LAPIN AMK
Lapland University of Applied Sciences

KESKITETTY ROS-PALVELIN

Artic AI & Robotics -hanke

Väisänen Sofie

Opinnäytetyö

Tieto- ja viestintätekniikka
Insinööri (AMK)

2024

LAPIN AMK
Lapland University of Applied Sciences

Opinnäytetyön tiivistelmä

Tieto- ja viestintätekniikka
Insinööri (AMK)

Tekijä	Sofie Väisänen	**Vuosi**	2024
Ohjaaja	Tauno Tepsa		
Toimeksiantaja	Artic AI & Robotics		
Työn nimi	Keskitetty ROS-palvelin		
Sivumäärä	53		

Opinnäytetyön aiheena on kehittää keskitetty ROS-palvelin, joka mahdollistaa useiden ROS-järjestelmää (Robot Operating System) käyttävien robottien yhtäaikaisen käytön ilman tarvetta käynnistää ja alustaa jokaista laitetta erikseen. Tavoitteena on luoda Vue-frameworkilla toteutettu nettisivusto, joka kommunikoi Azure-pilvipalveluiden avulla ROS-laitteiden kanssa reaaliajassa, mahdollistaen niiden hallinnan ja monitoroinnin yhdestä käyttöliittymästä. Sivuston kehityksessä on käytetty pääasiassa JavaScriptiä.

Työn aikana tutustutaan syvällisesti sekä ROS-järjestelmiin että Vue-frameworkin rakenteeseen ja toimintaan. Nettisivusto tarjoaa tehokkaan ja käyttäjäystävällisen ratkaisun ROS-laitteiden hallintaan, mikä vähentää manuaalista työtä ja tehostaa robottilaitteiden käyttöä esimerkiksi teollisuuden sovelluksissa.

Opinnäytetyö toteutettiin Lapin ammattikorkeakoulun IoT-laboratoriossa osana AI.R – Artic AI & Robotics -hanketta, joka on Lapin ammattikorkeakoulun ja Lapin yliopiston yhteinen projekti. Työssä saavutettu ratkaisu tarjoaa merkittäviä etuja ROS-pohjaisten järjestelmien hallintaan ja on askel kohti integroituneempia robotiikan sovelluksia.

Avainsanat pilvipalvelut, robotiikka, verkko-ohjelmointi

LAPIN AMK
Lapland University of Applied Sciences

Abstract of the thesis

Study Programme in Information
and Communication Technology
Bachelor of Engineering

Author	Sofie Väisänen	**Year**	2024
Supervisor	Tauno Tepsa		
Commissioned by	Artic AI & Robotics		
Title	Centralized ROS-server		
Number of pages	53		

The aim of this thesis study was to develop a centralized ROS server that enables the simultaneous use of multiple robots utilizing the Robot Operating System (ROS) without the need to start up and initialize each device separately. The objective was to create a website built with the Vue framework, which communicates with ROS devices in real-time via Azure cloud services, allowing their management and monitoring from a single interface.

The project involved an in-depth exploration of both the ROS system and the structure and operation of the Vue framework. The website development primarily utilized JavaScript. The thesis study was carried out in the IoT laboratory of Lapland University of Applied Sciences as a part of the AI.R – Arctic AI & Robotics project, a collaborative initiative between Lapland University of Applied Sciences and the University of Lapland.

The solution developed in this study offers significant advantages for managing ROS-based systems and represents a step toward more integrated robotics applications.The website provides an efficient and user-friendly solution for managing ROS devices, reducing manual effort and improving the utilization of robotic devices, especially in industrial applications.

Keywords: cloud services, robotics, web development

SISÄLLYS

1 JOHDANTO 7

2 ROS -KÄYTTÖJÄRJESTELMÄ 8

3 HYÖDYNNETYT TYÖKALUT JA TEKNIIKAT 11

3.1 HTML 11
3.2 CSS 12
3.3 JavaScript 13
3.4 Azure-palvelut 14
3.5 Webpack 15
3.6 Vue.js 15
3.7 GitHub ja Git 18
3.8 Canva 19
3.9 Visual Studio Code 19

4 KESKITETYN ROS-PALVELIMEN TOTEUTUS 20

4.1 Sivuston suunnittelu 20
4.2 Oracle VM VirtualBox ja Ubuntu 22.04.4 LTS -asennus 21
4.3 ROS 2 Humble Hawksbillin asennus 21
4.4 Nettisivuston kehitys 22
4.5 Yhteyden muodostaminen robotilta Azureen 33
4.6 Yhteyden muodostaminen sivustolta Azureen 39
4.7 Robotti- ja sivustoyhteyden testaaminen 47

5 POHDINTA 48

5.1 Johtopäätökset 48
5.2 Oman oppimisen pohdinta 49
5.3 Jatkokehittämisaiheet 50

LÄHTEET 52

ALKUSANAT

Opinnäytetyön tekeminen on ollut merkittävä ja opettavainen matka, joka on avannut minulle uusia näkökulmia ja syventänyt ymmärrystäni sekä osaamistani web-sovellusten arkkitehtuurista sekä luomisesta.

Kiitos IoT-laboratorion työntekijöille ja harjoittelijoille, jotka ottivat minut vastaan lämmöllä laboratorioon työskentelemään opinnäytetyöni parissa. Teidän apunne ja henkinen tukenne ovat olleet korvaamattomia.

Haluan myös kiittää perhettäni ja läheisiäni, jotka ovat olleet tukenani koko opinnäytetyöprosessin ajan. Ilman teidän kannustusta ja tukea tämä työ ei olisi ollut mahdollinen.

Kiitokset myös AI.R-hankkeen työntekijöille arvokkaasta panoksesta ja tuesta työn edistyessä.

Kiitos kaikille, jotka olette olleet mukana tällä matkalla.

KÄYTETYT TERMIT JA LYHENTEET

action	asynkroninen kommunikaatiotapahtuma ROS-järjestelmässä, jossa pyydetään toimintoa suoritettavaksi, määritetään toiminnolle haluttu tavoite ja odotetaan takaisin toiminnon lopputulosta
C++	ohjelmointikieli
CSS	Cascading Style Sheet, tyylilomakekieli
distro	disruption, ROS:n jakelu
HTML	Hyper Text Markup Language, merkintäkieli
Java	ohjelmointikieli
JavaScript	ohjelmointikieli, erityisesti web-sivuston toiminnallisuuksien luomiseen
Matlab	numeerisen ja teknisen laskennan ohjelmistoympäristö
node	solmu, yksittäinen ohjelmaprosessi ROS-järjestelmässä
Octave	avoimen lähdekoodin numeerisen laskennan ohjelmistoympäristö
publisher	komponentti ROS-järjestelmässä, joka lähettää viestejä tietystä topicista
Python	ohjelmointikieli
ROS	Robot Operating System
SDK	Software Development Kit, ohjelmistokehityspaketti
service	synkroninen kommunikaatiotapahtuma ROS-järjestelmässä, jossa pyydetään toimintoa suoritettavaksi ja odotetaan vastausta
SPA	Single-Page Application, yksisivuinen sovellus
SSG	Static Site Generation, staattisten sivujen generointi
SSR	Server-Side Rendering, palvelinpuolen renderöinti
subscriber	komponentti ROS-järjestelmässä, joka kuuntelee ja vastaanottaa viestejä tietystä topicista
topic	välityskanava ROS-järjestelmässä, joka mahdollistaa tiedon jakamisen komponenttien välillä

1 JOHDANTO

Robottien käyttö erilaisissa sovelluksissa on yleistynyt huomattavasti viime vuosina. Tämän myötä myös robottien hallintaan ja valvontaan liittyvät tarpeet ovat kasvaneet. Perinteisesti yksittäisten ROS-laitteiden (Robot Operating System) seuraaminen ja hallinta on tapahtunut avaamalla jokaiselle robotille oma ROS-alusta erilliselle elektroniselle laitteelle, kuten pöytätietokoneelle tai kannettavalle tietokoneelle. Tämä menetelmä mahdollistaa roboteilta tulevan datan vastaanottamisen topicien kautta, mutta se on hankalaa, koska se vaatii ylimääräistä työtä ja useita elektronisia laitteita erityisesti silloin, kun robotteja on useampia ja niitä pitää seurata yhtäaikaisesti. Opinnäytetyön tavoitteena oli kehittää Lapin ammattikorkeakoulun AI.R-hanketta varten verkkosivusto Vue-frameworkilla, joka on yhdistetty robotteihin Azure-palveluiden avulla. Hanketta varten oli tarpeellista kehittää sellainen järjestelmä, jossa laboratorion robotteja pystyisi ohjaamaan ja seuraamaan robottien antureista saatua dataa keskitetysti yhdellä palvelimella.

Opinnäytetyön toisessa luvussa käydään läpi ROS-järjestelmien teoriaa ja dataliikennettä, kolmannessa luvussa opinnäytetyössä hyödynnettyjä työkaluja ja tekniikoita, neljännessä luvussa aloitetaan itse ROS-palvelimen toteutus. Pohdintaosiossa käydään läpi opinnäytetyön tuloksia, johtopäätöksiä, jatkokehittämisaiheita sekä opinnäytetyön tekijän oman oppisen pohdintaa opinnäytetyön aikana.

Tässä opinnäytetyössä tarkastellaan uuden ratkaisun kehittämistä ja sen hyötyjä verrattuna perinteiseen menetelmään. Lisäksi opinnäytetyön aikana arvioidaan uuden järjestelmän tehokkuutta ja käytettävyyttä.

2 ROS -KÄYTTÖJÄRJESTELMÄ

ROS eli Robot Operating System ei ole nimestään huolimatta käyttöjärjestelmä, vaan ohjelmistokehityspaketti eli Software Development Kit (lyh. SDK.) ROS:in avulla käyttäjä voi luoda erinäisistä ohjelmiston osista toimivan robottisovelluksen. (Open Robotics 2022a.)

