark Wireless gjennom REST API. API-ene setter et sterkt fokus på enkelthet, de beste standardene og den beste praksisen. ThingPark OS gir funksjonaliteter knyttet til brukere og bestillinger, mens ThingPark Wireless gir tilgang til enhetsadministrasjonen og en trådløs logger. Den ene av API-ene som brukes i oppgaven, DX Admin API, brukes til tokenhåndtering og andre DX plattform funksjoner som håndterer administrative oppgaver. DX Core API, blir brukt til kontohåndtering, og til å tilby abonnementer og enhetshåndtering. [20]

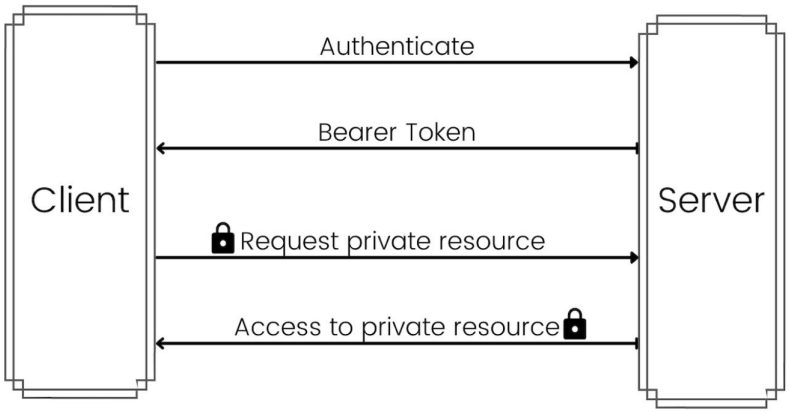
Alle DX API-ene bruker de beste praksisene for REST API-er og inkludere blant annet et Swagger-UI som gir en standard beskrivelse av API-ene som tas i bruk. API-ene bruker standard HTTP-verb og HTTP-svarkoder for

å håndtere kommunikasjon og responser, samt versjonsbehandling, for å lettere kunne migrere på klientsiden. Alle DX API-ene er også tilstandsløse, det vil si at de ikke er avhengige av noen form for informasjonskapsler for å gi brukervennlighet og god skalerbarhet. [20]

API-ene bruker OAuth2 til en tokenbasert autorisasjonsarbeidsflyt for å kunne bruke API-ene. Gjennom autorisasjonsflyten får ulike roller med ulike omfang rettigheter til å gjennomføre ulike handlinger. Det innebærer at man først må få en tilgangstoken for å kunne bruke API-en og spesifikke endepunkter. Token-endepunktet er der applikasjonen sender en forespørsel om å få en tilgangstoken til en bruker. Tilgangen varer så lenge tokenen ikke har utløpt eller har blitt tilbakekalt. For å generere en token, må man bruke en GET /oauth/token-forespørsel til DX Admin API-en. Når man genererer en token må man koble forespørselen til en ThingPark-tjenerforekomst og legge til en klientid og passord for en ThingPark-brukerkonto. Klientiden har formatet "ThingPark-profil/ThingPark-innlogging".[20]

Tilgangstokenen er det applikasjonen bruker for å sende API-forespørsler på vegne av brukerne som bruker applikasjonen og leverandørbrukeren. Denne representerer autorisasjonen til ThingPark for å få tilgang til bestemte deler av brukerens data på plattformen. Tilgangstokenen er formatert som en streng. Denne strengen vil klientapplikasjonen bruke i en HTTP-forespørsel. Disse tilgangstokenene skal alltid være konfidensielle under transport og lagring. Det er kun selve applikasjonen, autorisasjonstjeneren og ressurstjeneren som skal kunne se tilgangstokenen. Den må derfor lagres på en sikker måte som ikke gjøre den tilgjengelig for andre applikasjoner på den samme enheten. For at tredjeparter ikke skal kunne få tak i tokenen, må alle forespørslene sendes over en kryptert HTTPS-tilkobling. Alle API-kall er derfor opprettet med HTTPS. [21]

Diagrammet nedenfor illustrerer hvordan man må autentisere seg med en vellykket pålogging for å motta en bærertoken fra tjeneren. Det er denne bærertokenen som brukes i påfølgende samtaler for å få tilgang til de private ressursene og metodene som API-ene tilbyr. Bærertokenen er laget ut i fra tilgangstokenen på formatet "Bearer <din_genererte_token>". Uten bærertokenen vil tilgangen være nektet for brukeren. Man trenger også en autorisasjonstoken for å kunne bruke SwaggerUI-dokumentasjonen.



Figur 2.2: Figuren viser hvordan applikasjonen mottar en bærertoken for å få tilgang til ressursene til ThingPark API-en. Figuren er hentet fra Freecodecamp [22]

Hver bruker må altså ha en autorisasjonstoken som inneholder en tilgangstoken med en bestemt utløpsdato for å kunne gjennomføre handlinger på ThingPark. De to rollene som blir brukt i prosjektet er leverandør og abonnent. En leverandørrolle har for eksempel muligheten til å opprette andre sluttbrukere med leverandørstatus og opprette brukere med abonnentrolle. I tillegg kan en leverandør opprette nye lisenser, som også kalles for tilbud, samt legge til en bestilling for en bestemt abonnent.

Abonnentrollen kan blant annet opprette nye enheter til sin egen bruker. Dette er noe en leverandør ikke kan. Enkelte funksjoner som å hente eksisterende tilbud, eller hente brukere, opererer forskjellig ut i fra hvilken tilgangstoken som blir gitt som bærertoken til API-en. Ved en GET-funksjon for å hente ut tilbud, vil man motta en liste over tilbud innenfor det autoriserte omfanget som brukeren har. En leverandør vil derfor kunne hente ut alle tilbudene fra leverandøren, mens en abonnent vil hente ut alle tilbudene som abonnenten abonnerer på.

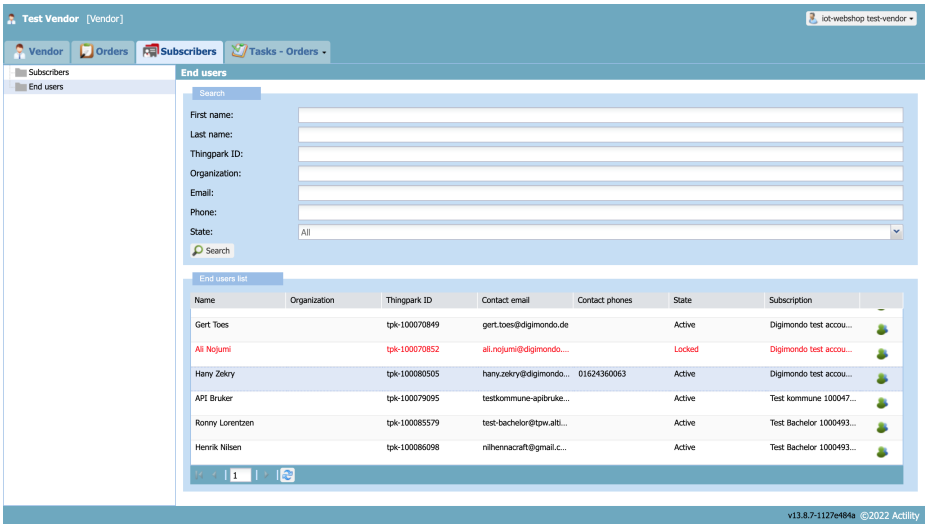
2.5.3 Testmiljø

Lyse har gitt gruppen tilgang til en testbruker med leverandørrolle på ThingPark. Denne brukeren brukes til å logge seg inn på ThingPark sin ad-

ministrasjonsside på `https://altibox.thingpark.com/admin/`. Brukeren har de samme funksjonalitetene som vanlige administratorbrukere normalt sett har, bare at den ikke har tilgang til hovednettverket til Lyse. Dette er svært gunstig under utviklingen av nettsiden, da nettsiden har en tilkobling mellom applikasjonen og tredjeparten. En feil med nettsiden kan ikke forårsake et krasj i produksjonsmiljøet. Det vil heller ikke spamme hovednettverket, og den vanlige bruken av tjenesten, med mange testbrukere og testenheter som ikke alltid blir konfigurert med rett verdi på de ulike parameterne. Et krasj i testmiljøet er ikke farlig, og det er derfor man benytter seg av testmiljø.

Man kan også få en visuell oversikt over brukere, ordrer og sensorer, som gjør at man lettere kan feilsøke og teste underveis. Her kan man sjekke hvilke brukere som har abonnentrolle, samt trykke inn på dem for å få oversikten over deres rettigheter, enheter og sluttbrukere. Her har man også mulighet til å slette og oppdatere informasjonen som er på de ulike brukerne.

I testmiljøet for administratorbrukere, og i testmiljøet for de tilhørende abonnentbrukerne, kan man sjekke om API-en blir brukt rett i nettbutikken og om dataen som blir opprettet er rett og knyttet til rett profil. Utenfor utviklingsfasen vil Lyse ha sine egne administratorbrukere der de vil ha den samme oversikten over dataen som blir sendt til Thingpark. Under vises et bilde av hvordan administratorsiden til testbrukeren ser ut.

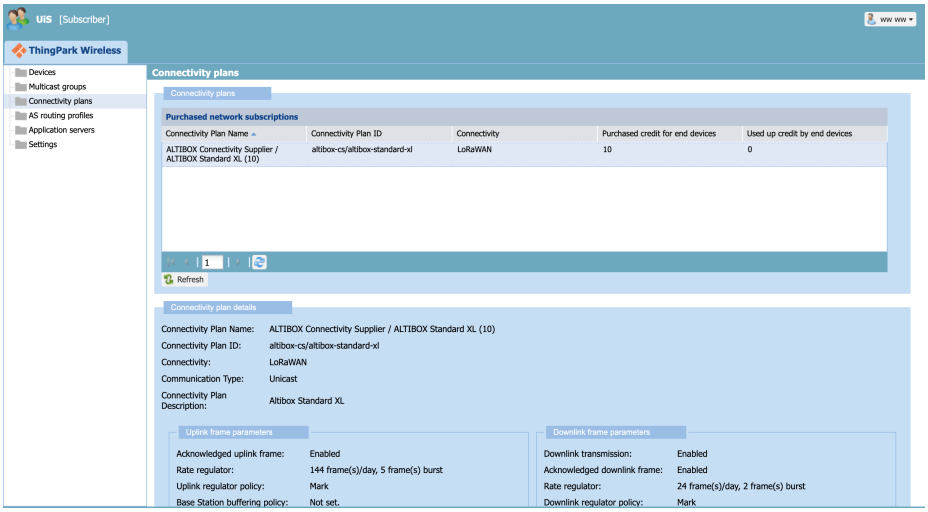


Figur 2.3: Testmiljøet for en leverandørbruker.

2.5.4 ThingPark Portal

ThingPark har også en egen portal hvor sluttbrukere kan logge seg inn på sin abonnentkonto. Portalen kan man finne på `https://altibox.thingpark.com/portal/`. Det er her sluttbrukerne kan sjekke oversikten over alle enhetene de har bestilt som inneholder et abonnement til Altibox sitt nettverk. Hver bruker vil få tildelt en basis lisenspakke som inneholder ulike pakker som gir de tilgang til verktøy. Disse verktøyene gir de rettigheter til å endre ting selv inne i portalen. Dette er en lisens som er utarbeidet av Lyse, og som inneholder tjenestene de synes det er viktig at en sluttbruker skal ha tilgang til. Denne brukeren er knyttet opp mot leverandørbrukeren i testmiljøet, slik at ingen av sluttbrukerne vil påvirke det faktiske nettverket i dette stadiet.

I portalen vil brukeren få tilgang til en trådløs logger og en enhetsbehandler via basispakken. Man har også mulighet til å sjekke tilbudene sine, sjekke sin egen bruker og endre antall sluttbrukere på en abonnentbruker. En abonnentbruker kan ha mange sluttbrukere, noe som gjør at en organisasjon kan opprette en abonnentbruker der alle sluttbrukerne er de ansatte. Figur 2.4 under viser hvordan en bruker sin enhetsbehandler-applikasjon ser ut. På selve bildet sjekker brukeren sine aktive konnektivitetsplaner. Man kan se at brukeren har fått tildelt en lisens som gir rettigheter til ti sluttenheter med konnektivitet via LoRaWAN nettverket.



Figur 2.4: Portalen til en sluttbruker. Her kan man administrere enhetene sine.

2.5.5 Aktivering av enheter

Det er mulig å koble til forskjellige sensorer med forskjellige egenskaper opp mot LoRaWAN-nettverket. Ronny Lorentzen (personlig kommunikasjon, 23. februar 2022) påpekte i en e-post at sensorene som blir tatt i bruk av Lyse er produsert av Decentlab og bruker produsentmodellen *LORA/AiB_1.0.2_ClassA_Rx2_SF12*.

DL-MBX er en av sensorene som blir brukt. Dette er en ultrasonisk distanse sensor som er laget for å kunne koble seg til LoRaWAN. Denne kan brukes til utendørs avstandsmåling av for eksempel dybdeovervåkning i vanntanker eller til flomovervåkning. [23] DL-IAM er en innendørs atmosfære monitor for LoRaWAN som kan måle luftkvaliteten i bygg. Den kan også blant annet måle bevegelse i et rom. Utenom disse to LoRaWAN-sensorene finnes det også flere eksempler på sensorer som er ment for LoRaWAN.

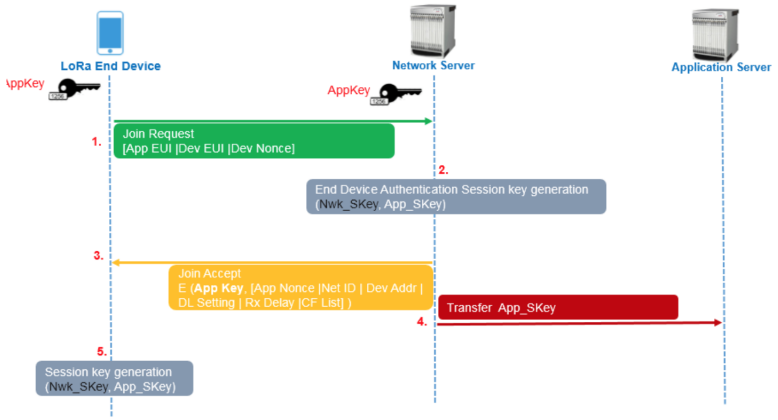
Selv om sensorene har forskjellige bruksområder, har de felles krav for hvilken informasjon som må oppgis for å kunne knytte en sensor til et aktivt abonnement. Utenom basislisensen har brukeren behov for en lisens som gir rettigheter til kunne koble opp et bestemt antall enheter til LoRaWAN. Dersom brukeren har en aktiv lisens, trengs informasjon som navn, *Device-*

ProfileId, *ActivationType EUI* og *ApplicationKey*.

Navnet skal hovedsaklig hjelpe brukere med å identifisere enheten sin på IoT-nettverket og på ThingPark portalen. *DeviceProfileId* er det samme som produsentmodellen som er nevnt tre avsnittet over og er lik for alle enhetene. EUI er en global unik identifikator til enheten som er tilpasset LoRaWAN. Det er altså en egen variabel for LoRaWAN. Denne står oppgitt i følgeseddelen til enheten og er sammensatt av 16 heksadesimale sifre. [24] Lyse har ikke mulighet til å gi ut ekte EUI-er til testing, så derfor må disse EUI-ene genereres som en tilfeldig testverdi for hver enhet. *ApplicationKey*-en er en AES-128 applikasjonsnøkkel som er tilordnet av eieren av appen til enheten for å kryptere kommunikasjonen mellom partene.[25] Denne trengs ved bruken av aktivering over OTAA. [26]

ActivationType er aktiveringstypen som man kan velge for enheten. Valget står mellom OTAA og ABP. OTAA står for "Over the Air Activation", mens ABP står for "Activation By Personalization". [20] I ABP-aktivering blir det brukt en fast utviklingsadresse sammen med flere øktnøkler for å koble til et forhåndsvalgt nettverk som er hardkodet inn i enheten. Utviklingsadressen blir brukt til å identifisere sensoren på nettverket. Denne adressen er likt ved ABP i hele levetiden til sensoren. [27]

Siden ABP har begrensninger knyttet til at det bruker en fast utviklingsadresse, en fastsatt sikkerhetsøkt og faste nettverksparametere, er OTAA ansett som en bedre tilkoblingsmetode. OTAA-enheter er utstyrt med rotnøkler. Sluttenheten i en aktivering med OTAA utfører en sammenføyningsprosedyre med et LoRaWAN-nettverk der en dynamisk utviklingsadresse gis til enheten. Disse rotnøklene blir brukt til å tilordne øktnøkler. Det vil si at utviklingsadressen endres hver gang en ny økt blir etablert. Derfor er det bestemt at man tar i bruk OTAA for alle enhetene. [27] Figur 2.5 viser hvordan OTAA fungerer.