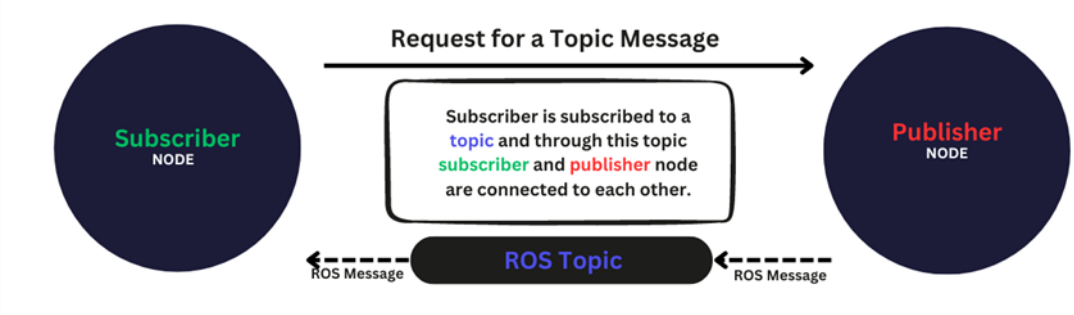
ROS sai alkunsa yhteistyössä valmistuneiden opiskelijoiden, tutkijoiden ja yliopiston professorien kanssa, kun International Conference of Robotics in Automation -tapahtumassa (ICRA) vuonna 2009 julkaistiin artikkeli ROS: an open-source Robot Operating System. Artikkelissa käsiteltiin hanketta, jossa pyrittiin rakentamaan yhteistä koodipohjaa robotiikan tutkimuksia varten alan innovaationopeuden lisäämiseksi. (Open Robotics 2022b.) Vuosien kuluessa ROS:ista on julkaistu useita versioita ja ROS-kehittäjille on syntynyt oma yhteisö. Vuodesta 2012 lähtien on järjestetty vuosittain ROSCon-tapahtumia ympäri maailman, jossa ROS-kehittäjät pääsevät esittelemään uusimpia innovaatioitaan ja saavutuksiaan.

ROS:sia käytetään yleisimmin Linux-käyttöjärjestelmissä, mutta ROS:sin kehityksen myötä myös Windows- ja MacOS-järjestelmät tukevat ROS:in käyttöä. ROS-järjestelmät tukevat useita ohjelmointikieliä, kuten C++, Python, Java, Matlab ja Octave.

ROS-järjestelmien dataliikenne koostuu pääasiassa nodeista, publishereista, subscribereista sekä topiceista. Node eli solmu on itsenäinen ohjelma tai prosessi ROS-järjestelmässä, joka suorittaa sille koodissa annettuja tehtäviä sekä kommunikoi muiden solmujen kanssa viestien (*messages*) ja palveluiden (*services*) kautta. Lisäksi nodet voivat toimia palveluntarjoajina (*service server*) tai käyttäjinä (*service client*), ja ne voivat myös käsitellä pitkään kestäviä laskennallisia tehtäviä toimimalla action clientina tai palvelimena. Nodet voivat myös tarjota konfiguroitavia parametreja, joiden avulla niiden toimintaa voidaan muuttaa ajon aikana. (Open Robotics 2024c.)

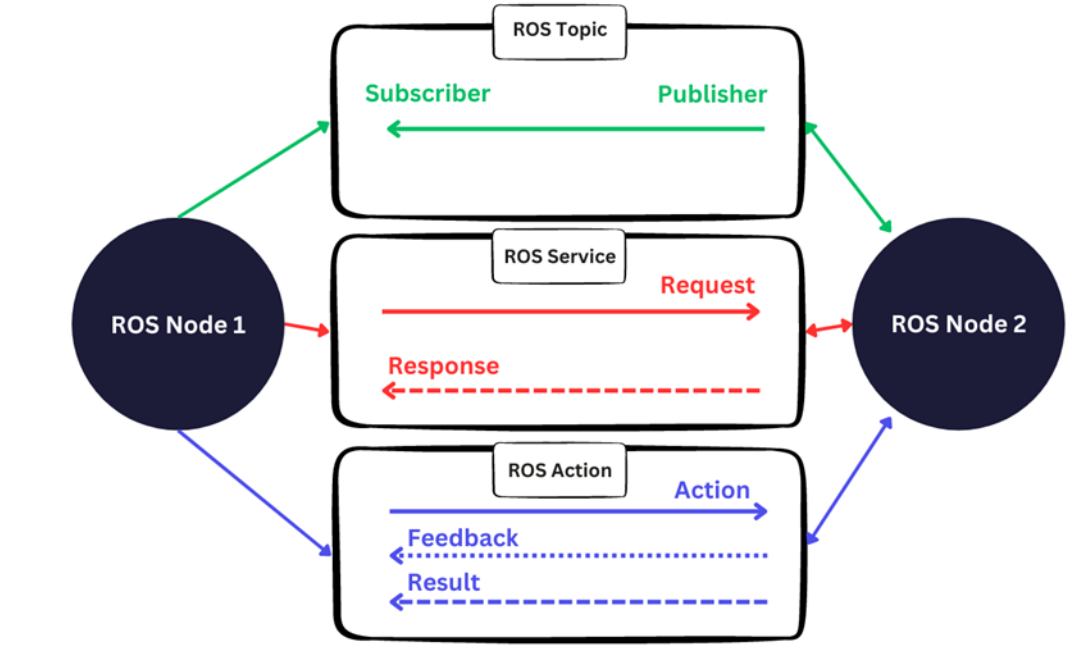
Yksinkertaisin dataliikenne ROS-järjestelmässä on ROS publisherin sekä subscriberin välinen datanvälitys viestin avulla. Seuraavassa kuviossa 1 esitellään

ROS-järjestelmän yksinkertaisinta dataliikennettä käyttäen hyväksi topicia. Kuviossa nähdään ROS publisher-node eli ROS -julkaisijasolmu, mikä julkaisee keräämänsä dataa topiciin, jossa se on saatavilla tilaajasolmua varten. ROS topic on nimensä mukaisesti aihe, joka pitää sisällään julkaisijasolmusta saatua dataa, esimerkiksi sensoridataa. Yhdellä julkaisijalla voi olla useita tilaajia aiheen kautta. (Open Robotics 2024d.)



Kuvio 1. ROS-järjestelmän yksinkertaisin dataliikenne

Viestien lähettämisestä hieman haastavammat ja monimutkaisemmat kommunikointimuodot ROS-järjestelmissä ovat palvelut (*services*) ja toiminnot (*actions*). Käytännössä ROS-palvelun avulla on mahdollista luoda yksinkertainen synkroninen asiakkaan ja palvelimen kommunikaatioviestintä solmujen välille. (Open Robotics 2024e.) Tämä mahdollistaa toiminnot, kuten tietyn toiminnon pyytämisen robotilta tai robotin asetusten muuttamisen. Toiminnot ovat tästä hieman edistyneempiä, sillä toimintojen avulla on mahdollista perua pyyntö esimerkiksi liikuttaa robottia paikasta toiseen. ROS-toiminnot perustuvat aiheisiin ja niiden kautta tiedon ja komentojen lähettämiseen. (Open Robotics 2024f.) Seuraavassa kuviossa 2 kuvataan pelkistetysti ROS-kommunikointitavat, missä action on kaikista kommunikointitavoista kaikista monimutkaisin.



Kuvio 2. ROS aihe, palvelu ja toiminto

3 HYÖDYNNETYT TYÖKALUT JA TEKNIIKAT

3.1 HTML

HTML on lyhenne sanoista Hyper Text Markup Language. Se on standardoitu merkintäkieli WWW-sivujen luomiseen. HTML:n avulla kuvataan web-sivuston rakennetta kertoen selaimelle, miten sisältö näytetään käyttäjälle. (W3Schools 2024a.) Vuonna 1990 Tim Berners-Leen kehitti ensimmäisen version Hyper Text Markup Languagesta "WorldWideWeb" (WWW) -selainohjelman yhteydessä. (W3C 2024.) HTML:n varhaiset versiot olivat yksinkertaisia ja sisälsivät peruselementtejä, kuten otsikot, kappaleet ja linkit, jotka mahdollistivat hyperlinkkien luomisen eri verkkosivujen välille. Sittemmin HTML on kehittynyt merkittävästi, ja uusia versioita on julkaistu lisäämällä uusia toimintoja, elementtejä ja ominaisuuksia, jotka ovat muokanneet web-kehityksen maisemaa ja parantaneet käyttäjäkokemusta verkkosivuilla. Seuraavassa kuviossa 3 nähdään esimerkki HTML-koodista, ja siitä, miltä se näyttää käytännössä nettisivustolla hyödyntäen Visual Studio Coden Live Serveriä.

Kuvio 3. HTML-esimerkki

Tiedosto näyttää sivustolla otsikon ja tekstin. Dokumentti on määritelty HTML-dokumentiksi tiedoston ensimmäisen rivin tagilla. Tämä myös kertoo selaimelle, että kyseessä on HTML5-tiedosto. html-tagien sisälle määritellään kaikki HTML-dokumentin sisältö. HTML-dokumentin body-osioon kirjoitetaan verkkosivun varsinainen sisältö eli tässä tapauksessa otsikko ja teksti, joka näytetään käyttäjälle selaimessa.

3.2 CSS

CSS (Cascading Style Sheets) on tyylilomakekieli, jota käytetään määrittelemään HTML- tai XML-dokumenttien ulkoasu ja esitystapa. CSS mahdollistaa sivuston ulkoasun muokkaamisen ja elementtien visuaalisen esittämisen eri mediatyypeissä, kuten näytöllä, paperilla, puheessa tai muilla laitteilla. (Mozilla Corporation 2024a.)

Yhdistämällä CSS-tyyli ja HTML-tiedosto toisiinsa on mahdollista luoda silmää miellyttäviä tyyliteltyjä kokonaisuuksia. Seuraavassa kuviossa 4 on esitettynä aiemmin esitettyyn HTML-tiedostoon yhdistettynä oma tyylitiedosto styles.css. CSS-tiedoston ja HTML-tiedoston yhdistäminen onnistuu määrittämällä HTML-tiedostoon tyylitiedosto ja se paikka, missä tämä tiedosto sijaitsee.

Kuvio 4. CSS ja HTML yhdistettynä

CSS-tiedosto muuttaa HTML-tiedoston body-osion fontin ja värin. Se myös määrittelee otsikon väriksi valkoisen ja tekstin koon 18 pikseliin ja värin mustaksi.

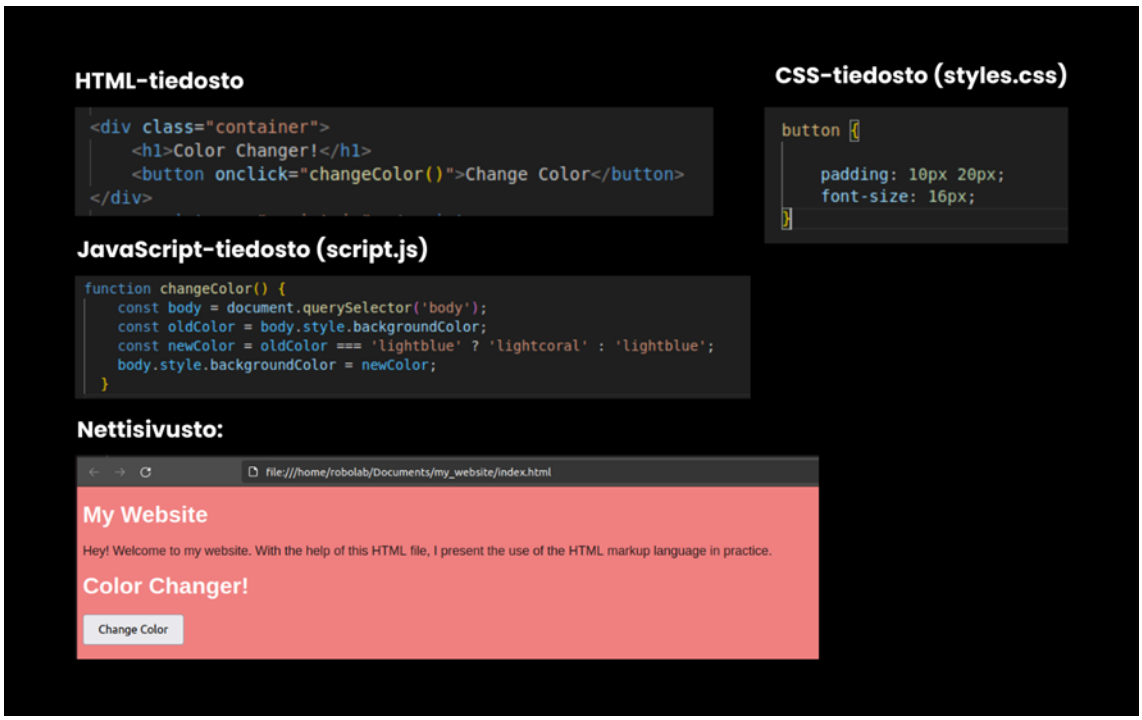
3.3 JavaScript

JavaScriptin alkuperäinen tarkoitus oli tehdä verkkosivuista interaktiivisempia ja elävämpiä. Tämä tarkoitti sitä, että sivujen elementit voivat reagoida käyttäjän toimintoihin, kuten napin painalluksiin, lomakkeiden täyttämiseen ja muihin tapahtumiin, ilman tarvetta lähettää pyyntöjä palvelimelle. Tämä tekee JavaScriptistä erittäin tehokkaan ja joustavan kielen verkkosivujen kehityksessä. (Mozilla Corporation 2024b.)

JavaScript-ohjelmia kutsutaan skripteiksi, ja ne voidaan kirjoittaa suoraan HTML-dokumenttiin <script>-elementin sisään tai erillisinä tiedostoina, jotka liitetään HTML-dokumenttiin. Skriptit suoritetaan automaattisesti sivun latautuessa tai tietyn tapahtuman laukaisemana, kuten käyttäjän toimiessa. (Kantor, I. 2022.)

JavaScript sai alkunsa vuonna 1995 Brendan Eichin aloitteesta. Tätä ohjelmointikieltä kehitettiin eteenpäin Netscape 2:sta varten, ja siitä tehtiin ECMA-262-standardi vuonna 1997 (W3Schools 2024b.)

Seuraavaan kuvioon 5 on luotu HTML-koodin avulla uuden nappi, jota painaessa JavaScriptin avulla luodaan toiminto, jossa sivusto vaihtaa taustaväriä yhteen kolmesta vaihtoehdosta napinpainalluksella. HTML-koodin puolella sivustolle lisätään nappi, CSS-koodissa muutetaan napin ulkoista olemusta ja JavaScript-koodissa annetaan napille sen toiminnallisuus eli värinvaihto.



Kuvio 5. HTML, CSS ja JavaScriptin luoma kokonaisuus

3.4 Azure-palvelut

Azure on Microsoftin kehittämä pilvipalvelualusta, joka tarjoaa laajan valikoiman erilaisia palveluita, kuten laskenta-, analytiikka-, tiedon tallennus sekä verkostoituminen. (Microsoft 2024d.) Azure mahdollistaa sen, että käyttäjän on mahdollista luoda haluamillaan työkaluilla ja rakenteilla haluamansa kokonaisuuden. Azurea käytetään eri toimialoilla ja organisaatioissa muun muassa sovellusten kehittämiseen, testaamiseen ja ylläpitoon sekä datan tallentamiseen ja analysointiin. Pilvipalvelut mahdollistavat joustavan ja skaalautuvan infrastruktuurin, jolloin yritykset voivat mukauttaa resurssien käyttöä tarpeidensa mukaan.

Keskeisimmät palvelut opinnäytetyön kannalta ovat Azure Iot Hub, Azure Stream Analytics, Azure Storage Account sekä Azure Event Hub. Azure Iot Hub on pilvipalvelussa tarjolla oleva palvelu, joka toimii viestintäkeskuksena IoT-sovelluksen sekä siihen liitettyjen laitteiden välillä. (Microsoft 2024a.) Azure Stream Analytics on Azuren tarjoama reaaliaikaisen tiedonkäsittelyn palvelu, joka mahdollistaa suurten tietovirtojen analysoinnin sekä käsittelyn lyhyellä alle millisekunnin viiveellä. (Microsoft 2024b.) Azuren Storage Account on tallennustili, minkä sisällä on kaikki käyttäjän luomat Azure Storage tieto-objektit, kuten taulukot, tiedostot,

jonot, kuvat, videot ja blobit. Azure Event Hub on Azuren tarjoama tiedonsiirtopalvelu, mikä mahdollistaa miljoonien tapahtumien siirtämisen sekunnissa lyhyellä viiveellä mistä tahansa lähteestä mihin tahansa kohteeseen. (Microsoft 2024c.) Yhdistämällä nämä neljä palvelua on mahdollista toteuttaa kaksisuuntainen viestintä IoT-laitteen (robotti), pilvipalvelun sekä nettisivuston välillä, ja varastoida saatua dataa.

3.5 Webpack

Webpack on static module bundler eli staattinen moduulien niputtaja, jota käytetään erityisesti JavaScript-sovelluksissa. (Webpack 2024.) Web-sivuston luomiseen tarvitaan yleensä kolmea eri tiedostomuotoa, jotka ovat JavaScript, HTML sekä CSS-tiedostot. Näiden kolmen avulla on mahdollista luoda toimiva kokonaisuus, joka näyttää web-sivustolla tekstiä, kuvia, värejä sekä toiminnallisuuksia, kuten nappeja. Jos projektissa on useampia näitä tiedostoja, voi nettisivuston latausaika pidentyä huomattavasti. Tämän vuoksi Webpackin käyttö on hyödyllistä, sillä sen avulla jokainen projektissa oleva tiedosto voidaan niputtaa yhdeksi yhtenäiseksi tiedostoksi, mikä varmistaa tiedostojen oikeanlaisen linkityksen sekä optimoinnin ja vähentää sivuston latausaikaa.

Webpackin avulla voi myös hyödyntää loadereita sekä plugineita, jotka myös osaltaan helpottavat projektin kehittämisessä. Loadereita käytetään erilaisten tiedostotyyppien muuntamiseen ohjelmalle ymmärrettävään muotoon. Webpackin omien pluginien käyttö mahdollistaa monia lisätoimintoja sekä optimointeja projektin build-prosessin aikana. Näitä ovat muun muassa sellaiset pluginit, jotka poistavat ylimääräiset välilyönnit ja kommentit koodista. Näin varmistetaan se, että projekti on mahdollisimman pienikokoinen, eikä ylimääräiset tiedostot tai ohjelmat vaikuta esimerkiksi sivuston latausnopeuteen.

3.6 Vue.js

Vue on JavaScript-kehys käyttöliittymien rakentamiseen. Se hyödyntää tavallista HTML:ää, CSS:ää ja JavaScriptiä ja tarjoaa deklaratiivisen, komponenttipohjaisen ohjelmointimallin, jonka avulla voit kehittää tehokkaasti minkälaisia monimutkaisia käyttöliittymiä tahansa. (Vue.js 2024a.)

JavaScript-kehykset eli frameworkit ovat ohjelmistokehitystyökaluja, jotka mahdollistavat valmiin rakenteen ja yleisiä toiminnallisuuksia JavaScript-koodin kirjoittamiseen ja web-sovellusten kehittämiseen. Kehyksien eli frameworkkien avulla kehittäjien on helpompaa luoda ja hallita sovelluksia tarjoamalla valmiita ratkaisua moniin ohjelmointia vaativiin tehtäviin, kuten tilanhallintaan, tietojen käsittelyyn sekä käyttöliittymien luomiseen. (Mozilla Corporation 2024c.) Muita suosittuja JavaScript-kehyksiä ovat muun muassa Facebookin kehittämä React sekä Googlen kehittämä Angular.

Vue on kehitetty joustavaksi ja asteittain omaksuttavaksi työkaluksi. Käyttötarkoituksesta riippuen Vue-kehystä voidaan hyödyntää esimerkiksi parantamaan staattista HTML:ää ilman rakennusvaihetta, upottamaan web-komponentteina mille tahansa verkkosivulle, rakentamaan yksisivuinen SPA, toteuttamaan fullstack-projekteja tai palvelinpuolen renderöintejä (SSR), luomaan Jamstack- tai staattisia sivuja sekä kohdistamaan työpöytäsovelluksiin, mobiililaitteisiin, WebGL:ään tai jopa päätelaitteisiin. (Vue.js 2024b.)