Figur 2.5: Figuren som er hentet fra Techplayon [28] viser hvordan en *Over the Air Activation skjer*

Det er også flere valgfri tilleggsinformasjon som kan bli satt på hver enhet som omhandler alt fra fysisk geolokasjon til nettverksinnstillinger. Siden nettsiden skal fungere som en *Proof of Concept*, blir ikke de ekstra valgfrie opplysningene tatt hensyn til.

2.6 Innlogging

Muligheten til å la brukere opprette og logge inn på sin egen bruker er svært nødvendig for bruken av nettsiden og funksjonaliteten som er knyttet til ThingPark API-en. Nettsiden er avhengig av å ta vare på brukerdata, som man får tak i via innloggingen, for å kunne bruke informasjonen i metodekallene knyttet til ThingPark. Dette gjelder navn og e-post. Applikasjonen er også avhengig av å ha en personlig brukertabell i databasen for å lagre personlig data som id for abonnementet til en spesifikk person. Dette er også noe som trengs til API-en.

Brukerinformasjonen er også nødvendig for funksjonalitetene på selve nettsiden. Dette gjelder for både personlig innhold og for å begrense enkelte funksjonaliteter slik at kun en administrator har tilgang til dem. For administratortilganger er det viktig å kunne knytte en bruker opp mot en rolle

slik at kun de med administratortilgang kan få de ekstra funksjonalitetene. Brukeriden blir også brukt til å få personlig innhold som kontroll over egne bestillinger, favoritter og brukerinformasjon.

Det er knyttet mange sikkerhetsspørsmål rundt temaet innlogging. Det å lage et sikkert innloggingsystem er noe som krever mye erfaring og lang tid. Det er ikke anbefalt å lage sin egen innlogging fra bunnen av. Det ble diskutert med Lyse hvordan man skulle håndtere innloggingen på en best mulig måte. Den mest optimale løsningen var å integrere Lyse sin interne innlogging inn i nettsiden, men dette var noe gruppen ikke hadde mulighet til å få tilgang til per nå. Det ble derfor bestemt at det skulle lages en sikker innlogging ved hjelp av godkjente krypteringsbibliotek og JWT-tokener.

Selv om dette ikke kan kvalifiseres som like sikkert som Lyse sine systemer, så vil den fungere som en midlertidig løsning frem til Lyse kan implementere inn sin egen løsning. På den måten vil koden være tilpasset et innloggingsystem, samtidig som man kan teste funksjonalitetene. Likevel ønsket gruppemedlemmene å ta i bruk en sosial innlogging for å kunne støtte en moderne og enkel løsning for brukerne. Dette var noe Lyse sa seg enig i, og noe de også kunne bruke i etterkant. Det ble derfor bestemt å ha to ulike muligheter for innlogging. Metoden for innloggingen er lik for begge løsningene. Den eneste forskjellen er hvor man får innloggingsinformasjonen fra. Dette konseptet viser hvor lett det er å integrere en ny innloggingsmetoden inn i nettbutikken. Det blir da enkelt å integrere inn Lyse sin innlogging i fremtiden.

2.6.1 Flask-JWT-Extended

Nettsidene er på to forskjellige tjenere, nemlig frontend og backend. Dermed er det viktig å ha en måte i forespørslene som kan identifisere brukerne. Dette kan gjøres ved hjelp av JWT. JWT (også kjent som *Javasciprt Object Notation Web Token*) kan brukes til å autentisere brukere og legge til tilleggsdata. Flask-JWT-extended er sikkert og forlanger en CSRF-token for å fungere. Det er mulig å konfigurer pakken til å ikke forlange CSRF-token, men det er en stor sikkerhetsrisiko og bør ikke gjøres. Å ha en CSRF-token vil forebygge *Cross Site Request Forgery* angrep. Med Flask-JWT-extended er det mulig å oppdatere levetiden til JWT-en. Flask-JWT-extended støtter også å oppheve tokens og det å svarteliste tokens. [29]

Mange benytter seg av Flask-Session for til innlogging med Flask applikasjoner. Det ville ikke fungert i dette prosjektet ettersom nettsiden benytter to forskjellige tjenere. Flask-Session laget kun for tjenerside sessions og ikke klintside. Derfor blir Flask-JWT-Extended brukt som inneholder en token og støtter session mellom klientsiden og tjenersiden.

2.6.2 Sosial innlogging

Bruk av sosial innlogging gir mange fordeler. Disse fordelene er blant grunnene til at det blir implementert. Det gir blant annet en bedre brukeropplevelse på den måten at den reduserer antall klikk og tid en bruker bruker på å logge inn. Det eliminerer også behovet for å registrere en ny bruker. Brukeren blir også mindre avhengig av passord, da man kan bruke det samme sterke passordet fra en tjeneste flere steder. Brukeren slipper dermed å huske enda et nytt passord, og sannsynligheten for at det blir brukt et unikt og sterkt passord øker. Da øker sikkerheten også.

Det er også mer sikkert som innlogging, da det blir tatt i bruk et sikkert innloggingsystem fra en sikker tjeneste. En annen fordel er at det skaper en felles gjenkjennbar tjeneste man kan bruke til å logge seg inn med. Dermed kan det være enklere for brukerne å dele sine data med nye og ukjente applikasjoner, fordi de allerede stoler på den sosiale plattformen som brukes. [30]

Lyse foreslo bruken av Google sin innlogging, og gruppemedlemmene var enig. Det finnes også andre populære tjenester som fungerer på samme måte. Eksempelvis Facebook, LinkedIn og Twitter. De to mest populære påloggingsplattformene er Facebook og Google. Disse tjenestene har flest brukere, noe som gjør at sannsynligheten for at en bruker kan ta i bruk denne løsningen er stor. [30]

Hver påloggingsplattform har forskjellige personvernsfunksjoner og ulike egenskaper når det kommer til implementasjonen av påloggingen. De største forskjellene omhandler hvilken informasjon man ønsker å hente ut fra plattformen. Noen av alternativene er mindre strenge enn andre, mens noen har mer fleksibilitet til å velge hvilke data man vil hente ut.[30] Siden applikasjonen kun behøver generell brukerdata har ikke valget mellom Google og Facebook så mye å si. Det er også mulig å implementere flere tjenester i

fremtiden.

Kapittel 3

Konstruksjon

Kommende kapittel omhandler oppbyggingen av nettsiden og dens funksjoner, samt hvordan de ulike komponentene og delene samhandler med hverandre. Selve oppgaven er delt opp i to underprosjekter i koden. En for tjenersiden og en for klientsiden. Grunnen til dette var at det gjorde jobben med å distribuere nettsiden via Azure enklere. Linken for tjenersiden på Azure er `https://iot-webshop-backend.azurewebsites.net/`.

Innstillingene på Azure gjør at dersom det har vært en stund siden tjenersiden har vært aktiv, så vil det ta cirka 30 sekunder før den starter opp. Valget for disse innstillingene er basert på kostnadene som er knyttet til tjenerbruken. Det er billigere å distribuere nettsiden med disse innstillingene. URL-en til klientsiden er `https://kind-stone-04fae9e03.1.azurestaticapps.net/`. En administrator har *iot.webshop.lyse@gmail.com* som e-post og et passord som er *UiSerbra123!*.

3.1 Azure-tjenester

For at nettsidene skal fungere, må den kunne koble seg opp mot Azure-tjenestene. Som nevnt i delkapittel 2.1.2 blir frontenden levert ved hjelp av Azure Static Web App og oppfører seg som en *single page application*. Det er et problem med SPA-nettsider i Azure Static Web App som gjør at det ikke

vil fungere å skrive inn en manuell URL til nettsiden slik som */products* uten ekstra konfigurering. Grunnen til dette er fordi SPA er avhengig av ruting fra klientsiden.

Denne klient-side-rutingen vil oppdatere lokasjonen uten å lage forespørsler til tjeneren. Ved Azure Static Web App må man definere en *fallback route* som gir HTML-filen *index.html*. Dette må være gjort for at det skal være mulig å skrive en manuell URL og benytte seg av klient-side-ruting. En *fallback route* er mulig å legge til under *navigationFallback* i en konfigurasjon fil kalt *staticwebapp.config.json*. Denne må ligge i root-mappen til frontend-prosjektet. Det er viktig at filen har rett navn og plassering for at Azure Static Web App skal finne den.[31]

Koden ligger i et Github repository, men den ligger også inne på et Azure repository. Azure Repos blir brukt til deployment, mens Github repository-et blir brukt til utvikling. Det er koden som ligger på Azure repository som blir brukt i produksjonen. Under DevOps er det pipelines som har ansvar for å dytte koden til frontend- og backend-nettsiden. Pipelines bruker yaml-filer for å vite hvordan de skal operere. Disse yaml-filene ligger under vedleggene B.1 B.2.

Azure Repo-et fungerer som et remote repository og kan derfor enkelt bli oppdatert. Den blir oppdatert ved å sende endringene til det remote repository-et ved *git push <remote_repository>* I dette prosjektet er det remote repository-et kalt "Azure". Yaml-filenene følger med på når hovedgrenen blir oppdatert. Når den blir oppdatert vil pipeline-ene kjøre og dermed oppdatere nettsidene automatisk.

At Azure Key Vault er riktig satt opp er essensielt for at nettsidene skal fungere. Uten tilkobling til Key Vault vil ikke nettsidene ha tilgang til viktige secrets som brukes i koden. Deployment av nettsidene vil også krasje fordi secrets blir benyttet i yaml-filene.

For få tilgang til Azure Key Vault på en privat maskin, er det nødvendig å laste ned Azure sin kommandolinje for å autentisere seg før man kan benytte secrets. Ved Azure sin kommandolinje må man skrive inn *az login*. Da blir man sendt til nettleseren sin for å autentisere seg via en Azure-portal. Da er det viktig at brukeren har tillatelser til å hente ut secrets. Tillatelsene kan legges til i Azure Key Vault i Azure-portalen under access

policies. Her er det også viktig å legge til at nettsideapplikasjonen har nødvendig tillatelser for å bruke Azure secrets. For å bruke secrets i koden må man importere *client* fra filen kalt *secret_ setup.py* og setter en variabel lik *client.get_ secret(secretName).value*.

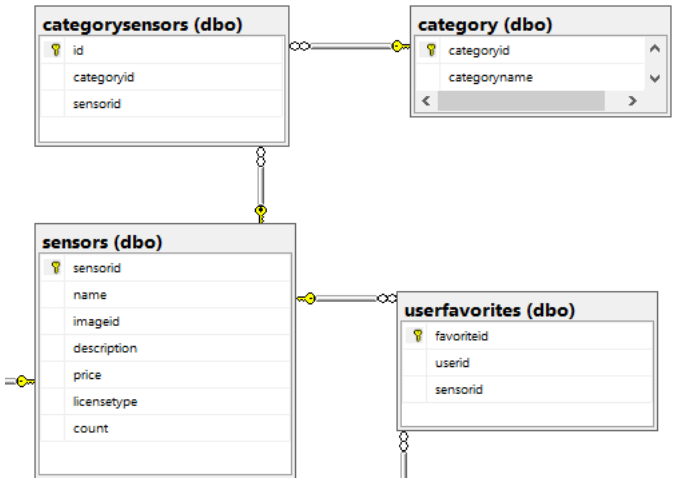
3.2 Database

Som nevnt i delkapittel 2.1.4, blir Azure SQL Database brukt til denne oppgaven. Dette underkapittelet vil diskutere hvordan databasen er bygd opp. Det vil også bli beskrevet hvordan nettsiden kobler seg opp mot databasen ved hjelp av Pyodbc og Azure secrets.

3.2.1 Database konstruksjons

Vedlegg A.1 viser et diagram av hvordan databasen er bygd opp. Lyse satt noen krav til hvilke data som måtte bli lagret. Gruppen hadde mulighet til å endre disse etter behov. Lyse ville at det skulle bli lagret pris, navn, id, antall, bilde og beskrivelse av hver sensor. Derfor blir informasjonen lagret i tabellen *sensors*, hvor bilde blir lagret i tabellen *images*.

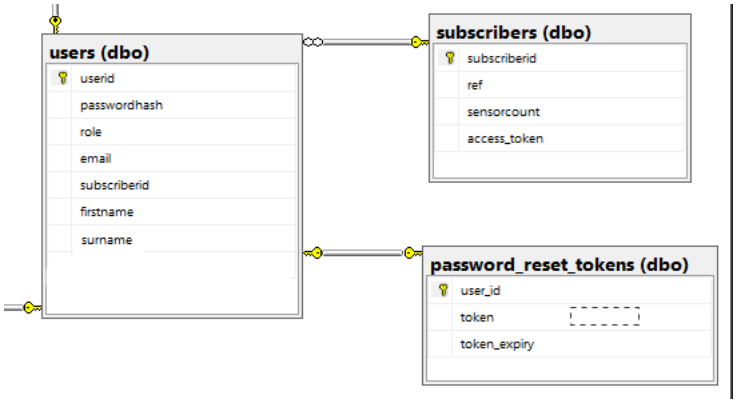
Tabellen *sensors* har en fremmednøkkel som peker til primær nøkkelen til *images*. Denne brukes for å hente ut bildet. Grunnen til at de er i forskjellige tabeller, er fordi bilder tar mye plass. Det er heller ikke garantert at man trenger bildet når man henter ut informasjon om sensoren. Da er det gunstig å gjøre tabellen *sensors* minst mulig og dermed raskere. Dersom bilde blir endret på trenger ikke tabellen *sensors* å reagere på det, fordi primærnøkkelen til *images*-tabellen fortsatt er lik. *Count* minsker når kunder kjøper produktet. Administratorer kan også endre denne verdien.



Figur 3.1: Utklipp av databasediagrammet som viser sensortabellen

Lyse ville også at det skulle være forskjellige kategorier. Derfor var det nødvendig med en tabell for dette som inneholder et unik kategorinummer og kategorinavn. Problemet som oppstår er at det blir en mange-til-mange relasjon mellom kategoritabellen og sensortabellen. Dette ble løst med å lage tabellen *categorysensors*, se fig 3.1. Den inneholder en unik Id, samt fremmednøkler som refererer til kategoritabellen og sensortabellen. *Categorysensors* har en en-til-mange relasjon med både kategoritabellen og sensortabellen.

I tillegg til kategorier, så har hver bruker ulike favoritt produkter. Derfor er det en tabell for favoritter som har fremmednøkler som peker på sensortabellen og brukertabellen. Figur 3.1 viser dette. I figur 3.1 vises ikke koblingen mellom brukertabellen og favorittabellen, men den kan ses i vedlegget A.1 som viser hele databasediagrammet.



Figur 3.2: Utklipp av databasen som viser users, password_reset_token, og subscribers tabellene

Brukertabellen som vises på fig 3.2 inneholder for det meste standard brukerdata. Passordet blir brukt av brukeren til å logge seg inn på nettsiden, og blir kun lagret dersom brukerne ikke benytter seg av sosial innlogging. Passordet brukes til å autentisere brukere ved innlogging. Ved bruk av en sosial innlogging er det en ekstern API som autentiserer brukerne. Dermed trenger man ikke å lagre passord til brukere som bruker Google-innlogging.