Vue on erityisen suosittu sen SPA-sovellusten, eli yksisivuisten sovellusten rakentamisessa, joissa koko sovellus ladataan kerralla ja sen jälkeen sivuston eri osia päivitetään dynaamisesti. SPA on lyhenne sanoista Single Page Application. Sovellus rakennetaan itsenäisistä, uudelleen käytettävistä komponenteista, mitä kutsutaan nimellä SFC-komponentti eli Single-File Component. Näiden komponenttien sisään määritellään <template>-osioon HTML-koodi, <script>-osioon komponentin logiikka eli JavaScript ja sovelluksen tyyli CSS <style>-osioon. (Vue.js 2024c.) Seuraavassa kuviossa 6 voidaan nähdä yksinkertainen malli SFC-komponentista.

src > views > Hello.vue > ...

```vue
<template>
  <p>{{ greeting }} World!</p>
</template>

<script>
  module.exports = {
    data: function() {
      return {
        greeting: "Hello"
      };
    }
  };
</script>

<style scoped>
  p {
    font-size: 2em;
    text-align: center;
  }
</style>
```

Kuvio 6. SFC-komponentti

Hello.vue-tiedoston <template>-osioon on määritelty kappale, jossa lukee “Hello World!”, mutta sana “Hello” haetaan luodusta data-osasta. Käyttämällä Vuen mustache-syntaksia upotetaan muuttujan arvo eli “Hello” HTML:ään. Näin selain näyttää käyttäjälle tekstin “Hello World!”. JavaScript-koodissa on määritelty data-metodi, joka palauttaa objektin, jossa on yksi avain “greeting”, jonka arvo on “Hello”. Komponentin tyylittely suoritetaan <style>-osiossa, jossa ‘scoped’-määritelmällä määritetään, että tämä tyyli koskee vain sitä komponenttia, eli tässä tapauksessa Vue-komponenttia Hello.vue.

Kuviossa 6 esitetyn esimerkin voi myös luoda tavalliseen HTML-tiedostoon ilman, että tarvitsee asentaa Node.js:ää tai npm:ää, mikä vähentää alkuvaiheen konfigurointia ja asennuksia. Luodaan tavallinen projekti ja lisätään sinne index.html-tiedosto ja kirjoitetaan sinne ohjelma. Seuraavassa kuviossa 7 nähdään index.html:n sisältö.

```
index.html > ...
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Vue Example</title>
  <script src="https://cdn.jsdelivr.net/npm/vue@2.6.14/dist/vue.js"></script>
</head>
<body>
  <div id="app">
    <h1>{{ greeting }} World!</h1>
  </div>

  <script>
    new Vue({
      el: '#app',
      data: {
        greeting: 'Hello'
      }
    });
  </script>
</body>
</html>
```

Kuvio 7. Vue ja HTML

Käytännössä ohjelman sisältö on sama kuin kuviossa 6 esitetty esimerkki. Ero on kuitenkin siinä, että tässä esimerkissä ei ole asennettu Vueen liittyviä paketteja tai käytetä Vue-komponenttia.

3.7 GitHub ja Git

Github on pilvipohjainen alusta, jota käytetään koodin jakamiseen, versionhallintaan sekä yhteistyön parantamiseen kehittäjien kesken. Tallentamalla koodia GitHubin arkistoon on muiden kehittäjien tai projektin yhteistyökumppanien mahdollista tarkastella kirjoitettua koodia, antaa parannusehdotuksia sekä työskennellä samaan aikaan saman projektin kanssa ilman ristiriitaisuuksia. (GitHub, Inc. 2024a.)

GitHubia käytetään usein yhdessä Git-nimisen versionhallintajärjestelmän kanssa, joka seuraa tiedostoihin tehtyjä muutoksia. Järjestelmä on siitä hyödyllinen, että se mahdollistaa eri yksilöiden pystyvän tekemään muutoksia samoihin tiedostoihin samanaikaisesti. (GitHub, Inc. 2024b.) Git mahdollistaa koodin his-

torian seuraamisen, versioiden hallinnan ja palautuksen aikaisempiin tiloihin tarvittaessa. Yhdessä GitHub sekä Git luovat versionhallinnalle otollisen kokonaisuuden, jota useat kehittäjät käyttävät hyödykseen.

3.8 Canva

Canva on verkkopohjainen suunnittelu- ja visuaalisen viestinnän alusta, joka perustettiin vuonna 2013. Sen tavoitteena on tarjota kaikille maailman ihmisille mahdollisuus luoda ja julkaista visuaalisia materiaaleja monenlaisista aiheista missä tahansa. (Canva 2024a.)

Canva on sunniteltu niin, että käyttäjät pystyvät luomaan ammattimaisia suunnitelmia ilman graafisen suunnittelun osaamista. Alusta toimii käytännössä drag-and-drop -käyttöliittymällä, joka mahdollistaa helppokäyttöisyyden ja lukemattomat mahdollisuudet suunnittelun jokaiselle käyttäjälle. Käyttäjän on mahdollista käyttää Canvan omia kuvia, malleja ja videoita, tai ladata itse omalta laitteelta niitä Canvan käyttöliittymään. (Canva 2024b.)

3.9 Visual Studio Code

Visual Studio Code on kevyeksi kehitetty lähdekoodieditori, joka on saatavilla työpöytäympäristössä Windowsille, macOS:ille ja Linuxille. Se tukee natiivisti JavaScriptiä, TypeScriptiä ja Node.js:ää, ja siihen on saatavilla laaja valikoima laajennuksia, jotka tukevat muita ohjelmointikieliä ja ympäristöjä, kuten C++, C#, Java, Python, PHP, Go ja .NET. (Visual Studio Code 2024a.)

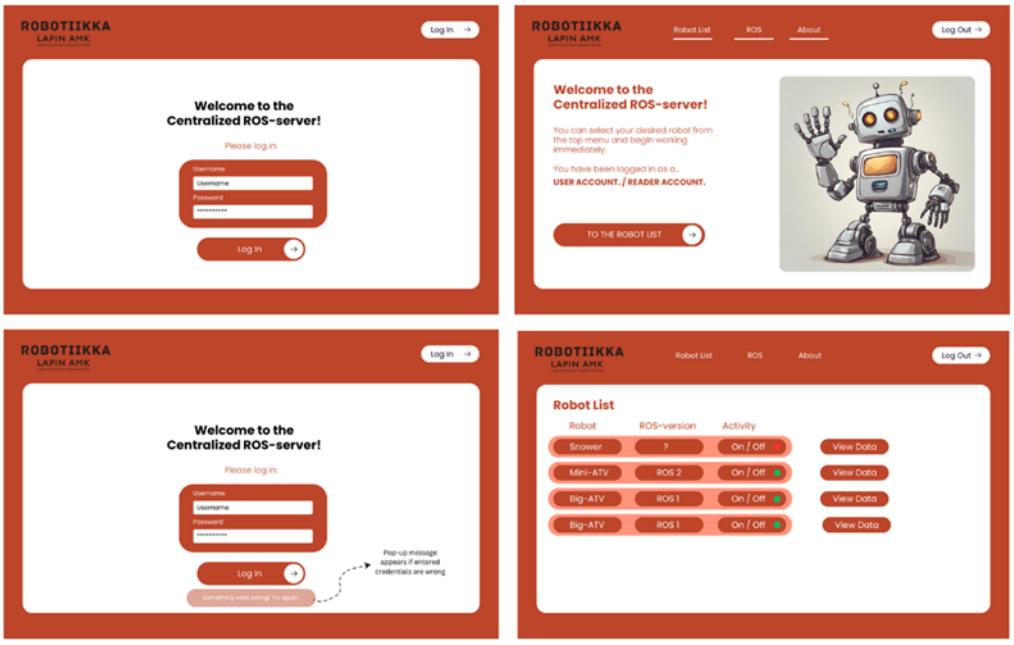
Visual Studio Coden keskeisimpiä käyttötarkoituksia ovat koodin kirjoittaminen ja muokkaaminen, koodissa olevien virheiden etsiminen ja korjaaminen, versionhallinta yhdessä versionhallintajärjestelmien, kuten Gitin kanssa, projektinhallinta sekä web-kehitys. Koodieditori mahdollistaa myös reaaliaikaisen koodin jakamisen ja yhteistyön muiden kehittäjien kanssa etänä hyödyntäen ohjelmiston työkaluja. (Visual Studio Code 2024b.)

4 KESKITETYN ROS-PALVELIMEN TOTEUTUS

4.1 Sivuston suunnittelu

Verkkosivustolle ei ollut asetettu graafisia vaatimuksia, mutta päätettiin suunnitella etusivu, robottien listaamiseen tarkoitettu sivu, ROS-tietosivu sekä yleinen informaatiosivu keskitetystä ROS-palvelimesta Canva-alustalla. Asiakkaalla oli graafisen olemuksen sijasta paljon toiveita sivuston toiminnallisista elementeistä: palvelimen tulisi näyttää roboteilta kerättyä dataa topicien kautta ja käyttäjän olisi mahdollista lähettää robotille komentoja sivuston kautta. Käyttäjän on aina mahdollista tarkastella saatua dataa, mutta ennen komentojen lähettämistä sivusto kysyisi käyttäjältä käyttäjätunnusta sekä salasanaa robottien turvallisuuden takaamiseksi. Asiakkaan toiveena oli, että palvelin toteutetaan ROS2-pohjalle ja että palvelimen on mahdollista kerätä dataa myös roboteilta, jotka ovat ROS1-version robotteja. Datan olisi tarkoitus virrata palvelimelle heti, kun robotti käynnistetään.

Suunnitteluvaiheessa ensimmäisenä huomioitiin sivuston värit. Keskitetty ROS-palvelin toteutettiin AI.R (Artificial Intelligence and Robotics) -hankkeelle, joka oli tiukasti yhteistyössä Lapin ammattikorkeakoulun kanssa. Lapin ammattikorkeakoululle ominaisia värejä, kuten tumman oranssia ja kermanvalkoista päätettiin käyttää. Kuten seuraavassa kuviossa 8 näkyy, ulkonäkö on suunniteltu kirjautumissivulle, robottinäkymälle, etusivulle sekä navigointipalkin ulkoasulle hyödyntäen Canvan graafisia työkaluja.



Kuvio 8. Sivuston suunnittelu

4.2 Oracle VM VirtualBox ja Ubuntu 22.04.4 LTS -asennus

Pöytätietokone, jossa on Windows 10-käyttöjärjestelmä, otettiin käyttöön ja sen tuli toimia laboratorion roboteille keskitettynä ROS-palvelimena. Projektin aloittamisen ensimmäinen askel oli ympäristön luominen ROS-järjestelmän käyttöä varten.

Linux on todettu kehitykselle ja ROS-luotettavaksi ympäristöksi, joten päätettiin käyttää sitä. Oracle VM VirtualBox versiolla 7.0 asennettiin tietokoneelle. VirtualBox ladattiin sen virallisilta verkkosivuilta. VirtualBox tarvitsi käyttöjärjestelmän toimiakseen, joten Ubuntu 22.04.4 LTS -versio asennettiin sen sivuilta ja lisättiin ISO-kuvana VirtualBoxiin. Virtuaalikone käynnistettiin ja käyttäjätunnukset lisättiin Ubuntuun.