Passordet blir *hashed* før det settes inn i databasen. Hashalgoritmen som blir brukt til passordene er *SHA256*. *SHA256* er en av de mest standard hash-algoritmene. Denne returnerer en hash som er 256 bits, også kjent som 32 bytes. Kolonnen *role* er et heltall som viser rollen til brukeren. Her betyr verdi 0 at det er en normal bruker, mens verdi 1 betyr at det er en administrator.

Brukertabellen inneholder også en fremmednøkkel som refererer til *subscriber*-tabellen. *Subscriber*-tabellen inneholder data som er nødvendig for å benytte seg av ThingPark DX API-en. Der *subscriberid* og *ref* kan brukes til å identifisere brukere på ThingPark. *Sensorcount* er antall sensorer som brukeren har på ThingPark. Grunnen til at denne verdien blir lagret og oppdatert, er for å kunne oppdatere lisensene automatisk ut i fra hvor mange sensorer som er registrert med abonnement.

Tilgangstokenen er en *OAUTH*-token som blir brukt for å gjennomføre handlinger som krever visse tillatelser. *OAuth* er en tilgangstoken som blir

brukt av API-er som gjennomføre forespørsler for brukere [32]. En token har blitt konfigurert til å vare i 90 dager og blir kryptert symmetrisk ved hjelp av Fernet. Fernet benytter seg av krypteringsalgoritmen AES128. For å legge til sensorer til brukere på ThingPark trenger man enten innloggingsinformasjonen eller en gyldig OAUTH-token. Tokenen blir lagret slik at brukeren slipper å skrive inn passordet sitt neste gang de kjøper sensorer. [33]

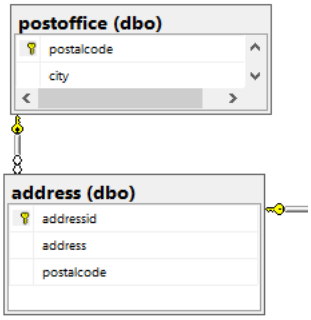
Databasen har også en tabell kalt *reset_password_tokens*. Denne vises på fig 3.2. Denne er for å støtte funksjonaliteten til å kunne endre passordet dersom brukere glemmer dette. Her er primærnøkkelen *user_id* også en fremmed nøkkel som refererer til brukertabellen. Dette er ikke normalt, men er lov dersom det er en-til-en relasjon mellom tabellene. Det betyr at hver rad i brukertabellen her en eller ingen tilsvarende rader i *reset_password_tokens*, og at hver rad i tabellen *reset_password_tokens* har en eller ingen tilsvarende rader i brukertabellen.

For å sette inn i *reset_password_token*-tabellen blir transaksjoner benyttet for å følge ACID prinsippene: *atomicity*, *consistency*, *isolation* og *durability*. *Atomicity* sørger for at enten hele transaksjonen skjer, eller så vil ingenting skje. *Consistency* blir benyttet for å sikre at informasjonen kun blir endret med lovlige verdier. [34]

Isolation sikrer at raden kun blir benyttet et sted om gangen, og dette er hovedgrunnen til at transaksjoner blir brukt. Det vil derfor forekomme en feil om to forskjellige brukere prøver å lage en token på samme bruker samtidig. Da vil databasen legge til 2 forskjellige rader med samme primærnøkkel. Siden kun en av dem skjer om gangen ved hjelp av transaksjonen, vil den første legge til en rad, mens den andre vil endre den eksisterende raden. Dette skjer fordi koden i transaksjonen først sjekker om det allerede finnes en token på denne brukeren. Om en token finnes vil den endre raden i databasen med ny token og ny utløpsdato. Om det ikke finnes en token vil den legge til en ny rad med all data som trenges. [34]

Durability sikrer at dersom transaksjon blir gjennomført vil den vare og ikke forsvinne dersom systemet får noen problemer. Tokenen som blir lagret blir hash-et med SHA256 før den blir satt inn i databasen. Den blir brukt til å autentisere en bruker når en bruker vil endre passordet sitt. En token varer i en time etter forespørsel om å bytte passord ble sendt. [34]

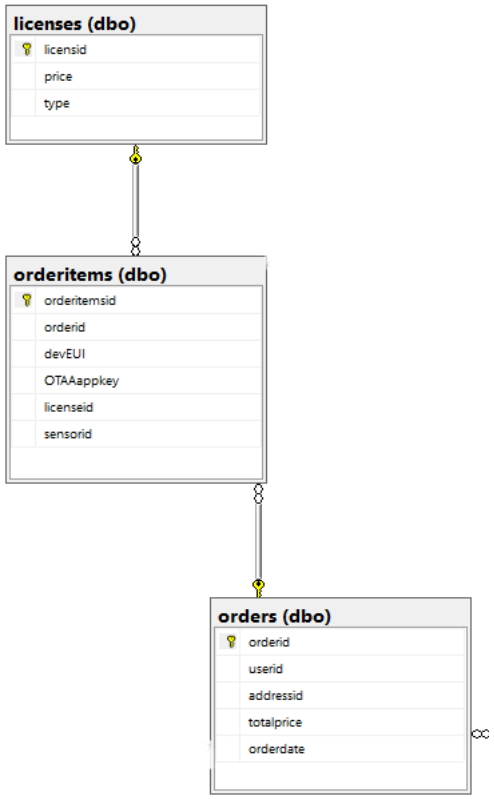
Adressetabellen inneholder adressen og postnummer som vises på fig 3.3. Postnummer er en fremmednøkkel som peker på tabellen *postoffice*. *Postoffice*-tabellen inneholder postnummer og poststed. Grunnen til at poststed blir lagret i en separat tabell er for å få en bedre designet database. Fordelen med å dele det opp slik, er at dersom et postnummer endres, så må man ikke endre alle radene i adressetabellene som inneholder dette postnummeret. Man trenger kun å endre en rad i tabellen *postoffice*. Man burde ikke lagre to felt i samme tabell som er avhengige av hverandre.



Figur 3.3: Utklipp av databasen som viser tabellene for adresser og postnumre

Ordretabellen inneholder data der hver rad er en ordre. Den inneholder totalpris, ordredato og fremmednøkler som refererer til adressen og brukeren. Ordretabellen har også en en-til-mange relasjon til *orderitems*-tabellen. *orderitems* er hver sensor som er i en ordre, og som har en fremmednøkler som peker på ordretabellen og sensortabellen. Kolonnene *devEUI* og *OTAAappkey* er to verdier som kreves for å registrere sensoren på ThingPark, og er unik til hver fysiske sensor.

DevEUI er en unik streng som består av 16 heksadesimale tegn, og som brukes til å identifisere hver enkel sensor. *OTAAappkey* er en nøkkel som består av 32 heksadesimale tegn, noe som tilsvarer 128 bits. Den brukes til å kryptere og dekryptere informasjonen mellom sensorene og ThingPark. Informasjonen blir kryptert symmetrisk ved hjelp av algoritmen AES128. Dersom brukeren ikke vil ha abonnement inkludert i kjøpet, vil disse feltene være tomme. Per dags dato ville Lyse at sensorene bare skal benytte seg av en standard lisens, og derfor er *licenseid* alltid tom. Tabellen *licenses* blir ikke benyttet nå, men den er laget dersom Lyse vil implementere ulike typer lisenser senere.



Figur 3.4: Utklipp av databasen som viser tabellene for ordre, ordreprodukter og lisenser

Den siste tabellen, *etc*, blir brukt til å lagre forskjellige priser som ikke er knyttet til en enkel sensor. Den har 3 kolonner: *id*, *name*, *price*. Det blir for eksempel lagret pris på frakt og prisen på standardabonnement som blir brukt. Dette gjør det lett for administratorer å endre disse prisene dersom det er ønskelig.

3.2.2 Database tilkobling

Databasen er Azure SQL Database, som er skybasert, og dermed må nettsiden kunne koble seg til databasen. Her har det blitt brukt en tilleggspakke kalt *Pyodbc* som har blitt beskrevet i delkapittel 2.1.5. For å benytte seg

av Pyodbc må man først laste ned ODBC-driver til datamaskinen. ODBC, også kjent som *Open Database Connection*, er en driver som blir benyttet til oppkobling til databaser. Videre trengs det informasjon om databasen som man kan finne på Azure-portalen under databasen. Pyodbc trenger informasjon om hvilken driver som blir brukt, hva database-tjeneren heter, hva selve databasen heter og brukernavn og passord til en administratorbruker. Dette kan man se i kodesnipp 3.1. På linje 6 og 7 i kodesnippen 3.1 vises ikke brukernavn og passord i klartekst, men det blir benyttet Azure secrets. Dette er av sikkerhetsmessige grunner for å ikke eksponere passord og brukernavn i koden. For at det skal være mulig å koble seg opp mot databasen må man hvitliste IP-adressen til datamaskinen sin. Dette kan gjøres via Azure-portalen. På Azure portalen kan man enten gå til databasen eller SQL-tjeneren og legger IP-adressen i brannmur innstillingene.

Kode 3.1: Snippet som viser oppkobling til databasen

```
1  import pyodbc
2  from backend.modules.secret_setup import client
3
4  server = 'iot-webshop-server.database.windows.net'
5  database = 'IoTDatabase'
6  username = client.get_secret('db-username').value
7  password = client.get_secret('db-password').value
8  driver= '{ODBC Driver 17 for SQL Server}'
9  string = 'DRIVER='+driver+'; SERVER=tcp:'+server+';PORT=1433;
10 DATABASE='+database+';UID='+username+';PWD='+ password
11
12 def get_db():
13     return pyodbc.connect(string)
```

Det er også mulig å endre på databasen uten å benytte seg av Python. Azure-portalen har en *Query Editor* som kan brukes til å skrive SQL-setninger til databasen og enkelt se alle tabellene som finnes i databasen. Microsoft SQL Server Management System, også kalt SSMS, et et program som har blitt benyttet for å skrive SQL-setninger og enkelt lage et diagram over databasen. Vedlegg A.1 ble laget ved hjelp av SSMS.

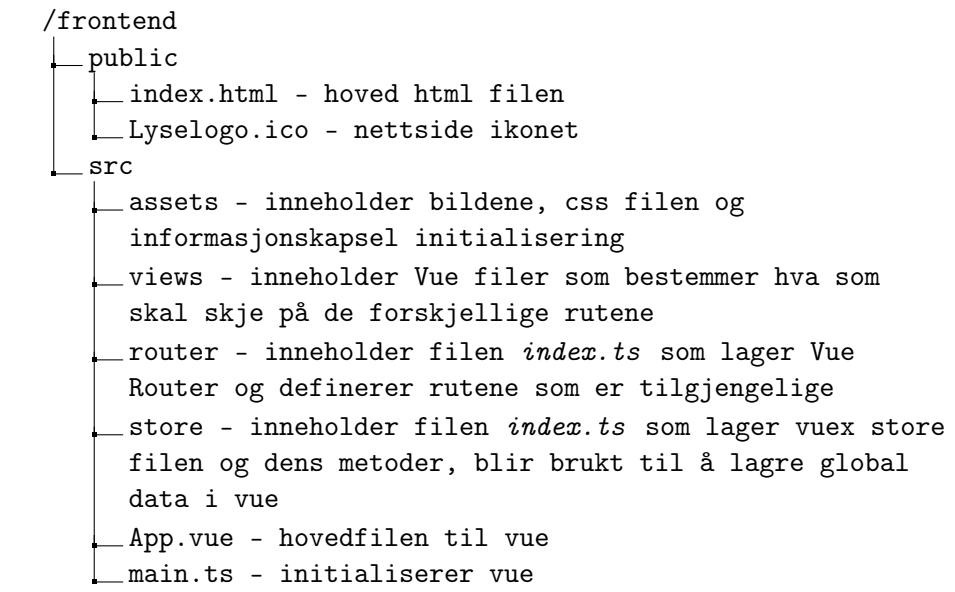
3.3 Klientside

3.3.1 Oppstart

Oppstart av klienttjeneren i produksjonen blir automatisk gjort ved hjelp av pipelines og Azure DevOps, slik som det er beskrevet i underkapittel 3.1. Oppstart av klienttjeneren under utvikling krever at node er installert på datamaskinen. Før nettsiden kan startes, må *npm i* kjøres med pathen "/frontend". Dette gjør at nødvendige tilleggspakker blir lastet ned eller oppdatert. Disse pakkene er definert i filen *package.json*. Dersom alle pakkene er lastet ned, kan klienttjeneren starte ved kommandoen *npm run serve* under "/frontend".

3.3.2 Kodestruktur

All kode som blir benyttet av klienttjeneren ligger i undermappen frontend. I frontend mappen ligger det diverse mapper og noen filer til konfigurasjon. Figur 3.5 viser de mest essensielle mappene til tjenersiden og en forklaring på hva som skjer hvor.



Figur 3.5: Figur som viser de viktigste mappene og filene til klientsiden

3.3.3 Ruter og tilgang

Vedlegg A.2 viser et nettstedskart over nettsiden. Den er fargekodet etter hvilke tillatelser en brukere må ha for å bruke dem. Hvit farge betyr at alle har tilgang uten noe form for autentisering. Grønn farge betyr at det er kun brukere som ikke er innlogget som har tilgang. Blå farge betyr at brukeren må være logget inn for å se disse sidene. Rød betyr at brukeren må ha være innlogget og være en administrator for å ha tilgang til de sidene. En administrator vil ha *role 1* i databasen. Konfigurering av tilgangen til klientsider har blitt gjort ved hjelp av Vue Ruter.

Dersom en bruker prøver å benytte seg av sider som krever administratortilgang uten tilstrekkelig tillatelser vil brukeren bli omdirigert til */404* siden. Ukjente URL-er vil også bli omdirigert her. Grunnen til at brukere uten tilstrekkelig tillatelser blir omdirigert fra administratorsidene, er for å ikke gi informasjon til eventuelle angripere om hvor administratorsidene ligger. Dersom en besøkende, som ikke er innlogget, prøver å benytte seg av sider som krever innlogging, vil de bli omdirigert til */login*. Dette er for å øke brukervennligheten, da det vil være enkelt for brukeren å logge seg inn eller

registrere seg. Dersom en bruker som er innlogget prøver å benytte seg av sider som krever at man ikke er innlogget vil siden bare være tom. Dette gjelder blant annet innloggingssiden og registreringssiden.

3.4 Tjenerside

3.4.1 Oppstart

Oppstart av tjeneren i produksjonen blir automatisk gjort av pipelines i Azure DevOps slik som nevnt i underkapittel 3.1. For å starte tjeneren må datamaskinen ha Python og pip installert. En må begynne med å aktivere et virtuell miljø som gjøres med forskjellig syntaks etter hvilket operativsystem datamaskinen kjører på. Etter det virtuelle miljøet er aktivert må man kjøre kommandoene som er vist i kodesnipp 3.2. Første kommando er for at pip skal laste ned pakkene som blir brukt i oppgaven og er definert i *requirements.txt*. Linje 2 og 3 lager miljøvariabler for hvor Flask kan finne prosjektet og hvilket miljø det skal bli kjørt i. I Kodesnip 3.2 blir det kjørt i utviklingsmiljø. Kommandoen på linje 4 er for å starte tjeneren. Før tjeneren blir startet er det viktig å hvitliste IP-adressen til databasen slik nevnt i underkapittel 3.2.2.