4.3 ROS 2 Humble Hawksbillin asennus

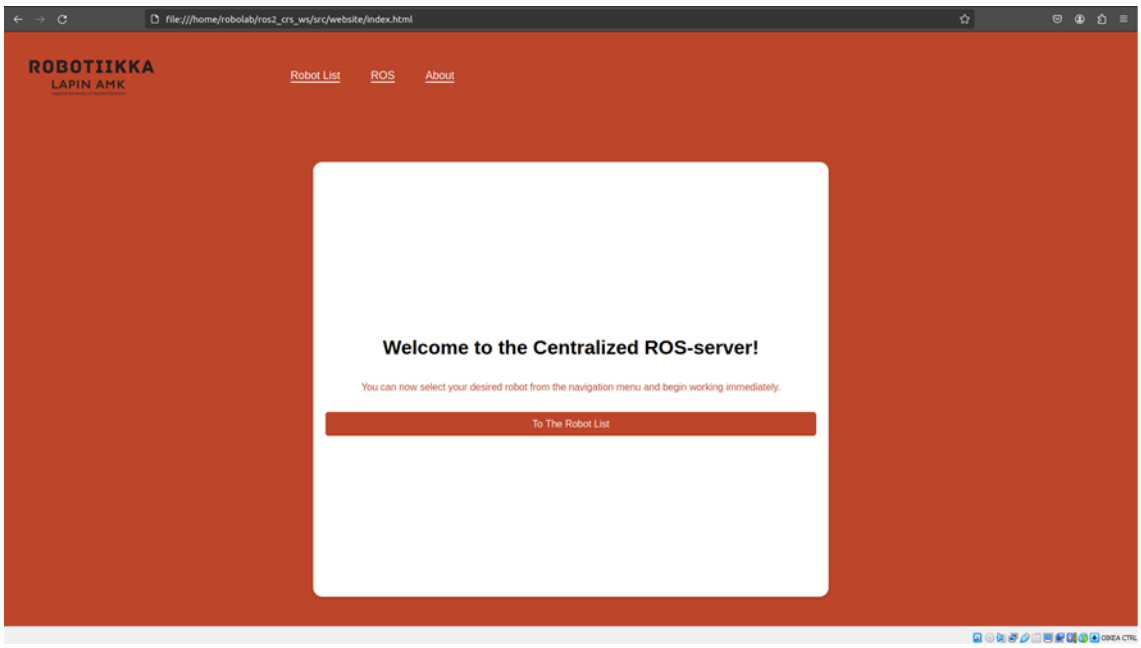
Pohdittaessa, mitä versiota tulisi käyttää keskitettynä ROS-palvelimena, päädyttiin käyttämään ROS2 toiseksi uusinta versiota, Humble Hawksbillia, joka on suositeltu ROS2-pohjaisille järjestelmille. Muut nykyiset ylläpidon alla olevat jakelut,

kuten Noetic Ninjemys ja Iron Irwin eivät soveltuneet keskitetyn ROS-palvelimen tarkoituksiin. Asiakkaan toiveena oli, että keskitetty palvelin olisi ROS2-pohjalla, ja jakeluista ainoastaan Humble Hawksbill sekä Iron Irwin olivat ROS2-pohjalle sopivia. Noetic Ninjemys on tarkoitettu ROS1-pohjaisille laitteille ja roboteille. Syynä Humble Hawksbillin valitsemiseen Iron Irwinin sijasta oli se, että Iron Irwin on edelleen kehitysvaiheessa, mikä voi aiheuttaa yllättäviä haasteita sen toiminnassa.

ROS:n asennus suoritetaan Ubuntun Terminalissa syöttämällä sinne erilaisia komentoja. Näitä ovat muun muassa komennot Ubuntu Universe-kirjaston olemassaolon tarkistukseen ja asennukseen, ROS2 GPG-avaimen lisäämiseen ja kirjastojen lisäämiseen source-listaan. Kun nämä on tehty, päivitetään apt-kirjastojen välimuisti komennolla *sudo apt update* sekä päivittää järjestelmä kaiken varalta *sudo apt upgrade*-komennolla ennen kuin lisäämme uusia paketteja. Itse ROS:in asentaminen onnistuu komennolla *sudo apt install ros-humble-desktop*. Tämän lisäksi lisättiin ROS:ia varten ROS kehitystyökalut komennolla *sudo apt install ros-dev-tools*. Lopuksi määritetään kehitysympäristö hakemalla tiedosto *source /opt/ros/humble/setup.bash*.

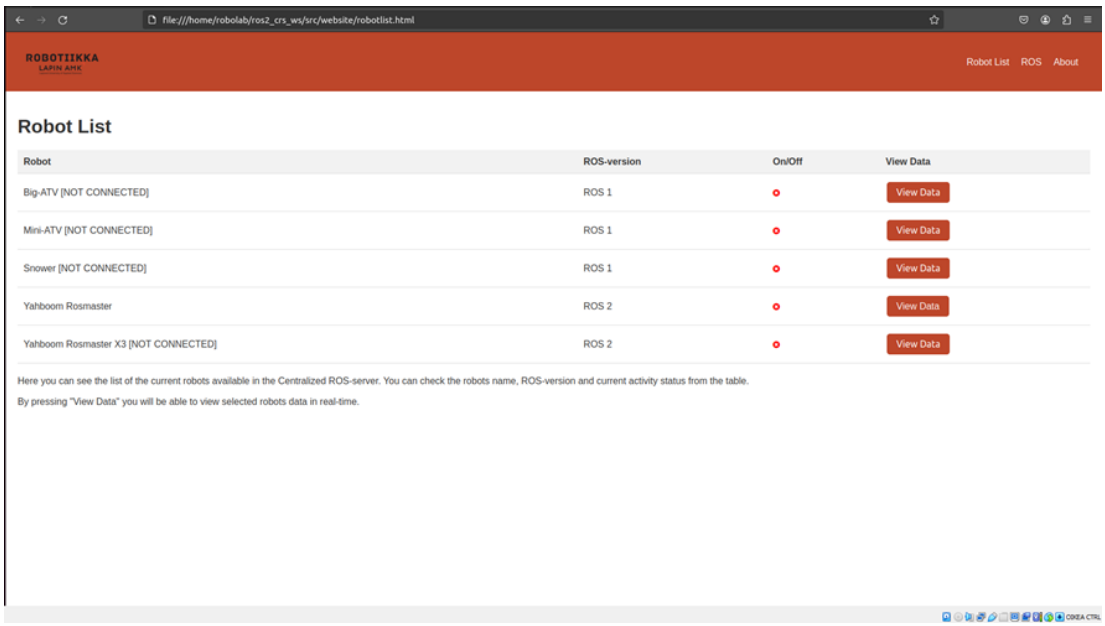
4.4 Nettisivuston kehitys

ROS-palvelimen on tarkoitus avautua nettisivustonäkymään, johon haettava data tulostetaan ja käyttäjän on mahdollista antaa robotille komentoja kirjautuessaan sisään ennalta määritellyille käyttäjätunnuksille. Yhdistelemällä HTML, CSS- ja JavaScript-tiedostoja oli mahdollista kehittää sivusto alustavasti näyttämään suunnitelman mukaiselta. Seuraavassa kuviossa 9 on esitettynä demonstraatio etusivun rakenteesta ja ulkonäöstä, tuotettuna CSS ja HTML-koodin avulla.



Kuvio 9. Nettisivuston etusivu

Nettisivun rakenne on yksinkertainen: etusivulta pääsee valitsemaan navigointivalikosta robottilistan, ROS-infosivuston sekä keskitetyn ROS-palvelimen infosivuston väliltä. Painamalla “To The Robot List” -näppäintä käyttäjä pääsee seuraavaan näkymään, jossa käyttäjän on mahdollista valita haluamansa robotti. Seuraavasta kuviosta 10 näkee, että jokaisen robotin kohdalla on oma “View Data” -nappi, minkä avulla käyttäjä pääsee seuraamaan käynnissä olevan robotin dataliikennettä reaaliajassa. “View Data”-painiketta painamalla avautuu sivustonäkymä, mikä on lähes jokaisella robotilla samalla tyylillä toteutettu.



Kuvio 10. Robottilistanäkymä

Käytössä on vain kolme eri tiedostotyyppiä (JavaScript, HTML, CSS) erilaisia tiedostoja, jotka luovat ulkoasun, toiminnallisuuden sekä rakenteen web-sivustolle. Ongelma on se, että Azureen tehtävä yhteys tarvitsee toimiakseen Azuren omia moduuleja ja riippuvuuksia, joita selain ei sellaisenaan ymmärrä. Jotta tämä ongelma korjaantuisi, olisi käytettävä jotain moduulien niputtajaa, tässä tapauksessa Webpackia yhtenäisen kokonaisuuden luomiseksi. Wepback mahdollistaa sen, että sivusto pystyy käyttämään moduuleja selaimessa ja näin kyetään demonstroimaan yhteyttä sivustolta Azureen.

Aluksi on luotava uusi hakemisto projektille ja ladattava Node.js sekä npm luodun hakemiston sijainnissa. Projekti alustetaan npm-komennolla, joka luo projektiin package.json-tiedoston, johon tallennetaan kaikki projektin riippuvuudet. Tämän jälkeen asennetaan Webpack ja lisätään Webpack-cli yhdeksi projektin riippuvuuksista. Tämä lisää package.json -tiedostoon riippuvuudeksi edellä mainitun paketin. Projektiin luodaan myös webpack.config.js-tiedosto, joka määrittelee, miten Webpack käsittelee projektiin sisältyvät tiedostot. Kaikki HTML-tiedostot laitetaan kansioon nimeltä templates, ja niiden JavaScirpt-koodit siirretään erillisiin JavaScript-tiedostoihin.

Jokaiselle tiedostolle on luotu oma bundle-tiedosto dist-kansioon. Näihin bundle-tiedostoihin pakataan jokainen tiedosto, jotta niitä voidaan sitten käyttää helposti

selaimessa. Webpack-konfiguraatioon voi lisätä erilaisia lataajia ja lisäosia, jotka auttavat lataamaan ja käsittelemään annettuja resursseja. Tämä konfiguraatio tapahtuu webpack.config.js-tiedostossa. Seuraavassa kuviossa 11 on tämän Webpack-projektin konfiguraatiotiedosto. Projekti käyttää plugineita eri tyyppisten tiedostojen käsittelyyn, automaattiseen html-tiedostojen generointiin, CSS-erotteluun ja kuvien kopiointiin. Jokaiselle pluginille on määritelty myös “rules” eli säännöt, joiden mukaan ne toimivat.

```
module.exports = {
  mode: 'production',
  entry: {
    index: './src/index.js',
    aboutcrs: './src/aboutcrs.js',
    robotlist: './src/robotlist.js',
    yahboomrm: './src/yahboomrm.js'
  },
  output: {
    filename: '[name].bundle.js',
    path: path.resolve(__dirname, 'dist'),
    clean: true // Puhdistaa output-hakemiston ennen jokaista buildia
  },
  plugins: [
    new HtmlWebpackPlugin({
      template: './src/templates/index.html',
      filename: 'index.html',
      chunks: ['index']
    }),
    new HtmlWebpackPlugin({
      template: './src/templates/aboutcrs.html',
      filename: 'aboutcrs.html',
      chunks: ['aboutcrs']
    }),
    new HtmlWebpackPlugin({
      template: './src/templates/robotlist.html',
      filename: 'robotlist.html',
      chunks: ['robotlist']
    }),
    new HtmlWebpackPlugin({
      template: './src/templates/yahboomrm.html',
      filename: 'yahboomrm.html',
      chunks: ['yahboomrm']
    }),
    new MiniCssExtractPlugin({
      filename: 'styles/styles.css'
    }),
    new CopyPlugin({
      patterns: [
        {
          from: path.resolve(__dirname, 'src', 'images'),
          to: path.resolve(__dirname, 'dist', 'images')
        }
      ]
    })
  ],
```

Kuvio 11. Webpack-projektin konfiguraatio

Webpack auttaa hallitsemaan ja yhdistämään JavaScript-moduuleja, mutta se ei tarjoa käyttöliittymäkehykseen liittyviä ominaisuuksia. Se on usein osa suurempaa työkalupakettia. Webpack vaatii kehittäjältä sen, että kehittäjä rakentaa itse suurimman osan ominaisuuksista tai yhdistää muihin työkaluihin ja kirjastoihin, mikä pidemmän päälle on haastavaa työkalujen ja kirjastojen versioiden yhteensopivuuden kanssa. Konsultointi eräältä organisaation front end -kehittäjältä liittyen käytettäviin kirjastoihin ja viitekehyksiin sekä niiden yksinkertaisimpiin vaihtoehtoihin johti siihen tulokseen, että aloittelijalle sopiva viitekehys on Vue. Vue on hyvä valinta tarkoituksiin, koska se ei käytä Typescriptiä kuten React, vaan tavallista JavaScriptiä. Tämä tarkoittaa sitä, että uuden ohjelmointikielen opettelu ei ole tarpeen Vuen käyttämiseksi.

Vue-kehyksen käytön opettelu aloitettiin tarkastelemalla Vuen dokumentaatiota ja luomalla muutamia harjoitusprojekteja Vue-kehyksellä. Saatua ymmärrystä Vuen SFC-komponenttien rakenteesta ja logiikasta hyödynnettiin aiemmin kirjoitettujen JavaScript-, HTML- ja CSS-tiedostojen muuntamisessa SFC-komponenteiksi. Ennen tätä luotiin Vue-projekti, mikä onnistuu asentamalla npm-komennon avulla Vuen oma kirjasto Vue CLI Windowsin Terminalissa sekä asennuksen jälkeen suorittamalla Vue-kansion luomiseen tarkoitettu komento eli *vue create (kansion nimi)*.

Kansion nimeksi annettiin Vue Centralized Server, lyhennettynä vue-cen-ros. Tämä kansio avattiin Terminalissa *code .*-komennolla, jotta kansio avattaisiin käyttäen Visual Studio Codea. Automaattisesti generoidut ylimääräiset tiedostot, kuten HelloWorld.vue poistettiin uusien komponenttien tieltä. App.vue-tiedosto pyyhitään kokonaan tyhjäksi. Aloitetaan luomalla navigointipalkkikomponentti nimeltään NavBarComponent.vue. Tähän SFC-komponenttiin luodaan samat toiminnallisuudet, jota aiemmin demonstroiduissa HTML-tiedostoissa oli. Alla olevissa kuvioissa 12 ja 13 esitetään navigointipalkin toteutus HTML-kielellä sekä Vuen SFC-komponentissa.

```
<header>
    <div class="logo">
        <img src="images/logo.png" alt="Logo">
    </div>
    <nav>
        <ul>
            <li><a href="index.html">Home</a></li>
            <li><a href="robotlist.html">Robot List</a></li>
            <li><a href="aboutcrs.html">About</a></li>
        </ul>
    </nav>
</header>
```

Kuvio 12. HTML-navigointipalkki

Molemmat kuvioista 11 ja 12 käyttävät samaa logo-kuvaa navigointipalkin logo-elementin luomiseen. Index.html -tiedoston sijasta sivuston etusivu on linkitetty sijaintiin ”/”, jossa projektin FrontPage.vue -komponentti eli etusivukomponentti sijaitse.

```
<nav class="navbar">
  <div class="navbar-logo">
    <router-link to="/">
      <img src="@/assets/logo.png" alt="Logo" />
    </router-link>
  </div>
  <ul class="navbar-menu">
    <li class="navbar-item">
      <router-link to="/">Home</router-link>
    </li>
    <li class="navbar-item">
      <router-link to="/robot-list">Robot List</router-link>
    </li>
    <li class="navbar-item">
      <router-link to="/about">About</router-link>
    </li>
  </ul>
</nav>
```

Kuvio 13. Vue-navigointipalkki

Näistä kahdesta kuviosta voidaan todeta, että nämä kaksi toteutustapaa eivät kovin eroa paljon toisistaan. Vue-komponentissa tulee kuitenkin ilmi <router-link>, jonka avulla navigoidaan web-sivustolla komponentista toiseen. Tämä korvaa HTML-tiedoston href-attribuutin, joka ohjaa käyttäjän viitattuun tiedostoon.

Seuraavassa kuviossa 14 esitetään <script> tagin sisällä oleva määrittely, joka määrittää yksinkertaisen Vue-komponentin nimeltä NavBarComponent. Nimeämällä komponentti on mahdollista käyttää tätä komponenttia muissa Vue-komponenteissa.

```
<script>
export default {
  name: 'NavBarComponent'
}
</script>
```

Kuvio 14. NavBarComponent-määrittely

App.vue on yksi projektin komponenteista, joka toimii Vue-sovelluksen pääkomponenttina. Tämä komponentti toimii juurikomponenttina, josta sovellus alkaa. Kaikki muut komponentit ja näkymät sisällytetään ja renderöidään tämän pääkomponentin sisällä. Jotta voi sisällyttää luodun NavBarComponent-komponentin pääkomponenttiin, on käytettävä <script> tagin sisällä import-komentoa, joka hakee halutun komponentin määritellystä tiedostosijainnista. Seuraavassa kuviossa 15 nähdään, kun NavBarComponent määritellään App-nimisen komponentin sisällä.

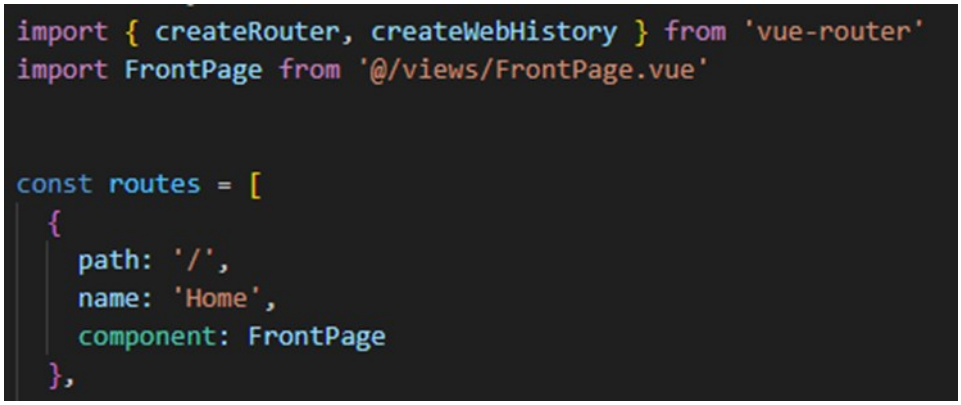
```
<script>
import NavBarComponent from './components/NavBarComponent.vue'

export default {
  name: 'App',
  components: {
    NavBarComponent
  }
}
</script>
```

Kuvio 15. NavBarComponent lisääminen App-komponenttiin

Seuraavassa kuviossa 16 <template> -osion sisälle sisällytetään tuotu NavBarComponent sekä Vue Routerin määrittelemä paikka, johon määritellyiden reittien perusteella renderöidään komponentteja. Käytännössä se toimii paikkamerkkinä, johon nykyinen reitti sisältöineen sijoitetaan.

Kuvio 16. App.vue-komponentin sisältö

Tässä vaiheessa sivustolla ei ole näkyvissä muuta kuin aiemmin luotu navigointipalkki. Jokainen sovellus tarvitsee itselleen etusivun, joten luodaan uusi näkymä (*view*) nimeltä FrontPage.vue. Tähän komponenttiin luodaan etusivu, joka aiemmassa HTML-, CSS- ja JavaScript -versiossa oli jo ennestään. Tämä versio tulee vain muuttaa Vue-projektille sopivaan muotoon. Käytännössä ainoa asia, joka koodissa muuttuu on href-attribuutin tilalle vaihdettava <router-link>, joka ohjaa käyttäjän “/robot-list”-nimiseen sijaintiin. Tämä sijainti on määritelty routerissa routes-nimisessä taulukossa, jossa jokainen reitti on objekti, joka sisältää reitin polun, nimen ja siihen liittyvän komponentin nimen. Frontpage.vue-komponentin reitti on määritelty Home- nimellä ja sen polku on “/”, kuten seuraavassa kuviossa 17 näkyy.