Kode 3.2: Kommandoer for å starte tjeneren første gang

```
1   $ pip install -r requirements.txt
2   $ export FLASK_APP=startup
3   $ export FLASK_ENV=development
4   $ flask run
```

3.4.2 Kodestruktur

Tjenersiden er delt inn i et filsystem for å holde tjenersiden lettere å vedlikeholde og utvikle. Figur 3.6 viser et forenklet filsystem. Her vises kun de mest essensielle filene. Startup.py er filen som flask kjører når den starter (se kode 3.2). Denne importerer alt som skjer i program.py. I program.py skjer all generell konfigurasjon, mens i config.py skjer det en konfigurasjon

som er spesifikt til hvilket miljø som blir brukt. For eksempel vil klientsiden sin URL være avhengig av miljøet. I filen views.py er det kode for hva som skal skje på de forskjellige rutene som tjeneren har. Alle filene i modules-mappen er laget for å dele opp koden bedre. Den inneholder klasser, metoder og funksjoner som blir brukt i views.py.

```
/
├── startup.py
├── program.py
├── config
│   └── config.py
└── backend
    ├── views.py
    └── modules
        ├── auth_token_client.py - kode for å hånderting av
        │   ThingPark authtoken
        ├── blob_reader.py - kode base64 konvertering
        ├── confirmationEmail.py- kode for generering og sending
        │   av mail
        ├── db_methods.py - kode for å endring, henting av data
        │   i databasen
        ├── db_setup.py - kode for å oppkobling mot databasen
        ├── dxCore_client.py - klasse, metoder og funksjoner
        │   rettet mot ThingPark
        ├── google_auth.py - funksjoner som har med Google
        │   innlogging
        ├── jwt.py - funksjoner med JSON Web Tokens
        ├── proccess_order_methods.py - funksjoner for Thingpark
        │   rettet mot bestillinger
        ├── secret_setup.py - oppkobling mot Azure sitt
        │   nøkkelhvelv
        └── validation.py - funksjoner for å autentisere
            brukerdata
```

Figur 3.6: Figur som viser de viktigste mappene og filene til tjenersiden

3.4.3 Ruter og tilgang

Listen under viser rutene og forventet tilgang til rutene. Det står også en kort beskrivelse på hva som skjer, i tillegg til de forventede parametere. På parameterene står det også hvilken datatypen parameteren har.

- Ruter for alle
  - / (GET) returnerer en tom streng
  - /getProducts (GET) henter ut all produktene
  - /getProducts (GET) henter ut kategoriene
  - /getProductsByIds/<string:productIds> (GET) henter ut noen produkter
  - /getProduct/<int:productId> (GET) henter ut en bestemt sensor
  - /getUser (GET) henter brukerobjekt til aktiv bruker
  - /search (GET) henter ut sensorid, sensornavn og pris
  - /login (POST) logger inn en bruker, parametere:
    - username: streng som er e-posten.
    - password: streng
  - /google_login (GET) henter ut URL til Google innloggingsapi
  - /google_login/callback (POST) logger bruker inn ved hjelp av Google kode: parametere:
    - data: streng, inneholder URL-en til klienten som igjen inneholder koden som blir sendt til Google
  - /register (POST) registrerer en bruker, parametere:
    - firstName: streng
    - surName: streng
    - username: streng som er e-posten.
    - password: streng
  - /forgotPassword (POST) sender mail til brukeren for å nullstille passord, parametere:
    - email: streng

  - /resetPassword (POST) endrer passordet til en bruker dersom det er en gyldig token, parametere:
    - email: streng
    - password: streng
    - token: streng

- Ruter for innlogget brukere
  - /getOrder/<int:orderId> (GET) henter ut en bestemt bestilling til innlogget bruker
  - /processOrder (POST) registrerer ordre og eventuelt registrerer sensorer på ThingPark, parametere:
    - items: List<object>, liste av sensorene
    - subscription: boolsk
    - pwd: streng som er ThingPark passord
  - /addAddress (POST) registrerer adressen før bestillingen, parametere:
    - postalcode: heltall
    - city: streng
    - address: streng
  - /getEtc (GET) henter ut abonnementspris og fraktpris
  - /getFavorites (GET) henter ut favorittproduktene til en bruker
  - /deleteFavorite (POST) fjerner et produkt fra favorittene, parametere:
    - sensorId: heltall
  - /addFavorite (POST) legger et produkt til favoritter, parametere:
    - sensorId: heltall
  - /getOrders (GET) henter ut bestillingene til innlogget bruker
  - /getAccessToken (GET) henter ut tilgangstoken til brukeren på ThingPark
  - /newAccessToken (POST) lager ny tilgangstoken til brukeren på ThingPark, parametere:
    - password:streng
  - /getUserInformation (GET) henter ut brukerinformasjonen til innlogget bruker

  - /logout (GET) logger ut innlogget bruker

- Ruter for administratorer
  - /addCategory (POST) legger til kategori, parameter:
    - categoryname: name
  - /deleteCategory (POST) sletter en kategori, parameter:
    - categoryId: heltall
  - /updateCategorySensor (POST) oppdaterer hvilke sensorer som er registrert på en kategori, parametere:
    - categoryId: heltall
  - /addProduct (POST) legger til en sensor, parametere:
    - imageblob: streng
    - name: streng
    - description: streng
    - price: flyttall
    - count: heltall
  - /deleteProduct (POST) sletter en sensor, parametere:
    - sensorId: heltall
  - /updateProduct (POST) oppdaterer en sensor, parametere:
    - imageblob: streng
    - name: streng
    - description: streng
    - price: flyttall
    - count: heltall
    - license: heltall eller null
  - /updateEtc (POST) oppdaterer en av prisene, parametere:
    - name: streng, navn på prisen som blir oppdatert
    - price: flyttall
  - /getAllOrders (GET) henter ut alle bestillingene

3.5 Innlogging

Da oppgaven ble diskutert med Lyse var det et ønske om et innlogging system på nettsiden. Dette skulle vise visuelt hvordan det ville fungert. Selve innloggingssystemet ville bli laget selv og dermed ikke like sikkert som det Lyse benytter seg av. Ettersom oppgaven hovedsakelig er en POC syntes Lyse at dette var sikkert nok. Ettersom nettsidene er delt i frontend og backend er det naturlig at en innlogget bruker sender en tilgangstoken i headeren til forespørslene.

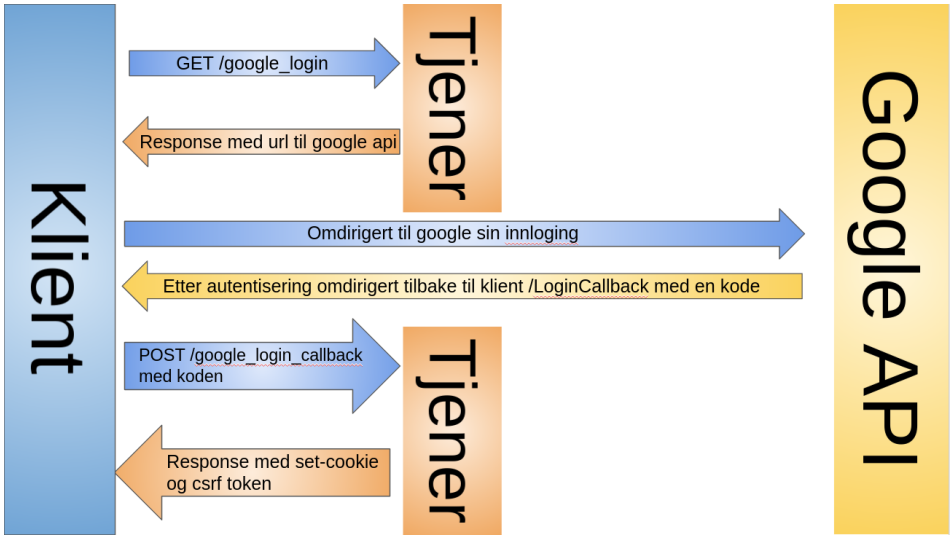
Tilgangstokenen er en JSON Web Token (JWT). For å lage JWT blir tilleggspakken Flask-JWT-extended brukt. Identiteten til JWT er brukeriden, men det har også blitt lagt til et ekstra felt som er rollen til brukeren. Dette blir lagt til ved hjelp av JWT sin *additional_ claims*. Dette er for å ha enkel tilgang til rollen til brukeren. JWT har blitt konfigurert med en levetid på 7 år.

Når en bruker logger inn sender klienten brukernavn og passord i en forespørsel til tjeneren. Tjeneren vil først validere informasjonen som ble sendt. Dersom brukernavn og passord er rett, vil tjeneren lage en JWT-token. Deretter vil tjeneren returnere en respons som inneholder en *set-cookie* og en CSRF-token. En CSRF-token er en token som Flask-JWT-extended forlanger i POST-forespørsler. Dette er av sikkerhetsmessige grunner for å forhindre *Cross Site Request Forgery*-angrep. I konfigurasjonen til tjeneren er det definert at JWT skal bli lagret i informasjonskapselen også kalt ”cookie” på engelsk. Denne skal vare i 7 dager. Det er også en metode som oppdaterer levetiden til 7 dager, dersom det er en forespørsel som har en JWT som har kortere levetid enn 3 dager.

Klient får en respons av tjeneren etter innloggingen. Ved hjelp av informasjonen i denne responsen vil alle fremtidige forespørsler til tjeneren inneholde CSRF-token og tilgangstoken som er JWT. Når fremtidige forespørsler kommer til tjeneren, vil tjeneren skjønne hvem som sendte forespørselen ved hjelp av JWT, i tillegg til hvilken rolle de har. Slik kan man gjennomføre handlinger som krever visse tillatelser.

3.5.1 Google Innlogging

Slik nevnt i delkapittel 2.6.2, kan en bruker logge seg inn med Google-innlogging. For at Google-innlogging skal fungere er det nødvendig med en utviklerbruker hos Google. Der må man lage et prosjekt og registrerer URL-en til nettsiden og hvilken URL de skal omdirigere til etter brukeren har blitt autentisert. Se fig 3.7. Slik som man kan se på fig 3.7 er det viktig å definere URL-en til frontend-nettsiden og ikke backend-nettsiden. Dette prosjektet er registrert som et utviklingsprosjekt hos Google. Det betyr at man må manuelt legge inn de Google-kontoene som kan benytte seg av Google innloggingen.



Figur 3.7: Viser forespørslene som skjer ved google innlogging

Når en bruker trykker på Google-knappen på registrerings- eller innloggingssiden så blir det sendt en GET-forespørsel til tjeneren. Da vil tjeneren finne URL-en til Google-innlogging som den skal sende som en respons tilbake til klienten. Deretter vil brukeren bli omdirigert til en Google innloggingsside. Etter brukeren har logget seg inn vil brukeren bli omdirigert tilbake til klient med */login/callback* og en kode slik som i figur 3.7. Når klienten får denne koden vil den videresende den til tjeneren ved hjelp av en POST-forespørsel til *google_login_callback*. Her vil tjeneres først dekode koden den fikk fra klienten. Deretter vil den sende koden til Google sin API for

å hente ut brukerdata. Her henter den ut navnet til brukeren og e-posten. Dersom denne e-posten har en bruker vil den bli logget inn som normalt. Dersom brukeren ikke finnes vil tjeneren lage en bruker og sette den inn i databasen. Deretter vil brukeren bli logget inn som normalt.

For at kommunikasjon med Google sin API skal fungere, må nettsiden benytte seg av HTTPS og ikke kun HTTP. Produksjonsnettsidene er med HTTPS, men under utvikling er den i HTTP. Dette fikses ved å sette en miljøvariabel kalt *OAUTHLIB_INSECURE_TRANSPORT* lik "1", og dermed gjøre det mulig å kommunisere med Google sin API ved hjelp av HTTP. Denne variablene bør kun være 1 under utvikling.

3.6 Autorisasjon til API-ene

Siden begge ThingPark API-ene krever en autorisasjonstoken for å kunne utføre handlinger på brukeren sin vegne, er det opprettet en klasse som håndterer alle forespørslene knyttet til autorisasjonstokenene. Denne klassen heter *AuthorizationTokenClient* og bruker ThingPark DX Admin API til forespørslene som omhandler tokenene. Klassen inneholder metoden som genererer en autorisasjonstoken og metoden som henter ut data om en allerede eksisterende token. Klassen krever en klient-id og tilhørende passord for å kunne tas i bruk. På den måten kan man opprette forskjellige autorisasjonstokener for de ulike rollene. Det er nødvendig å opprette en token for leverandørbrukeren som har administrasjonstilgang på applikasjonen, og en token per bruker som ønsker å kjøpe produkter og tjenester.

Autorisasjonstoken inneholder informasjon som utløpsdato og selve tilgangstokenen som trengs for å bruke ThingPark DX Core API-en. Denne tilgangstokenen må plasseres inn i en bærertoken som skal bli sendt sammen med HTTPS-forespørslene i en autorisasjonslinje. En bærertoken er en sikkerhetstoken som tilhører enhver part som er i besittelse av tokenen, altså bæreren av tokenen. Bæreren kan bruker tokenen på en hvilken som helst måte som enhver annen part som er i besittelse av den. [35]

To andre felt må også bli definert for at forespørslene skal fungere. Kodelinjene 3.3 viser konstruktøren til klassen som fungerer som DX Core klient. Der blir de nødvendige headerene, som må bli satt for å kunne bruke DX

Core API, lagt til ved hjelp av en tilgangstoken. Denne kommer fra en autorisasjonstoken.

Kode 3.3: Snippet som viser headerene som må bli satt for å kunne sende forespørsler til DX Core API

```
1 class dxCoreClient:
2
3     def __init__(self, access_token):
4         self.headers = {
5             "Authorization": "Bearer %s" %access_token,
6             "Accept": "application/json",
7             "Content-Type": "application/json"}
```

3.7 Thingpark DX Core API-funksjoner

Det trengs to tilgangstokener for å utføre alle handlingene som trengs for å behandle et kjøp som inkluderer abonnementer. Den ene har leverandørtilgang og blir opprettet ved behov. Nødvendig informasjon blir hentet i Azure Key Vault. Den andre tokenen har abonnenttilgang knyttet opp mot abonnentbrukeren til den spesifikke kunden som gjennomfører kjøpet.

Målet med denne delen er å gjøre det lettest mulig for brukeren, og administratorer, å gjennomføre handlinger under kjøpet. Prosessen skal automatiseres slik at mest mulig av prosessen skjer av seg selv på tjenersiden uten at brukeren eller administratorer trenger å gjør noe særlig. Denne seksjonen forklarer hvordan man bruker disse tilgangene til å utføre sentrale konsepter i applikasjonen.

3.7.1 ThingPark DX Core API-klient

Alle metodene som er knyttet direkte til ThingPark er laget i en egen klasse som heter *dxCoreClient*. Denne klassen finner man i filen *dxCore_ client.py*. Alle metodene er ulike API-kall som er knyttet til ThingPark DX Core API. Alle metodene krever de samme headerene for å fungere. Det er derfor de samles i en klasse, slik at denne headeren allerede blir definert i konstruk-

tøren.

Felles for alle metodene er at de bruker et bibliotek i Python som heter *Requests* til å kommunisere med API-en. Dette biblioteket krever en URL til API-en for å sende HTTP-forespørsler, i tillegg til at man må spesifisere HTTP-verbet til kallet. URL-en er bygget opp av en *base url*, et metodenavn og eventuelle parametere som spesifiserer ønsket informasjon og tilpasninger. Oppbygningen av en typisk URL til ThingPark er vist i figur 3.8. Dersom det dreier seg om en POST-forespørsel, trengs det også variabler som skal sendes. Disse legges til ved data=variabler. Variablene må være en streng formet som et "dictionary".