Kuvio 17. Reittien polkujen määrittely

Tehtävänä on seuraavaksi tuoda Vue-projektiin myös RobotList-sivu, johon kaikki yhdistettävät robotit listataan. HTML-tiedostossa RobotList oli toteutettu käyttäen

taulukkoelementtiä. Muita robotteja Yahboom Rosmaster X3:n lisäksi ei ole yhdistetty tähän HTML-tiedostoon. Seuraavassa kuviossa 18 on RobotListin toteutus tuotettuna HTML:n avulla.

```
<div class="content">
    <h1>Robot List</h1>
    <table>
        <thead>
            <tr>
                <th>Robot</th>
                <th>ROS-version</th>
                <th>On/Off</th>
                <th>View Data</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>Yahboom Rosmaster X3</td>
                <td>ROS 2</td>
                <td><input type="checkbox"></td>
                <td><a href="yahboomrm.html" class="view-data-button">View Data</a></td>
            </tr>
            <tr>
                <td>Mini-ATV [NOT CONNECTED]</td>
                <td>ROS 1</td>
                <td><input type="checkbox"></td>
                <td><button class="view-data-button">View Data</button></td>
            </tr>
            <tr>
                <td>Big-ATV [NOT CONNECTED]</td>
                <td>ROS 1</td>
                <td><input type="checkbox"></td>
                <td><button class="view-data-button">View Data</button></td>
            </tr>
            <tr>
                <td>Snower [NOT CONNECTED]</td>
                <td>ROS 1</td>
                <td><input type="checkbox"></td>
                <td><button class="view-data-button">View Data</button></td>
            </tr>
            <tr>
                <td>Yahboom Rosmaster X3 Pro [NOT CONNECTED]</td>
                <td>ROS 2</td>
                <td><input type="checkbox"></td>
                <td><button class="view-data-button">View Data</button></td>
            </tr>
            <!-- Lisää uusi robotti tarvittaessa -->
        </tbody>
    </table>
```

Kuvio 18. RobotList-sivusto toteutettuna HTML-tiedostona

Tämä taulukko on muutettava Vue-komponenttiin sopivaksi. Uusi näkymä nimeltä RobotList.vue luodaan, mihin lisätään taulukko. Taulukon otsikot määritellään <template>-osion sisällä, ja “v-for”-direktiiviä käytetään, jotta komponentti voi renderöidä jokaisen robotin tiedot taulukon riviksi. Nämä tiedot haetaan <script>-tagin sisällä määritellystä data-funktiosta, joka palauttaa objektin sisältäen robottien tiedot. Seuraavassa kuviossa 19 esitetään data-funktion sisältö.

```
<script>
export default {
  name: 'RobotList',
  data() {
    return {
      robots: [
        { name: 'Yahboom Rosmaster X3', rosVersion: 'ROS 2', status: true },
        { name: 'Mini-ATV [NOT CONNECTED]', rosVersion: 'ROS 1', status: false },
        { name: 'Big-ATV [NOT CONNECTED]', rosVersion: 'ROS 1', status: false },
        { name: 'Snower [NOT CONNECTED]', rosVersion: 'ROS 1', status: false },
        { name: 'Yahboom Rosmaster X3 Pro [NOT CONNECTED]', rosVersion: 'ROS 2', status: false }
        // Lisää muita robotteja tarvittaessa
      ]
    }
  }
}
</script>
```

Kuvio 19. RobotListin data-funktion sisältö

Seuraava kuvio 20 esittää, miten taulukko esittäytyy <template> -osion sisällä. Käyttäjä pystyy navigoimaan <router-link>-elementin avulla “View Data”-nappia painamalla robotin tietoihin.

```
<div class="content">
  <h1>Robot List</h1>
  <table>
    <thead>
      <tr>
        <th>Robot</th>
        <th>ROS-version</th>
        <th>On/Off</th>
        <th>View Data</th>
      </tr>
    </thead>
    <tbody>
      <tr v-for="robot in robots" :key="robot.name">
        <td>{{ robot.name }}</td>
        <td>{{ robot.rosVersion }}</td>
        <td><input type="checkbox" :checked="robot.status" disabled></td>
        <td>
          <router-link :to="`/robot/${robot.name.replace(/\s+/g, '-')}`" class="view-data-button">View Data</router-link>
        </td>
      </tr>
    </tbody>
  </table>
```

Kuvio 20. RobotListin template-sisältö

Sivuston komponenteista puuttuu vielä kaksi: About-sivu sekä RobotDetail-sivu. About-sivun rakenne on hyvin yksinkertainen, siinä on vain muutama rivi tekstiä <h1>- sekä <p>-elementtien sisällä. Luodaan About.vue-komponentti ja siirretään teksti aboutcrs.html-tiedostosta tämän komponentin <template>-osioon. Määritellään myös komponentille nimi “AboutComponent”.

RobotDetail-sivua tarvitaan näyttämään robottien tiedot sekä myöhemmin projektin edistyessä lähettämään sekä vastaanottamaan dataa Azure Event Hubista suoraan robotilta. Luodaan RobotDetail.vue -näkymäkomponentti sekä sen lisäksi TextFieldButton- ja ChatBox-komponentit. HTML-toteutuksessa chatbox, nappi sekä tekstikenttä ovat samassa tiedostossa, mutta Vue-projektia rakentaessa komponentteja on helpompi hallita ja järjestellä, jos ne on eroteltu eri komponentteihin.

RobotDetail-komponenttiin sisällytetään tämä kaksi Vue-komponenttia. Seuraavassa kuviossa 21 näytetään, miten komponentit tuodaan tiedostopolusta, ja asetetaan päällekkäin <template>-osioon. TextFieldButton sisältää tekstikentän ja napin, mikä mahdollistaa käyttäjän lähettämään viestejä Azure Event Hubiin. “handleMessageSent”-metodi käsittelee lähetetyt viestit ja ohjaa ne ChatBox-komponenttiin, jotta ne näytetään siellä käyttäjälle indikaattorina lähetetystä viestistä.

```vue
<template>
  <div class="container">
    <h1>{{ robotName }} Robot Data Center</h1>

    <div class="content">
      <p>Welcome to {{ robotName }} Robot Data Center! Here you can monitor {{ robotName }} robot's data in real time and give it commands.</p>
    </div>

    <div id="chat-container">
      <ChatBox ref="chatbox"></ChatBox>
      <TextFieldButton @message-sent="handleMessageSent"></TextFieldButton>
    </div>
  </div>
</template>

<script>
import ChatBox from '@/components/ChatBox.vue';
import TextFieldButton from '@/components/TextFieldButton.vue';

export default {
  name: 'Yahboom-Rosmaster-X3',
  components: {
    ChatBox,
    TextFieldButton
  },
  props: {
    name: {
      type: String,
      required: true
    }
  },
  computed: {
    robotName() {
      return this.name.replace('-', ' ');
    }
  },
  methods: {
    handleMessageSent(message) {
      this.$refs.chatbox.sendMessageToChatbox(message);
    }
  }
}
</script>
```

Kuvio 21. RobotDetail.vue

TextFieldButton sekä ChatBox käyttävät molemmat WebSocketia kyetäkseen kommunikoimaan Azure Event Hubin kanssa. Myöhemmin implementoidaan erillinen JavaScript-tiedosto, joka mahdollistaa yhteyden luomiseen Azure Event Hubiin WebSocketin sekä Azure Event Hubin yhteysmerkkijonon, nimen ja kuluttajaryhmän avulla.

4.5 Yhteyden muodostaminen robotilta Azureen

ROS:iin tutustumisen jälkeen tuli nopeasti selväksi, että monen eri laitteen yhdistäminen yhteen samaan palvelimeen olisi ROS-järjestelmässä haasteellista. Jokaiselle robotille tulisi manuaalisesti lisätä publisher ja subscriber, jotka toimisivat viestin välittäjinä robotin ja keskitetyn ROS-palvelimen välillä. Publisherin avulla robotti lähettäisi keräämänsä datan topiciin, josta ROS-palvelin hakisi datan. Subscriberin avulla robotti saisi topicista datan komennoista, jotka käyttäjä antaa robotille ROS-palvelimen päässä. Tämä tulisi työlääksi robottien suuren määrän ja erilaisten ROS-versioiden vuoksi. Osa roboteista käyttää vielä toistaiseksi ROS1-versiota, kun taas toiset robotit käyttävät uusinta ROS2-versiota.

Lopulta päätettiin, että robotin ja ROS-palvelimen välille tulisi löytää tehokkaampi keino viestintäyhteyden luomiseen. Alan asiantuntijalta saatiin vinkki, että Azure tarjoaa tähän tarkoitukseen sopivan palvelun. Azure on Lapin ammattikorkeakoulun ja Frostbitin vahvasti hyväksymä ja käyttämä palvelu, mikä vaikutti suuresti palvelun valintaan. Muita vaihtoehtoja olivat muun muassa CSC CPouta ja Googlen Firebase. Azurea päädyttiin kuitenkin käyttämään, koska se sopi parhaiten käyttötarkoituksiin: tarvittiin palvelu, johon voitaisiin varastoida ja seurata dataa sekä luoda helppo yhteys robotilta Azuren kautta nettisivustolle.

Firebase ei sopinut tarkoitukseen, koska se on maksullinen palvelu, jonka laskutusta on haastavaa budjetoida. Tämä olisi tarkoittanut sitä, että opinnäytetyön toimeksiantajalla olisi ollut haasteita arvioida Firebasen käytön kustannuksia opinnäytetyön aikana ja jatkokehityksessä. Koska Azure oli jo organisaation käytössä, sen käytöstä ei syntynyt käytännössä lisäkustannuksia. CSC CPouta oli yksi vaihtoehdoista, joka olisi tarjonnut pilvipalvelualustan useilla eri palveluvaihtoehdoilla, kuten Azure. Ainoa ongelma tässä oli, että CPouta tarjosi pienemmän määrän palveluita verrattuna Azureen. CPouta tarjoaa laskentatehoa ja tallennustilaa, kun taas Azure tarjoaa näiden lisäksi useita muita palveluita, kuten analytiikkaa ja IoT.