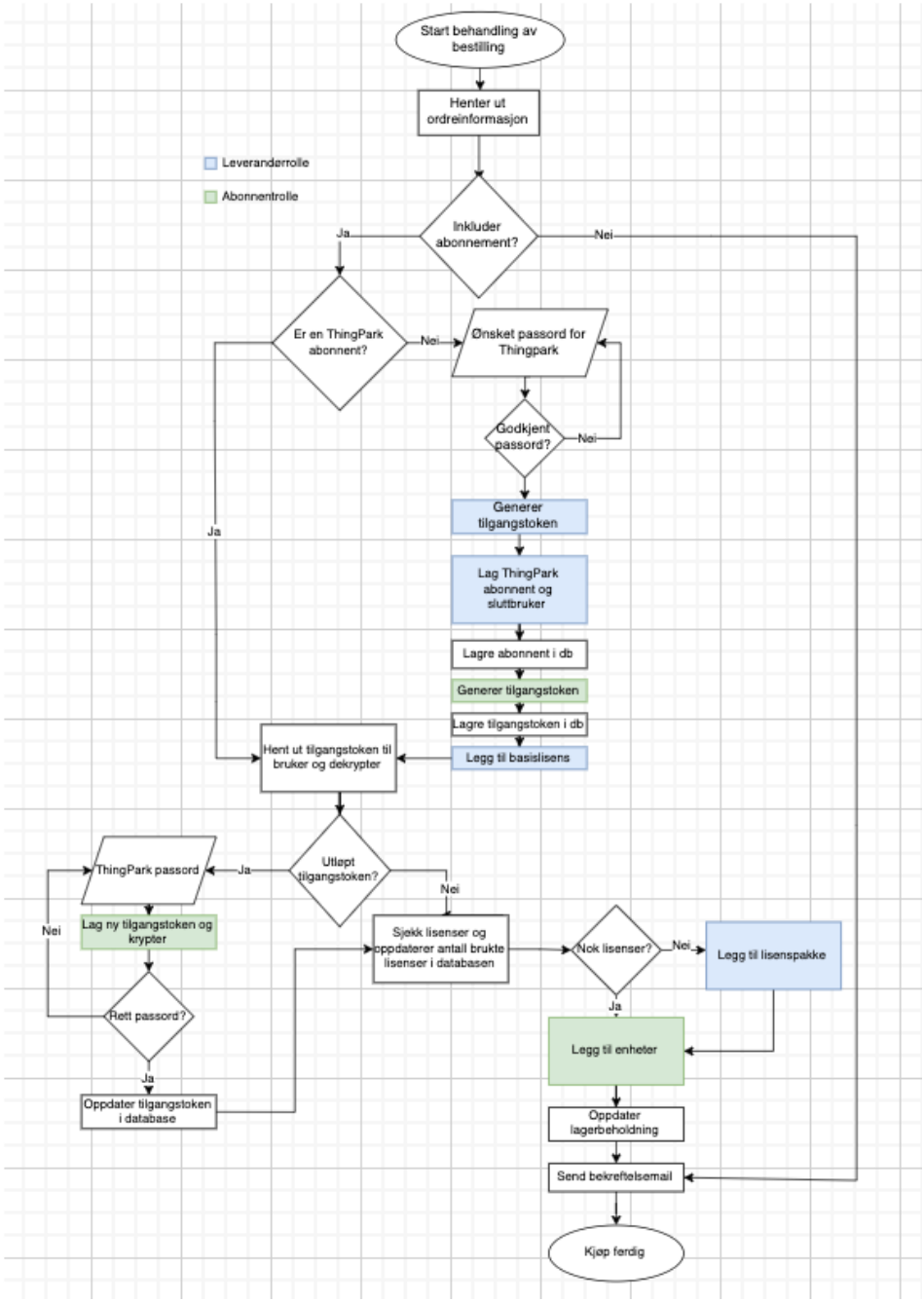
Figur 3.8: Oppdeling av URL-en som brukes ved en HTTP-forespørsel

API-en dekker alle CRUD-operasjoner for de fem objektene som brukes av applikasjonen. Gruppeoperasjonene gir muligheten til å administrere alle ThingPark-rollene, som vil si leverandører og abonnementer. Brukeroperasjoner gir muligheten til å administrere alle brukerne i ThingPark. Abonnementoperasjoner gir muligheten til å administrere tilbud i ThingPark i tillegg til abonnementbestillinger. Disse tilbudene er de ulike lisensene som blir brukt. Man må være en leverandør for å kunne opprette nye. Dette kan være aktuelt i fremtiden dersom man vil tilby forskjellige abonnementstørrelser. Per nå er det kun en vanlig lisens som inneholder tilgang til ti plasser. Det er i abonnementbestillingene man legger til de ønskede enhetene. Denne delen av API-en dekker alt som har med tilkoblingen å gjøre [20]. Hvordan disse operasjonene blir brukt av applikasjonen blir forklart under.

3.7.2 Behandling av bestillinger

Figur 3.9 er et flytdiagram som illustrerer prosessen som skjer i det en bruker trykker "Fullfør kjøp"-knappen i handlekurven. Det er hovedsakelig her API-kallene tas i bruk. De blå figurene symboliserer handlinger som er avhengig av leverandørtilgang for å bli kjørt, mens de grønne figurene

symboliserer handlinger som krever abonnenttilgang. Dersom noen av metodene fra ThingPark blir brukt feil, vil man motta en feilmelding i form av en streng som er formatert som et "dictionary". Den vil da ha formen {code: xxx, message: xxx}, hvor koden er en standard HTTP-kode og meldingen inneholder feilmeldingen. Man må derfor hente ut denne koden for å kunne håndtere feilmeldinger på rett måte.



Figur 3.9: Viser prosessen som skjer i det en bruker trykker "fullfør kjøp"-knappen

Som vist i flytskjemaet henter *ProcessOrder*-metoden informasjonen om brukeren vil inkludere abonnement eller ei. Dersom brukeren ikke ønsker å inkludere dette, vil metoden hoppe over alle API-kallene som er knyttet opp mot ThingPark. Hvis brukeren derimot ønsker abonnementer, vil tilkoblingen til LoRaWAN starte.

ProcessOrder henter også ut adresseinformasjon og handlekurven. Handlekurven blir lagret i en informasjonskapsel hos klienten. Informasjonskapselen vil se slik ut *cart: "{"contents":{"sensorId"-antall, "sensorId2"-antall....}}"*. Et eksempel på handlekurven vil være
cart:"{"contents":{"1":2,"2":2}}". I dette eksempel har brukeren 2 stk av sensor med sensor-id 1 og 2 stk av sensor med sensor-id 2. "Cart" er navnet på informasjonskapselen, mens resten er verdien til informasjonskapselen.

Man trenger kun en abonnentbruker på ThingPark første gang man kjøper sensorer som inkluderer et abonnement. Det er derfor ikke nødvendig å opprette en bruker på ThingPark før den første gangen man kjøper noe som inkluderer et abonnement. Dersom en abonnentbruker blir opprettet hver gang en ny bruker tar i bruk nettsiden, hadde det ført til unødvendige mengder med brukere på ThingPark som ikke trenger å være der. Når koden oppretter en abonnent første gang, vil brukeren få opp en dialog der de kan skrive inn sitt ønskede passord for ThingPark. Dette blir brukt til å opprette en abonnent og en sluttbruker med det valgte passordet.

Det er dette passordet de bruker for å logge seg inn på brukerportalen til ThingPark. Når man oppretter en ny bruker vil det også lages en tilgangstoken ved hjelp av passordet som ble skrevet inn. Denne blir kryptert ved AES128 før den lagres i databasen. Tokenen lagres på grunn av brukervennligheten. Tilgangstokenen trengs hver gang en bruker gjennomfører et kjøp med abonnement. Så i stedet for å spør brukeren om passordet hver gang de kjøper noe, vil man kunne hente ut tilgangstokenen fra databasen i stedet. Noe av abonnentinformasjon lagres også i databasen. Dette trengs for å bruke noen av de andre API-kallene senere. En basislisens vil også bli lagt til den nye brukeren via et tilbudskall.

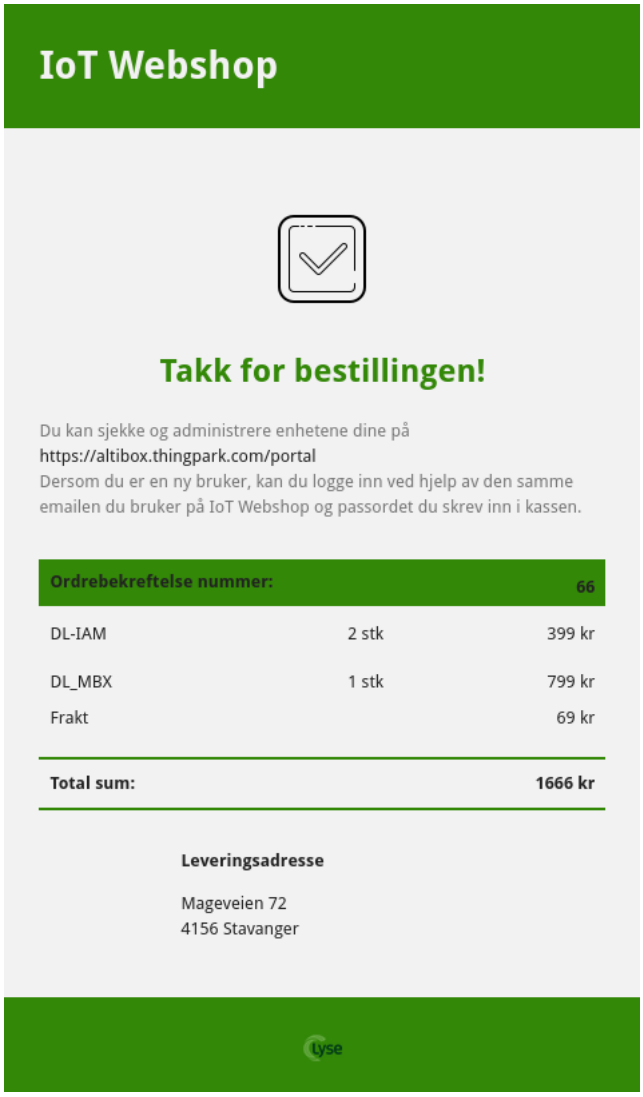
Etter at abonnenthåndteringen er utført, sjekker applikasjonen om tilgangstokenen til brukeren er gyldig. Det er satt en utløpsdato på 90 dager for hver token. Dersom denne skulle være utløpt, vil brukerne få opp en ny dialog der de blir bedt om å skrive inn passordet sitt for ThingPark. Passordet vil

på nytt bli brukt til å generere en ny tilgangstoken. Denne erstatter den gamle tokenen i databasen.

Videre vil det komme en prosess som holder styr på alle lisensene til en bruker. Her vil det komme en sjekk på hvor mange sensorer det er som krever lisenser, og oppdatere det totale tallet for antall sensorer i databasen. Deretter sjekker metoden om det eksisterer nok lisenser til å legge til alle sensorene. Hvis det ikke er nok plasser igjen med de nåværende lisensene, vil en ny lisenspakke bli lagt til ved hjelp av et bestillingskall med leverandørstatus.

Det siste steget som inneholder ThingPark er å legge til alle sensorene. Her vil man gå igjennom hele bestillingen og legge til enhetene med rett informasjon. Hver enhet mottar automatisk genererte verdier for *ApplicationKey* og *EUI* i tillegg til spesifikke parametere knyttet til sensoren og type nettverkstilkobling. De genererte verdiene kan lett byttes ut med Lyse sine data når det blir tilgjengelig. Da vil disse verdiene bli hentet ut av databasen i stedet for.

Til slutt vil applikasjonen bruke informasjonen den har hentet ut for å generere ordrebekreftelsen og automatisk sende denne som en bekreftelsesmail til brukeren. Figur 3.10 viser hvordan denne e-posten ser ut. Varebeholdningen blir også oppdatert for å reflektere reduksjonen av varelageret. Ved en senere anledning ville varelageret blitt oppdatert med akkurat hvilke sensorer med sensornummer som ble kjøpt, og ikke antallet av den typen. Etter at prosessen er ferdig, kan brukeren sjekke enhetene sine i brukerportalen til ThingPark.



Figur 3.10: Viser bekreftelsen man mottar på epost.

Kapittel 4

Økonomisk oversikt

Microsoft tar betalt for de forskjellige tjenestene de tilbyr. Dette kapittelet vil ta for seg de økonomiske kostandene som forekommer under utvikling og for en eventuell storskala produksjon. Det er forskjell i kostnadene som forekommer under utvikling og produksjon, ettersom det er forskjellige krav til nettsiden og databasen. Fordelen med Azure er at alle studenter kan få en studentkonto som inneholder 100 USD. Disse kan brukes på deres tjenester.

4.1 Utviklingskostnader

Som nevnt i starten av kapittel 3 så tar backenden litt tid på å starte opp, fordi den bruker en gratis utviklingsversjonen av nettsiden. Denne fungerer fint under utvikling. Gratisversjonen støtter lagringsplass inntil 1 GB. Frontenden benytter seg og av en gratisversjon under utvikling. Denne støtter lagringsplass inntil 0,5 GB. Dette har også fungert fint under utvikling. Ettersom bildene blir lagret i databasen, og ikke som filer på tjeneren, vil ikke frontend-tjeneren ta mye lagringsplass.

Under utvikling er det kun databasen som har kostet penger. Fra januar til mai har databasen kostet 19 dollar. Databasen som blir brukt under utvikling har en makskapasitet på 2 GB. Dette ville nok ikke holdt til kravene for produksjon, ettersom det sannsynligvis blir mye mer data på produk-

sjonsnettsiden. Da blir det lagret mye mer data for alle brukerne, alle bestillingene og sannsynligvis flere enn de tre sensorene som har blitt brukt til nå. Det vil sannsynligvis vært mulig å gjøre databasen enda billigere under utvikling ved annen konfigurasjon. Ettersom Azure tilbyr studentkontoer med 100 USD, ble ikke reduksjon av kostanden et stort fokus da gruppen så at kostandene til databasen holdt seg langt under denne grensen.

4.2 Produksjonskostnader

Som nevnt i underkapittel 4.1 kreves det mer fra nettsidene og databasen i produksjon enn under utvikling. Databasen vil som regel være det dyreste elementet knyttet til en nettside. Prisen vil avhenge ut i fra hvor mye lagringsplass Lyse vil trenge på nettsiden. Dette vil igjen variere i forhold til hvor mange sensorene Lyse vil ha på nettsiden. Det vil også variere dersom de vil ha flere bilder per sensor, og ut i fra hvor mange bestillinger de tror de vil få. Bilder krever mye lagringsplass.

Backend og frontend må også oppgraderes dersom nettsiden skal ut i produksjon. Da må man betale for å slippe unna dvalefunksjonen. Selv om dvalen er tolererbar under utvikling, er ikke dette akseptabel under produksjon. En rimelig nettside på backend vil koste 66,16 USD i måneden. Dette inkluderer 10 GB lagring og 1,5 GB ram. [36] Her må Lyse se etter hvor mye trafikk de forventer på nettsiden deres. Frontenden må også bli oppgradert fra gratisversjonen. Her vil det sannsynligvis holde med standardversjonen som koster 9 USD i måneden. Her har Azure forskjellige pakker som Lyse må se på for å finne ut hva som passer best til deres krav. [37]

Kapittel 5

Miljøregnskap

I løpet av de siste årene har det kommet mer og mer fokus på bærekraft og innvirkningen handlingene våre har på miljøet. Selv en nettside kan ha negativ konsekvens på kloden i form av strømforbruk, frakt av produkter og utbyggelse av IoT-nettverk. Likevel oppfordrer applikasjonen til økt innovasjon av nye bærekraftige løsninger på miljøproblemer, i tillegg til utviklingen av den moderne smartbyen. Denne seksjonen drøfter hvordan nettsiden i seg selv påvirket miljøet, i tillegg til hvordan økt tilgjengelighet av IoT-løsninger setter sitt preg på omgivelsene rundt oss.

5.1 Inngrep i naturen

Det er mulig å drøfte konsekvensene produktet har på miljøet fra to ståsteder. Den ene er konsekvensene fra bruken av selve applikasjonen, mens den andre er hvordan bruken av applikasjonen indirekte oppfordrer til økt bruk av IoT-løsninger.

5.1.1 Ressursbruk for nettsiden

Ressursbruken av strøm er blant de mest åpenbare inngrepene nettsiden har på naturen. Her må man ta hensyn til strømforbruket til alle partene involvert i bruken av nettsiden, i tillegg til karbonutslippet som oppstår ved utnyttelsen av denne ressursen. Hvordan energien produseres har også stor påvirkningen på miljøet, og valg av vertleverandør kan ha mye å si for mengden CO_2 som blir sluppet ut.

WebsiteCarbon er en nettside man kan bruke for å sjekke karbonutslippet til nettsiden. Utviklerne bak nettsiden har jobbet i mange år med å lage en avgrenset metodikk for å beregne dette. Nettsiden bruker tredje versjon av denne metodikken. Metodikken er utarbeidet sammen med bransjekolleger, slik at de kan utvikle en oppdatert og koordinert tilnærming via forskning. Deres mål med siden er å oppfordre flere til å ta miljøvennlige tilnærminger som utviklere i webindustrien. De vil også øke bevisstheten rundt valgene man tar og effekten de har på miljøet. De vil også gjøre det lettere å oppdage "greenwashing". [38]

I følge nettsiden produserer applikasjonen bare 0,29 gram med CO_2 hver gang noen besøker nettsiden. Dersom man antar at nettsiden får 1000 månedlige sidevisninger i løpet av et år, tilsvarer dette like mye produsert mengde karbon som ett tre klarer å absorbere i løpet av et år. Da er det snakk om at 8 kWh er brukt i løpet av året. Dette er like mye strøm som en elbil bruker på å kjøre 50 kilometer. Dersom siden kun får 100 sidevisninger per måned i løpet av et år, tilsvarer dette fortsatt like mye karbon som et tre klarer å absorbere i løpet av et år. Likevel vil man kun bruke 1 kWh med energi i den samme perioden. Dette er like mye strøm som en elbil trenger for å kjøre fem kilometer. [39]

Datapunktene som blir brukt til å beregne energien og utslippet til nettsiden er dataoverføringen over ledningen, energiintensiteten til nettverksdata, energikilden som brukes av datasenteret, karbonintensiteten til elektrisiteten og trafikken til nettsiden. Det er altså mange faktorer som spiller inn på ressursbruken for driften av nettsiden, og ikke bare strømmen som blir brukt for å holde den aktiv. [38]

Dataoverføring over ledning må tas med i beregningene, fordi dataene som overføres når en nettside lastes inn også trenger ressurser. Energien som

brukes her er cirka proporsjonal med mengden data som overføres. For å beregne konsekvensene av dette blir mengden data som overføres over ledningen multiplisert med energiforbruksdataene som WebsiteCarbon har. Bufferen i nettleseren har også påvirkning på energibruken. Dersom noe er lagret i bufferen på enhetene til brukerne, vil mindre data bli overført over ledningene. Det skjer derfor justeringer i energiberegningene for å veie opp for dette. [38]

Energiintensiteten til nettverksdataen varierer for hver nettside og hver besøkende, så det må brukes et gjennomsnittstall ved beregningen. Mengden energi i beregningene er summen av energien som brukes i datasenteret, telenettverket og sluttbrukerens enhet. [38]

Valget datasenteret har tatt om bruk av energikilde har også mye å si. Datasenteret i vårt tilfelle er Microsoft Azure. For å sjekke om Azure tar i bruk grønn energi brukte gruppen The Green Web Fundation sin hjemmeside. I følge dem blir nettbutikken tilgjengeliggjort ved bruk av grønn energi. Det vil si at Azure enten bruker fornybar energi, eller at de kompenserer for tjenesten sin på alternative måter. [40] Valg av en grønn energikilde resulterer i ni prosent mindre CO_2-utslipp. [39]

5.1.2 Oppfordring til IoT-løsninger

Applikasjonen gjør det lettere for folk å koble opp enheter på et eksisterende IoT-nettverk. Den bidrar derfor med både bruk av enheter og utbyggelse av nettverket. Hver eneste enhet vil produsere og sende en betydelig mengde data. Alle disse dataene vil kreve energi når de passerer gjennom nettverket. Over tid kan dette samle seg opp til å bli av betydelig mengde. I tillegg vil sannsynligvis mange av enhetene fungere ved hjelp av batterier. Jo flere batterier som brukes, jo flere batterier havner på søppelfyllinga. [41] Når batterier korrodere, vil de farlige kjemikaliene i batteriene trekke inn i jorden og videre forurense grunnvann og overflatevann. [42] Store mengder enheter gir store mengder batterier og strømforbruk. Dette kan forårsake et økende problem etter hvert som appetitten for IoT vokser. [41]

Per i dag har LoRaWAN sensornettverket dekning til over 1.000.000 husstander i 100 kommuner Norge. Dette dekker langt fra alle husstandene i landet. Nettsider er med på å øke etterspørselen etter nettverkstilganger og

bidra til at utbyggingen av nettverket fortsetter i samme takt som i dag. [1] Dette blir et inngrep i naturen som nettsiden indirekte har medansvar for. Nettverket krever utplassering av ulike komponenter til nettverket som vanligvis ikke hadde vært der hvis det ikke hadde vært for det menneskelige inngrepet i naturen. Alle disse komponentene er også avhengig av elektrisitet for å fungere, som igjen økes med utvidelsen. Denne energibruken må også tas hensyn til.

Frakten av varene er også et stort problem sammenlignet med kjøp i fysisk butikk. Leveringen av sensorene fører til merkbar økning av forurensing. Det vil bli flere leveringssteder med færre produkter per levering, sammenlignet med å levere et varelager til en butikk. Man må også ta hensyn til eventuelle returer. All denne transporten kan regnes som et av de største miljøproblemene knyttet opp mot nettbutikker.

5.2 Bærekraftig utvikling

I følge WebsiteCarbon er nettsiden renere enn 68% av nettsidene som er testet. [39] Det er likevel 32% av de testede nettsidene som er grønnere. Nettsiden i seg selv har altså et forbedringspotensiale. Likevel oppfordrer nettbutikken til modernisering og effektivisering av eksisterende løsninger. Disse smarte løsningene er grunnen til at bruken av IoT faktisk kan drøftes til å være bra for miljøet i den store sammenhengen.

IoT-enheter kan føre til betydelige energibesparelser ved for eksempel å regulere gatelys slik at de kun skrur seg på ved behov. Det kan også brukes til å varsle vannbehandlingsanlegg når det er en lekkasje, slik at man kan spare dyrebare naturressurser. [41] Dette er kun noen få eksempler på hvilke problemer IoT kan løse. Det kan derfor argumenteres for at IoT veier opp for energibruken ved at teknologien hjelper miljøet mer enn det det påvirker miljøet negativt. I følge ThingPark kan IoT forbedre teknologi og hjelpe med utvikling av nye løsninger innenfor flere viktige områder som helse, jordbruk, sikkerhet, transport, industri, energiproduksjon og anleggsledelse. [2] Mange av disse områdene kan regnes som miljøverstinger, og en forbedring innenfor disse vil bidra mye til en bærekraftig utvikling.

Kapittel 6

Evaluering

6.1 Endring av oppgave underveis

Etter at oppgaven ble gitt fra Lyse er det fire punkter som har blitt tatt vekk på grunn av praktiske omstendigheter og fra valg Lyse tok underveis mens de så utviklingen av produktet. Grunnen til dette er at det er viktig å tilpasse nettsiden kontinuerlig etter kundens krav. Etterhvert som nettsiden blir mer og mer ferdig er det lettere for kunden å se hva de trenger og hva de ikke trenger allikevel. De fire punktene som har falt bort underveis vises i figur 6.1.

- Mulighet for å kunne bekrefte bestilling
- Mulighet til å sende bestilling til pakking på varelager
- Mulighet til å generere pakkseddel for sending
- Sørge for at ved bestilling av sensorer velges disse fra lageret

Figur 6.1: Punkter som har falt vekk

Lyse fant ut at det ikke var behov for at en administrator skulle bekrefte bestillinger. Man mister mye av automatikken når man trenger menneskelig

godkjennelse av noe som koden kan gjøre automatisk. Likevel kan administrator se alle bestillingene på administratorsiden. Dette gir en oversikt for administratorene, slik at de kan oppdage feil i bestillingene her.

I et møte med Lyse ga de beskjed om at lagersystemene deres var under utvikling, og at det ikke ville rekke å bli ferdig tidsnok til denne oppgaven. Det førte til at lagerbeholdningen til hvert produkt bare er ett tall som administratorene skriver inn. Det minsker ved bestilling, og administratorene kan endre det når de vil. Dette kan simulere en økning, eller en reduksjon, i varelageret. Problemet er at tallet ikke har en sammenheng med det faktiske varelageret, ettersom Lyse sitt lagersystem enda er under utvikling. Derfor sa Lyse at punktene som innebar lageret kan falle bort, dersom det blir implementert et dynamisk tall som skal forestille lagerbeholdningen. Det har ført at punkt 2-4 fra figur 6.1 ikke lenger er relevant for oppgaven.

Ettersom noen punkter har blitt tatt vekk, har det også blitt lagt til funksjonaliteter. Lyse hadde krav om at det skulle være en handlekurv, men de var klare på at nettsiden også skulle ha funksjonalitet som er normale for nettbutikker. Derfor har det også blitt lagt til favoritter og andre generelle funksjonaliteter. Grunnen til at favoritter har blitt lagt til er for at handlekurven blir lagret som en informasjonskapsel i 7 dager, mens favoritter blir satt i databasen på brukeren. Det vil si at favoritter blir lagret helt til brukeren selv fjerner dem. Det kan være fint for brukere å kunne lagre visse produkter som favoritter.

6.2 Utfordringer

6.2.1 JWT og samesite

En av utfordringene som ble møtt var ved innloggingen. Som nevnt i underkapittel 3.5 blir det benyttet tokener for å autentisere innloggede brukere. Dette skjer når en bruker blir logget inn, så får de et svar fra tjeneren som setter en informasjonskapsel som inneholder tokenen. Uten gyldig konfigurasjon vil informasjonskapselen sin samesite egenskapen være ”Lax” i noen nettlesere. I chromiumbaserte nettlesere, slik som Google Chrome, vil samesite egenskapen være ”Lax”. Dette gjør at informasjonskapselen vil fungere

i Firefox, mens chromiumbaserte nettlesere vil avvise informasjonskapselen. De avviser informasjonskapselen fordi samesite egenskapen er "Lax".

Dersom nettleseren avviser informasjonskapselen, vil den ikke inneholde JWT. Dermed vil ikke innloggingen fungere. Derfor er det viktig å definere samesite egenskapen til "None". Dersom egenskapen er "None", vil informasjonskapselen bli sendt i forespørsler mellom forskjellige nettsider. I denne oppgaven vil det være frontend og backend. Det kreves også at nettsidene skal benytte seg av HTTPS eller være localhost. Samesite egenskapen kan bli satt ved *app.config["JWT_COOKIE_SAMESITE"] = "None"*. Her er det viktig at variabelen er lik "None" som en streng, og ikke None som typen i Python. Dersom den er lik None som typen i Python, vil den bli satt til den standard verdien som i noen tilfeller er "Lax". Da får man samme problem.

6.2.2 SQLite

I starten av prosjektet var planen at det skulle være en egen database til utvikling. Grunnen til dette var for å slippe å bruke VPN og hvitliste en ny IP-adresse for hver dag nettsiden skulle videreutvikles. I tillegg vil man ikke risikere å endre databasen i produksjon ved et uhell under utvikling. Gruppen ville ha utviklingsdatabasen med SQLite, fordi dette var enkelt å implementere. Det var heller ikke store krav koblet til utviklingsdatabasen.

Man møtte på problemer med SQLite, fordi databasen har litt annerledes syntaks enn Azure SQL Database. Det funket fint i starten med enkle insert- og selectoperasjoner, men forskjellen med syntaksen kom frem ettersom flere avanserte operasjoner ble brukt. Det ble prøvd å bruke forskjellige SQL-setninger etter hvilket miljø tjeneren ble kjørt i. Det førte til at det bare kom nye problemer hver gang et problem ble løst.

Løsningen ble å bruke samme database til produksjon og utvikling. Som nevnt er dette en ulempe, siden man med et uhell kan ødelegge produksjonsdatabasen mens man utvikler. Det kan likevel gi en fordel. Det skyldes at produksjonsnettsiden også er en testnettside i dette prosjektet. Det vil si at det ikke er så viktig dersom databasen blir midlertidig nede.

Fordelen man får med å ha samme databasen er at man slipper å endre all

dataen i databasen to ganger. Det blir fort mye data å legge til, fjerne og endre dersom man må gjør alt to ganger. Her kan man teste ved hjelp av samme bruker uansett om det er på produksjonsnettsiden eller utviklingsnettsiden. Dersom produksjonsnettsiden faktisk blir brukt under utvikling ville en separert database vært nødvendig. Da kunne man ha konfigurert to ulike databaser ut i fra hvilket miljø man var i og endret dette med en variabel. Kommandoene *export FLASK_ENV=development* og *export FLASK_ENV=production* fungerer bra til å definere miljøet man jobber i.

6.2.3 ThingPark tilgangstoken

Når forskjellige forespørsler blir sendt til ThingPark DX API-en, blir de sendt med forskjellige roller ut i fra hvem som sendte forespørselen. For å registrere sensorer må man ha abonnentrolle. For å ha abonnentrolle må man ha en tilgangstoken til en bruker. For å hente ut en tilgangstoken må man ha innloggingsinformasjonen til en sluttbruker som er registrert på brukeren sin abonnement på ThingPark. Det funket fint første gang en bruker kjøpte sensorer. Grunnen til dette er fordi da blir abonnementet og endebrukeren laget og man har dermed passordet. Problemet oppstår neste gang en bruker kjøper sensorer med abonnement, fordi man da ikke har tilgang til passordet. Utenom å spør etter passordet hver gang, så er det tre forskjellige løsninger på dette problemet som figur 6.2 viser.

1. Lagre et kryptert passord til brukeren i databasen.
2. Lage en administratorbruker på abonnenten.
3. Lagre tilgangstokenen kryptert i databasen.

Figur 6.2: Alternativer til ThingPark tilgangstoken utfordring

Det ene alternativet var å kryptere ThingPark passordet og lagre det i databasen. Dette ville vært det enkleste måten å løse det på, men samtidig er det sannsynligvis den dårligste. Det er en stor sikkerhetsrisiko ved å lagre passord i databasen med kun kryptering. Man kan ikke lagre det som en hash, fordi man trenger å dekryptere det for å få tak i ThingPark tilgangstokenen. Det er også funksjonelle problemer med denne løsningen. Dersom brukeren

endrer passordet sitt på ThingPark, vil det ikke bli endret i databasen. Da vil det gamle passordet i databasen ikke fungere lenger.

Det andre alternativet var å lage en administratorbruker som en sluttbruker på abonnentbrukeren til brukeren. Dette gjøres når abonnementet blir laget. Da lager man to brukere på abonnementet. En normal bruker med brukerens innloggingsinformasjon og en administratorbruker. Fordelen med denne løsningen er at man ikke trenger noe innloggingsinformasjon til brukeren på ThingPark senere, fordi man kan benytte seg av administratorbrukeren i stedet for. Alle administratorene har samme passord som blir lagret ved hjelp av Azure secrets. Et sikkerhetsproblem med dette er at dersom en angriper finner ut passordet, har de tilgang til alle administratorbrukerne på alle abonnentene. Et annet problem er at brukerne har mulighet til å slette alle brukerne registrert på sitt abonnement. Det vil si at en bruker kan slette adminsitratorbrukeren. Dette kan ikke gjøres noe med, fordi det skjer på ThingPark. Dersom administratorbrukeren blir slettet, vil ikke nettbutikken fungere for den brukeren lenger. Dette er altså ikke aktuelt.

Det tredje alternativet var å kryptere tilgangstokenen og lagre den i databasen. Tilgangstokenen har en levetid som har blitt konfigurert til 90 dager. Dersom tokenen er utgått, vil brukeren bli spurt om ThingPark-passordet sitt. Fordelen med dette er at brukeren ikke nødvendigvis blir spurt om ThingPark passordet hver gang han/henne kjøper sensorer, men kun når tilgangstokenen har utløpt. Å skrive inn passordet hver gang kan være plagsomt for brukeren og minsker brukervennligheten. Det er en liten sikkerhetsmessig utfordring med at tokenen blir kryptert og lagret i databasen. Dersom en angriper får tak i den og greier å dekryptere den, kan angriperen gjøre endringer på ThingPark abonnenten ved hjelp av ThingPark DX API-en. Likevel er det verdt det med tanke på brukervennligheten man får ved å slippe å spørre brukeren om passordet så ofte. Derfor falt valget på å kryptere tilgangstokenen og lagre den i databasen.

Tilgangstokenen kunne også hatt uendelig utløpstid, altså at den aldri gikk ut på dato. Da hadde brukeren aldri trengte å skrive inn passordet på nytt. Ulempen med dette forslaget er at man aldri kan oppdatere passordet noen gang hvis man bruker en ”uendelig” token. Dermed kan en bruker aldri bytte passord eller spørre om et nytt passord dersom de skulle glemme sitt eksisterende. Det vil derfor måtte utarbeides en egen sikkerhetspolicy i en situasjon hvor passord aldri utløper. Dette skaper flere problemer enn det

løser, og også flere sikkerhetproblemer. Derfor ble utløpstiden satt til 90 dager.

6.2.4 Aktivering av lisenser

Et problem som oppsto under den automatiske prosessen skyldes aktiviseringen av lisensene i forhold til hverandre. Dette innebærer at kjøpet inkluderer abonnementtilgang. Alle lisensene henger sammen, og må utføres i rett rekkefølge for å kunne fungere. Man kan heller ikke legge til enheter dersom disse lisensene ikke er på plass, så oppkoblingen av enhetene er også avhengig av statusen på lisensene. Når en ny abonnent blir opprettet første gang, legger programmet til en basislisens. Det tar mellom 1 til 4 sekunder fra lisensen er lagt til abonnenten, til at statusen er endret fra *INIT* til *ACTIVE*. Sensorpakkelisensen kan kun legges til en abonnent dersom abonnenten har en aktiv basispakke. Hvis denne sensorpakken ikke legges til, vil det bare stå "No connectivity plan" i ThingPark. Uten sensorpakken, vil man få en feilmelding på at man ikke har nok lisensplasser til å legge til en enhet. Ingenting vil derfor funke som forventet hvis basispakken ikke blir aktiv.

Den foreløpige løsningen for dette problemet er å legge inn en time.sleep-funksjon som varer i fire sekunder. Fire sekunder er den lengste tiden det er registrert at en lisens trenger på å bli aktiv. Funksjonen sørger for at lisensen alltid blir aktiv før man legger til flere lisenser, men det er ikke en optimal løsning å stoppe prosessen på denne måten. Det fører også til ekstra venting for kunden før den får opp en bekreftelse. Siden dette problemet kun skjer ved brukerens første bestilling, er prosessen kun lenger første gang. Ved alle andre kjøp vil prosessen skje raskt. Selv om løsningen ikke er optimal, ble løsningen likevel beholdt på grunn av dette.

En alternativ løsning kunne vært å videreføre brukeren til siden med bekreftelsen før prosessen var ferdig. Brukeren er ikke involvert i prosessen etter at tilgangstokenen er behandlet, og resten av prosessen kunne da ha skjedd i bakgrunnen. For brukerens del vil prosessen oppfattes kortere.

6.2.5 Større bestillinger

I dagens løsning sjekker prosessen om man trenger flere lisensplasser. Hvis brukeren trenger flere, vil en ny lisenspakke på ti sensorer bli lagt til. Siden dette er den eneste lisenspakken utenom basispakken, vil man ikke kunne legge til mer enn ti enheter per bestilling. Dette skaper et problem med større bestillinger hvor kunden vil kjøpe flere enheter enn antall ledige lisenser pluss ti. Da vil man ikke klare å legge til de siste enhetene som man ikke har nok lisenser til.

Dette er noe som enkelt kan fikses dersom man har forskjellige størrelse på lisenspakkene. Dersom man ønsker 100 enheter, kan man legge til en lisenspakke på 100 enheter i lisenssjekken. Dette ordnes lett ved en utvidelse av nettsiden som innebærer økt kontroll over hvilke lisenser som kan bli kjøpt og brukt. I dette tilfellet ville enten rett type lisens legges til ved kjøp og endre på prisen, eller så kunne kunden fått muligheten til å kjøpe lisenspakker i kassen. Da ville det ha dukket opp en advarsel dersom kunden ikke har nok lisenser tilgjengelig. Siden Lyse allerede hadde bestemt hvilke lisenser som kunne brukes, hadde ikke gruppen mulighet til å opprette flere typer. Lisenstypene er noe Lyse som bedrift må bestemme ut i fra deres strategi. Alternativt kunne man også ha sjekket nødvendig antall lisenser og lagd en for-løkke som legger til nødvendig antall lisensplasser.

6.2.6 Tilfeldig genererte verdier

Den midlertidige løsningen for *EUI* og *AppKey* er å tilfeldig generere verdier frem til Lyse sin informasjon er tilgjengelig. Selv om den første verdien består av 16 tegn med heksadesimaler som gir 16^{16} forskjellige kombinasjoner, og den andre verdien gir 16^{32} kombinasjoner, er det fortsatt mulig at en kombinasjon blir lik som en annen tidligere verdi på et tidspunkt. I verste fall vil to like verdier krasje prosessen, men sannsynligheter for dette er minimal. Dette problemet er kun midlertidig frem til Lyse er klare til å legge til sine egne verdier.

6.3 Fremtidig utvikling

Siden nettsiden fungerer som en PoC for Lyse, vil den naturlig ha flere områder der den kan utvikles videre og forbedres. Noen av disse mulighetene vil bli diskutert under.

6.3.1 Betaling

Siden dette kun er en demo-nettside, har derfor ikke betaling. Et punkt for videreutvikling vil derfor være betalingsformer. Dette er ikke lagt til, siden det er vanskelig å teste om det fungerer uten å faktisk gjennomføre betalinger. Det ble vurdert å bruke Vipps sin API for å gjennomføre betaling. De har ett testmiljø der man kan teste betalinger uten betalingen faktisk blir gjennomført. Problemet med å bruke Vipps er at man må registrere nettsiden hos dem. For at nettsiden skal bli registrert og godkjent, må nettsiden inneholde gyldige salgsvilkår. Det er en god del krav som salgsvilkårene må inneholde (se fig 6.3), og utformingen av dette er noe som Lyse må gjøre og ikke gruppen. Nettsiden har per dags dato ikke salgsvilkår siden det er en demo-nettside. Derfor ble ikke Vipps implementert. Å implementere Vipps som betalingsalternativ er fullt mulig å legge til senere.

- Parter
- Betaling
- Levering
- Angrerett
- Retur
- Reklamasjonshåndtering
- Konfliktløsning.

Figur 6.3: Krav for å bli godkjent av Vipps [43]

6.3.2 Varelager

Som nevnt i delkapittel 6.1 ble det endret fra å være koblet opp mot varelageret, til at det ble satt ett tilfeldig tall som forestilte varelageret. Senere når Lyse sitt varelager blir mer ferdig vil det være mulig å kommunisere med varelageret. Da vil det være mulig å legge til punkt 2-4 fra figur 6.1 som ikke ble fullført. Per dags dato skriver brukeren inn adressen sin før en bestilling. Denne adressen blir lagret i databasen, men blir ikke brukt noe videre. Den vil da bli brukt til pakkseddel og mottakerinformasjon for selve forsendelsen når dette blir implementert.

6.3.3 Lisenser

Lyse sin opprinnelig ide var at nettsiden skulle vise hvordan det var mulig å lage en nettside som automatisk registrerte sensorene på nettverket ved hjelp av API-kall. Her var det tilstrekkelig å kun benytte seg av de lisensene som allerede fantes. Det er likevel mulig å lage nye lisenser og gi brukere valg mellom forskjellige lisenser. Da må koden og logikken endres slik brukere velger mellom forskjellige lisenser. Per dags dato kan de kun velge mellom å ha lisens eller ikke. Det er allerede laget en tabell i databasen til dette formålet. Dersom det blir lagt til ulike lisenser kan disse ha forskjellige priser. De ulike lisensene vil da støtte forskjellige antall sensorer. Nå er det en standardlisens for ti sensorer som automatisk blir oppgradert i henhold til antall sensorer, men de blir ikke fjernet om antall sensorer minker. Det vil være mulig å automatisk redusere antall lisenser etter antall sensorer er registrert på brukeren.

Per dags dato kan administratorer endre på sensorer, kategorier og generelle priser. De kan ikke legge til, endre eller slette lisenser. Dette vil være gunstig å legge til under ”/admin” dersom nettsiden skal benytte seg av forskjellige lisenser. Tabellen finnes allerede i databasen, så det vil være enkelt å legge til noen SQL-setninger for at administratorene skal kunne gjøre disse operasjonene.

6.3.4 Administrasjonsfunksjonaliteter

Det er en del administrasjonsfunksjonaliteter som kan bli implementert senere. Per dags dato blir brukere lagt til som administrator ved hjelp av SQL-setninger. Dette skjer ved en egen tilkobling opp mot databasen, men det vil være mulig å la administratorer gjøre andre brukere til administratorer på nettsiden. Da tar man i bruk brukeroperasjonene på ThingPark.

Det er også muligheter for å legge til administrasjonsfunksjonaliteter knyttet opp mot ThingPark. Nå er det kun en adminstratorbruker, også kjent som en leverandør, på ThingPark. Det vil være mulighet å lage flere av disse brukerne dersom det er ønskelig fra Lyse sin side. Det kan være en fordel dersom de ønsker å ha flere administratorer for å slippe at alle skal dele den samme innloggingsinformasjon.

Nå må administratorene logge seg inn på Thingpark for å gjøre diverse administrative operasjoner, men dette kan være ganske tungvint til tider. Å gjøre disse lett tilgjengelige på samme sted på nettsiden kan derfor være aktuelt. På ThingPark kan administratorer slette ordrer og få en oversikt over alle brukerne. Dette kan også implementeres slik at det kan bli gjort fra nettbutikken ved hjelp av ThingPark DX API-en. Da kan administratorene se alle ThingPark brukerne på nettbutikken, samt slette ordrer. Det å gi administratorer muligheten til å slette bestillinger som ikke er godkjent, kan erstatte behovet Lyse opprinnelig hadde om å kunne godkjenne bestillingene før de ble opprettet.

Dersom administratorer kan slette ordre på ThingPark er det også en mulighet å legge til at normale brukere kan avbryte ordren sin. Her ville det sannsynligvis vært naturlig å begrense når brukeren kan avbryte bestillingen sin. Det kan være gunstig å legge til et status felt på bestillinger. Dermed kan brukere avbryte bestillingen så lenge den ikke er sendt. Et status felt på bestillingene kan også være naturlig å ha med uten denne funksjonaliteten.

Funksjonalitetene over er kun eksempler på hva som kan implementeres over tid. API-ene åpner også for å implementere flere funksjonaliteter.

6.3.5 API-kall

Flere API-kall er laget i *dxCoreClient* enn det som til slutt ble brukt i koden. Grunnen til dette er at alle ble lagd før det var avklart hvilke kall man trengte. I og med at dette er CRUD-operasjoner til grunnleggende ThingPark-funksjonaliteter, ble de tatt vare på. Disse har som formål å bli brukt ved utvidelsen av nettsiden i fremtiden. Alle metodene må brukes for å implementere lignende funksjonaliteter som beskrevet over.

Det finnes mange valgfrie parametere i kallene som kan endre måten enhetene blir registrert på, eller informasjonen som blir lagret om brukeren. Dette gjør at man kan utvide applikasjonen til å legge til ekstra informasjon og gi økte funksjonaliteter inne på ThingPark. Per i dag er ikke all mulig informasjon lagt til på hver enhet, men det kan være aktuelt i fremtiden. Disse tilpassede valgene kan være lokasjonen på enheten, nettverksinnstillinger og bevegelsesindikasjonen til enheten. Bevegelsesindikasjonen er hastigheten enheten beveger seg i. Den kan konfigureres som blant annet "statisk", "sykkelfart" og "kjøretøysfart". Alle de ekstra parameterene forbedrer hvordan man kan bruke enhetene på, og er noe som kan brukes til å utvikle nye funksjonaliteter i fremtiden. [20]

6.3.6 Testing

Programmet benytter seg ikke av automatiske tester selv om dette kan være veldig lurt. Grunnen til dette er fordi mesteparten av koden som blir kjørt i backend er databasemetoder eller forespørsler til ThingPark DX API-en. Dette er ikke enkelt å teste med automatiske tester uten å lage flere forespørsler til ThingPark. Databasemetodene er heller ikke lette å teste uten å faktisk endre selve databasen. Validerings-funksjonene er ett eksempel på noe som burde blitt enhetstestet ettersom det ikke er avhengig av kobling mot en ekstern aktør. Validerings-funksjonene har blitt testet manuelt, men ikke med enhetstesting.

Kapittel 7

Konklusjon

Oppgaven oppfyller hovedpunktene til Lyse. Det er en nettbutikk hvor brukere kan kjøpe sensorer og samtidig automatisk koble dem opp mot et eksisterende sensornettverk. API-kall fra ThingPark er brukt for å implementere dette. Administratorer kan gjennomføre CRUD-operasjoner på sensorer, kategorier og generelle priser. Nettsiden blir levert ved hjelp av Azure og det blir brukt Azure DevOps til kildekodehåndtering. Nettsiden fungerer som en SPA og inneholder brukervennlige funksjonaliteter. Nettsiden fungerer som en *Proof of Concept*, altså en demo, og har derfor et stort potensiale for hvordan den kan utvikle seg videre til en fullskala nettside. Dette gjelder både for koden som allerede eksisterer og videre utvikling av programvaren for Lyse sin del. For at Lyse skal kunne bruke denne i kommersiell sammenheng, er det enkelte ting som må være på plass før nettsiden lanseres. Likevel er en god grunnmur for produktet på plass.

Bibliografi

[1] "Altibox gjør norge smartere." https://www.altibox.no/iot/. Hentet 27.01.2022.

[2] "Get started with our solution catalog." https://community.thingpark.org/iot-solutions-catalog/, 2022. Hentet 27.04.22.

[3] "Lyse - om oss." https://www.lysekonsern.no/om-oss/. Hentet 27.01.2022.

[4] "Lyse - internet of things." https://www.lysekonsern.no/virksomhet/telekommunikasjon/internet-of-things/. Hentet 27.01.2022.

[5] Microsoft, "Cloud computing services | microsoft azure." https://azure.microsoft.com/en-gb/. Hentet 14.05.2022.

[6] "About azure key vault." =https://docs.microsoft.com/en-us/azure/key-vault/general/overview. Hentet 12.03.2022.

[7] "Odbc." https://no.wikipedia.org/wiki/ODBC, 2017. Hentet 22.03.2022.

[8] "Overview." https://cli.vuejs.org/guide/. Hentet 28.01.2022.

[9] Vuetify, "Vuetify 3 beta." https://next.vuetifyjs.com/en/getting-started/installation/. Hentet 13.05.2022.

[10] "Which vue.js version to use in 2021 and why." https://vueschool.io/articles/news/

which-vue-js-version-to-use-in-2021-and-why/. Hentet 27.04.2022.

[11] "Introduction." https://vuetifyjs.com/en/introduction/why-vuetify/. Hentet 28.01.2022.

[12] "Bootstrapvue." https://bootstrap-vue.org/, 2022. Hentet 29.01.22.

[13] Buefy, "Buefy." https://buefy.org/. Hentet 29.01.2022.

[14] "A desktop ui library." https://element.eleme.io/#/en-US. Hentet 28.01.2022.

[15] "Getting started." https://vuex.vuejs.org/guide/. Hentet 28.01.2022.

[16] "How to make a web application using flask in python 3." https://www.digitalocean.com/community/tutorials/how-to-make-a-web-application-using-flask-in-python-3. Hentet 28.01.2022.

[17] "Lora - wikipedia." https://en.wikipedia.org/wiki/LoRa, 2022. Hentet 23.03.22.

[18] R. Thomas, "Advanced encryption standard (aes): What it is and how it works." https://securityboulevard.com/2020/04/advanced-encryption-standard-aes-what-it-is-and-how-it-works/, 2020. Hentet 27.04.2022.

[19] Actility, "Thingpark enterprise overview." https://docs.thingpark.com/thingpark-enterprise/6.1/Content/TPE-overview.htm. Hentet 23.03.22.

[20] Actility, "Dx api platform." https://dx-api.thingpark.com/platform/#dx-api-platform. ThingPark dokumentasjon. Hentet: 22.03.22.

[21] "Access tokens." https://www.oauth.com/oauth2-servers/access-tokens/, 2022. Hentet 29.04.22.

[22] A. Dave, "Api security – how to authenticate and authorise api's in .net 5." https://www.freecodecamp.org/news/authenticate-and-authorize-apis-in-dotnet5/, 2021. Hentet 29.04.2022.

[23] Decentlab, "Dl-mbx datasheet." Brochure, 2022.

[24] Actility, "Provisioning your trackers on thingpark enterprise." https://docs.abeeway.com/thingpark-location/C-Procedure-Topics/ProvisionTrackerTPE_T/, 2022. Hentet 29.04.2022.

[25] S. NZ, "Using your thingpark webportal." https://sparkiot.bitbucket.io/tpw-portal.html, 2019. Hentet 29.04.2022.

[26] ThingPark, "Using your thingpark webportal." https://dx-api.thingpark.com/admin/latest/swagger-ui.html. Hentet 29.04.2022.

[27] N. Selimović, "Abp vs otaa." https://www.thethingsindustries.com/docs/devices/abp-vs-otaa/, 2021. Hentet 29.04.2022.

[28] "Lora – device activation call flow (join procedure) using otaa and abp." https://www.techplayon.com/lora-device-activation-call-flow-join-procedure-using-otaa-and-abp/, 2018. Hentet 13.05.2022.

[29] L. A. Gilbert, "Flask-jwt-extended." https://github.com/vimalloc/flask-jwt-extended, 2022. Hentet 29.04.2022.

[30] J. Flores, "What is social login and is it worth implementing?." https://www.okta.com/blog/2020/08/social-login/, 2020. Hentet 28.04.2022.

[31] Microsoft, "Configure azure static web apps." https://docs.microsoft.com/en-us/azure/static-web-apps/configuration, 2022. Hentet 27.04.2022.

[32] Okta, "Access tokens." https://www.oauth.com/oauth2-servers/access-tokens/. Hentet 13.05.2022.

[33] T. Maher, "Fernet spec." https://github.com/fernet/spec/blob/master/Spec.md, 2014. Hentet 27.04.2022.

[34] "Acid." https://en.wikipedia.org/wiki/ACID. Hentet 13.05.2022.

[35] R. Kumar, "What is bearer token and how it works?." https://www.devopsschool.com/blog/what-is-bearer-token-and-how-it-works/, 2021. Hentet 14.05.2022.

[36] Microsoft, "Priser på app service." https://azure.microsoft.com/nb-no/pricing/details/app-service/windows/. Hentet 07.05.2022.

[37] Microsoft, "Priser på static web apps." https://azure.microsoft.com/nb-no/pricing/details/app-service/static/. Hentet 08.05.2022.

[38] WebsiteCarbon, "How does it work?." https://www.websitecarbon.com/how-does-it-work/. Hentet 07.05.2022.

[39] WebsiteCarbon, "Carbon results for kind-stone-04fae9e03.1.azurestaticapps.net." https://www.websitecarbon.com/website/kind-stone-04fae9e03-1-azurestaticapps-net/. Hentet 07.05.2022.

[40] R. of the green web check — kind-stone-04fae9e03.1.azurestaticapps.net is hosted green!, "How does it work?." https://www.thegreenwebfoundation.org/green-web-check/?url=https%3A%2F%2Fkind-stone-04fae9e03.1.azurestaticapps.net%2F. Hentet 07.05.2022.

[41] IEEE, "Is your iot device harming the environment?." https://innovationatwork.ieee.org/iot-environmental-impact-ieee-wake-up-radio/. Hentet 08.05.2022.

[42] R. services, "Battery recycling is important for environmental health 4 things you should know about battery disposal and the environment." https://gsiwaste.com/

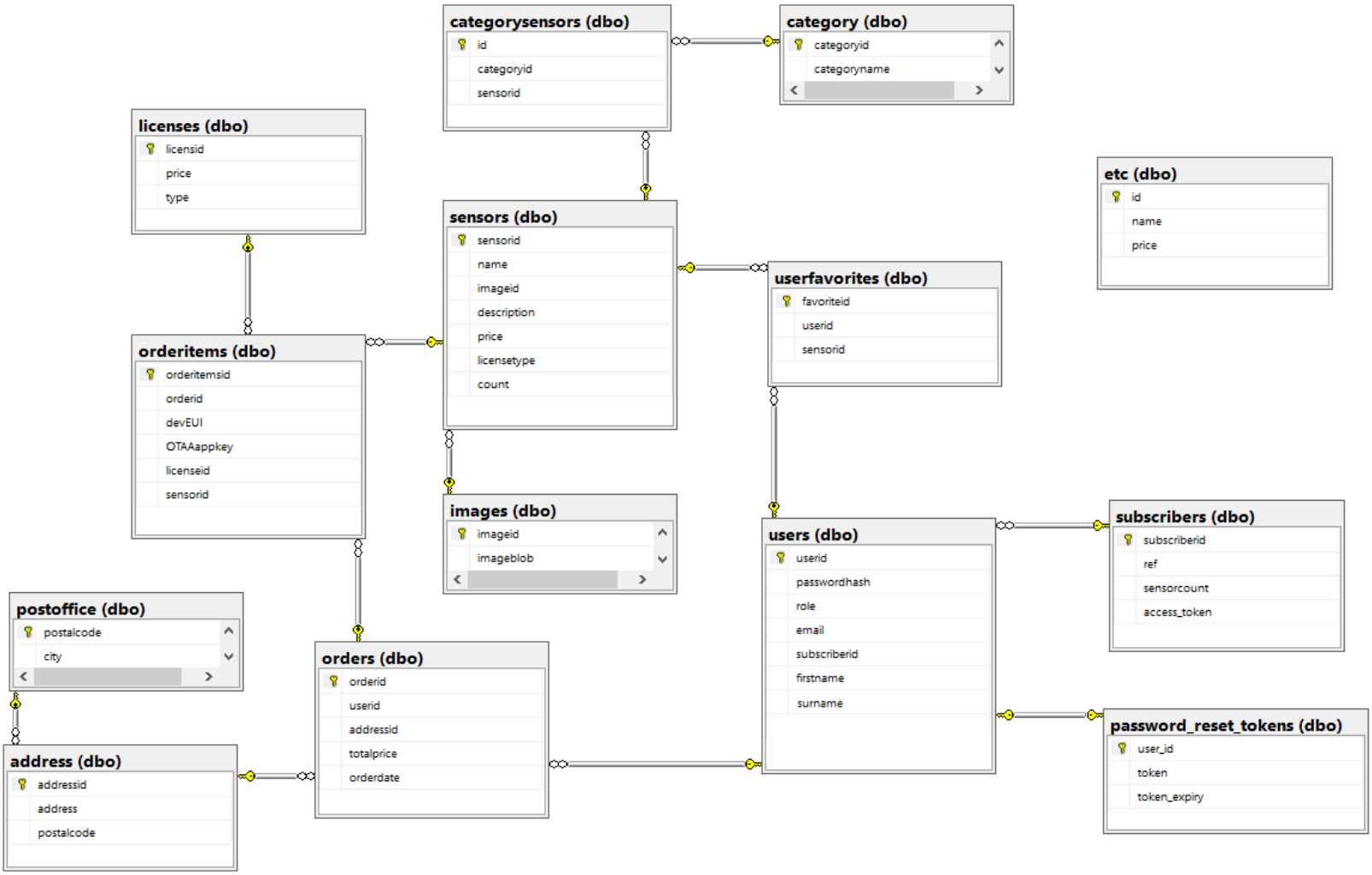
battery-recycling-is-important-for-environmental-health/. Hentet 08.05.2022.

[43] Vipps, "Salgsvilkår." https://www.vipps.no/produkter-og-tjenester/bedrift/ta-betalt-paa-nett/ta-betalt-paa-nett/krav-til-innhold/. Hentet 14.05.2022.

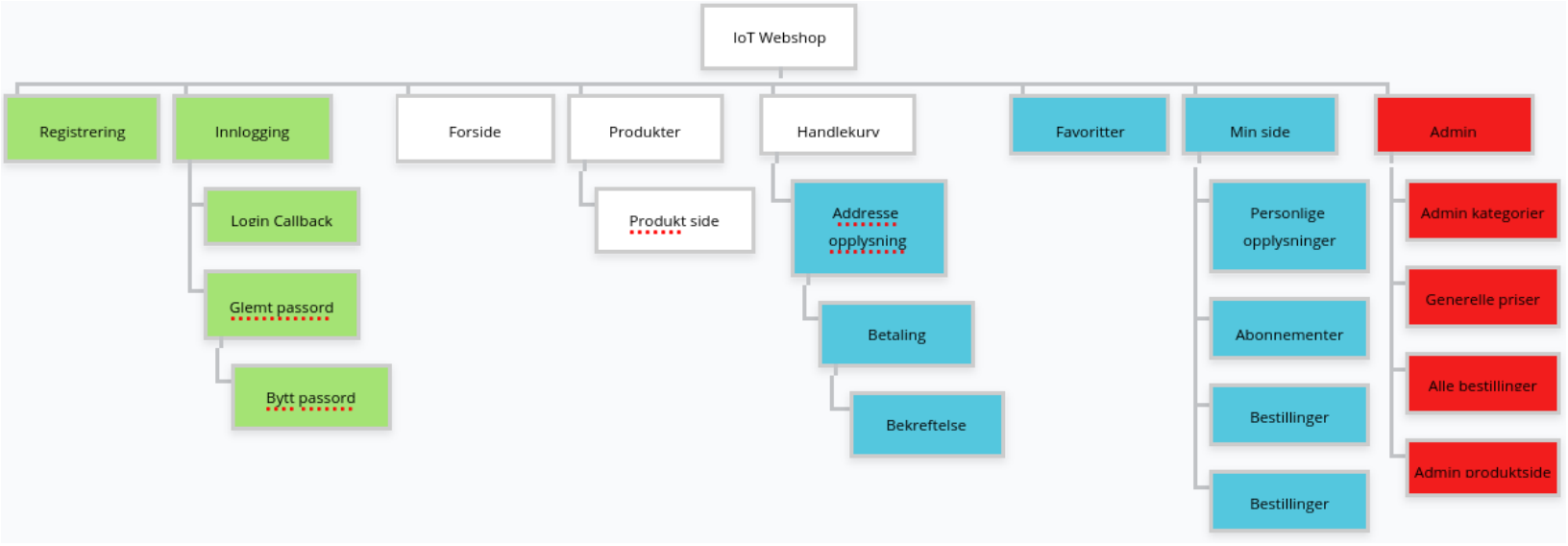
Vedlegg A

Diagrammer

A.1 Databasediagram



A.2 Nettsidekart



Figur A.1: Figur som viser nettsidekartet

Vedlegg B

Kodevedlegg

Kode B.1: Yaml fil for frontend

```
# Node.js with Vue

# Build a Node.js project that uses Vue.
# Add steps that analyze code, save build artifacts, ... deploy, and more:
# ... https://docs.microsoft.com/azure/devops/pipelines/languages/javascript

pool:
  vmImage: ubuntu-latest

steps:
- task: AzureKeyVault@2
  inputs:
    azureSubscription: 'Azure for studenter ... (348c9f6a-cad0-4123-b48d-5f81855103a8)'
    KeyVaultName: 'iotwebshopkeyvault'
    RunAsPreJob: false
- task: NodeTool@0
  inputs:
    versionSpec: '14.18'
  displayName: 'Install Node.js'
- script: |
      cd frontend
      npm install
      npm run build --fix
```

```
24    displayName: 'npm install and build'
25
26  - task: AzureStaticWebApp@0
27    inputs:
28      app_location: 'frontend'
29      api_location: ''
30      output_location: ''
31    env:
32      azure_static_web_apps_api_token: $(azure-static-api)
```

Kode B.2: Yaml fil for backend

```
1  # Python to Linux Web App on Azure
2  # Build your Python project and deploy it to Azure as a ...
       Linux Web App.
3  # Change python version to one thats appropriate for your ...
       application.
4  # ...
       https://docs.microsoft.com/azure/devops/pipelines/languages/python
5
6
7  variables:
8    # Azure Resource Manager connection created during ...
         pipeline creation
9    azureServiceConnectionId: ...
         '02ed9972-47f3-4e89-874b-a9a8f1dce126'
10
11   # Web app name
12   webAppName: 'iot-webshop-backend'
13
14   # Agent VM image name
15   vmImageName: 'ubuntu-latest'
16
17   # Environment name
18   environmentName: 'iot-webshop-backend'
19
20   # Project root folder. Point to the folder containing ...
         manage.py file.
21   projectRoot: $(System.DefaultWorkingDirectory)
22
23   # Python version: 3.9
24   pythonVersion: '3.9'
25
26 stages:
27   - stage: Build
28     displayName: Build stage
29     jobs:
```

```
30        - job: BuildJob
31          pool:
32            vmImage: $(vmImageName)
33          steps:
34            - task: UsePythonVersion@0
35              inputs:
36                versionSpec: '$(pythonVersion)'
37              displayName: 'Use Python $(pythonVersion)'
38
39            - script: pip install --upgrade pip
40              displayName: 'Upgrade pip'
41              workingDirectory: $(projectRoot)
42
43            - script: pip install pipenv
44              displayName: 'Install pipenv'
45
46            - script: python -m pipenv install --dev
47              displayName: 'Install Python dependencies'
48
49            - script: python -m pipenv run pip freeze > ...
                requirements.txt
50              displayName: 'Generate requirements.txt'
51
52            - script: |
53                pip install codecov
54                pip install pytest
55                pip install pytest-sugar
56                pip install pytest-cov
57                pip install pytest-azurepipelines
58                python -m pipenv run pytest ...
                      --junitxml=$(System.DefaultWorkingDirectory)/testResults.xml ...
                       --cov=app --cov-report=xml ...
                      --cov-report=html
59              displayName: 'Run tests with pytest'
60
61            - task: PublishTestResults@2
62              displayName: 'Publish test results'
63              inputs:
64                testResultsFiles: ...
                      '$(System.DefaultWorkingDirectory)/testResults.xml'
65                testRunTitle: '$(Agent.OS) - ...
                      $(Build.BuildNumber)[$(Agent.JobName)] - ...
                      Python $(python.version)'
66                failTaskOnFailedTests: true
67              condition: succeededOrFailed()
68
69            - task: PublishCodeCoverageResults@1
70              displayName: 'Publish code coverage'
71              inputs:
```

```
72                codeCoverageTool: Cobertura
73                summaryFileLocation: ...
                      '$(System.DefaultWorkingDirectory)/**/coverage.xml'
74                reportDirectory: ...
                      '$(System.DefaultWorkingDirectory)/**/htmlcov'
75
76            - task: ArchiveFiles@2
77              displayName: 'Archive files'
78              inputs:
79                rootFolderOrFile: '$(projectRoot)'
80                includeRootFolder: false
81                archiveType: zip
82                archiveFile: ...
                      $(Build.ArtifactStagingDirectory)/$(Build.BuildId).zip
83                replaceExistingArchive: true
84
85            - upload: ...
                  $(Build.ArtifactStagingDirectory)/$(Build.BuildId).zip
86              displayName: 'Upload package'
87              artifact: drop
88
89    - stage: Deploy
90      displayName: 'Deploy Web App'
91      dependsOn: Build
92      condition: succeeded()
93      jobs:
94        - deployment: DeploymentJob
95          pool:
96            vmImage: $(vmImageName)
97          environment: $(environmentName)
98          strategy:
99            runOnce:
100             deploy:
101               steps:
102                 - task: UsePythonVersion@0
103                   inputs:
104                     versionSpec: '$(pythonVersion)'
105                   displayName: 'Use Python version'
106
107                 - task: AzureWebApp@1
108                   displayName: 'Deploy Azure Web App :  ...
                        iot-webshop-backend'
109                   inputs:
110                     azureSubscription: ...
                          $(azureServiceConnectionId)
111                     appName: $(webAppName)
112                     package: ...
                          $(Pipeline.Workspace)/drop/$(Build.BuildId).zip
113
```

```
114                         startUpCommand: 'gunicorn ...
                                --bind=0.0.0.0 --workers=4 ...
                                --timeout 600 startup:app'
```