Videodatan lähettämiseksi robotin kamerasta Azureen luotiin Pythonilla tiedosto, joka olisi yhteydessä Azureen yhteysmerkkijonolla ja lähettäisi kuvadatan Azureen. Yahboom Rosmaster X3-kameralla oli jo oma ohjelmansa, joka vaati vain vähäistä lisäohjelmointia halutun tuloksen saavuttamiseksi. Seuraavassa kuviossa 22 näkyy pieni koodinpätkä, joka yhdistää Azuren ja robotin toisiinsa. Azuren yhteysmerkkijonoa hyödyntämällä luotiin MSG-niminen muuttuja, joka sisältää lähetettävän viestin ID:n sekä kuvan eli imagen. Tämä MSG-muuttuja lähetetään Azureen.

```python
# Connection String
CONNECTION_STRING = 
MSG = '{{"messageId": {}, "Image": "{}"}}'

# Initialize the camera object
camera = Rosmaster_Camera(debug=True)

# Functions to create an IoT Hub client and send a message to Azure
def iothub_client_init():
    client = IoTHubDeviceClient.create_from_connection_string(CONNECTION_STRING)
    return client

def send_image_to_azure(client, image_id, image):
    try:
        # Convert image to base64 format
        _, encoded_image = cv.imencode('.jpg', image)
        image_base64 = base64.b64encode(encoded_image).decode('utf-8')

        # Create a message (MSG)
        message = MSG.format(image_id, image_base64)

        # Send a message to Azure
        print("Sending message: " + message)
        client.send_message(message)
        print("Message successfully sent")
    except Exception as e:
        print("Failed to send message:", str(e))
```

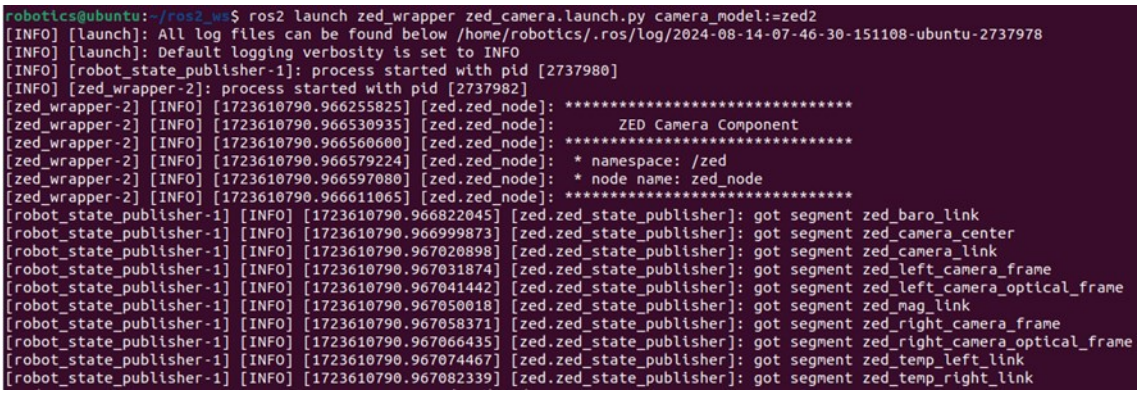
Kuvio 22. Azuren ja robotin yhteys

Yahboom Rosmaster X3-robotin kanssa tuli yllättäviä haasteita. Azure-yhteyttä varten ladattavaa azure-iot-device-kirjastoa ei saatu toimimaan halutulla tavalla. Linux-ympäristö ei ymmärtänyt kirjastojen olemassaoloa, vaikka kirjastot oli asennettu asianmukaisesti pip-komentoa käyttäen. Useamman kokeilun jälkeen tultiin siihen tulokseen, että ympäristö on turhan epävakaa, jotta siihen kannattaisi käyttää enempää aikaa. Robotti vaihdettiin toiseen robottiin, joka käyttää Jetson Orin Nanoa aivoinaan. Robotin nimi on Snower, ja alunperin se oli käytössä robotissa, joka oli suunniteltu Suomen kylmien säiden olosuhteisiin toimimaan lumiaurarobottina.

Käyttämällä Snowerin tekoälyalustaa on mahdollista demonstroida myös toisen robotin Mini-ATV:n yhteyttä Azureen sekä web-sivustolle. Molemmat robotit käyttävät samaa Jetson Orin Nano-tekoälyalustaa, joten sama konfiguraatio toimii teoriassa molemmille roboteille. Käytössä on ZED 2-kamera, jonka erikoisominaisuuksia ovat spatiaalisen tekoälyn hyödyntäminen, liiketunnistus ja syvyyshavainnointi. Tämän kameran käyttämiseen Jetsonilla vaaditaan kameran oma SDK nimeltä ZED SDK sekä Nvidian JetPack SDK Manager. Koska käytössä on Jetson Orin Nano, on asennettava tätä laitetta vastaava ZED SDK-versio, ZED SDK for JetPack 6.0 GA (L4T 36.3) 4.1, mikä tukee Jetson Orin-laitteita. Asennus on suoritettu onnistuneesti. Onnistuneen asennuksen jälkeen Jetson kykenee

suorittamaan ZED Explorer -sovellusta, jossa ZED 2-kamera näyttää reaaliajassa kameran kuvaa ja kykenee tallentamaan kuvausmateriaalia. Seuraavaksi on luvassa ZED 2-kameran yhdistäminen ROS 2-ympäristöön.

Jetson Orin Nanolle on valmiiksi asennettu ROS 2 Humble Hawksbill, joten työskentelyn voi aloittaa välittömästi. Luodaan mkdir-komennolla uusi hakemisto nimeltään ros2_ws ja siihen alikansio src. Kloonataan GitHubista Stereolabsin repositorysta zed-ros2-wrapper-moduulipaketti, joka sisältää ZED 2 -kameran ROS-topicit ja kameramallit 3D-visualisointia varten RVIZ 2:sessa. Tässä vaiheessa päivitetään järjestelmä komennolla *sudo apt update*, ja asennetaan tarvittava rosdep-riippuvuus. Kun riippuvuus on asennettu, on aika käyttää komentoa *colcon build* koko projektin kokoamiseksi yhdeksi kokonaisuudeksi. Asetetaan ympäristömuuttujat sekä suoritetaan ZED Node komennolla *ros2 launch zed-wrapper zed_camera_launch.py camera_model:zed2*. Seuraavassa kuviossa 23 voidaan nähdä, kuinka zed.zed_state_publisher vastaanottaa jatkuvasti muista aiheista dataa.



Kuvio 23. ZED 2-kameran käynnistäminen nodena

Azure-yhteyden luominen vaatii sen, että valitaan yksi aihe ja luodaan ROS node, joka lähettää tämän aiheen dataa viiden sekunnin välein Azureen base64-muodossa. Luodaan aiemmin luotuun ros2_ws-hakemistoon ROS 2 package nimeltä zed2_image_publisher, ja lisätään siihen riippuvuudet rclpy ja sensor_msgs. Asennetaan Jetson Orin Nanolle azure-iot-device-kirjasto, ja lisätään erikseen zed2_image_publisher.py-tiedostoon import-komennolla kirjastot rclpy, Node ja

Image, Azure Iot Hubin käsittelyyn liittyvät kirjastot IotHubDeviceClient ja ConnectionFailedError, kuvien käsittelyyn PIL-kirjasto ja Python-kirjastot io, base64 sekä json.

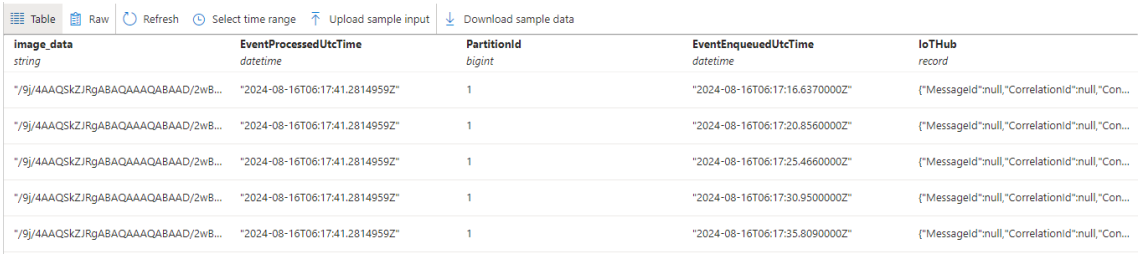
Määritellään Azure Iot Hubin yhteysasetukset yhteysmerkkijonoon. Luodaan Node nimeltä ZED2ImagePublisher, joka luo tilauksen ROS 2 Image-viesteillä, jotka tulevat ZED 2 -kameran topicista image_rect_color. Luodaan ajastin, joka kutsuu publish_to_azure-funktiota viiden sekunnin välein. Luodaan yhteys Azure Iot Hubiin yhteysmerkkijonon avulla, mikäli yhteyden muodostaminen epäonnistuu, määritellään, ettei clienttiä ole eli self.client = None.

Callback-funktio käsittelee topicista vastaanotetut viestit ja muuntaa ne PIL Image -objekteiksi. Seuraavassa kuviossa 24 nähdään, kuinka tämän jälkeen muunnetaan viimeisin self.last_image-muuttujaan tallennettu kuva base64-muotoon ja tarkistetaan sen koko. Mikäli koko ei ylitä 256 kilotavua, kuva lähetetään Azure Iot Hubiin. Määritellään vielä kaksi apufunktiota resize_image ja ros2_to_pil_image, joista resize_image muuntaa kuvien kokoa, mikäli ne ovat liian suuria ja ros2_to_pil_image-funktio muuntaa ROS 2 Image-viestin PILImage-objektiksi. Pääfunktiossa alustetaan ROS-järjestelmä, luodaan ZED2ImagePublisher-solmu ja käynnistetään se. Solmu pyörii niin kauan, kunnes käyttäjä sammuttaa sen.

```python
def listener_callback(self, msg):
    # Convert ROS2 Image message to PIL Image
    self.last_image = self.ros2_to_pil_image(msg)

def publish_to_azure(self):
    if self.last_image and self.client:
        try:
            # Resize image if needed
            resized_image = self.resize_image(self.last_image)

            # Convert the image to a base64 string
            buffered = io.BytesIO()
            resized_image.save(buffered, format="JPEG")
            image_base64 = base64.b64encode(buffered.getvalue()).decode('utf-8')

            # Create JSON payload
            payload = json.dumps({
                "image_data": image_base64
            })

            # Check payload size
            if len(payload.encode('utf-8')) > 256 * 1024:
                self.get_logger().error("Payload size exceeds 256 KB limit.")
                return

            # Send the data to Azure IoT Hub
            self.client.send_message(payload)
            self.get_logger().info("Image sent to Azure IoT Hub.")
        except Exception as e:
            self.get_logger().error(f"Failed to send image to Azure IoT Hub: {e}")

def resize_image(self, image, max_size=(800, 600)):
    """Resize image to fit within max_size without exceeding it"""
    image.thumbnail(max_size, PILImage.ANTIALIAS)
    return image
```

Kuvio 24. Azure-yhteyden luominen ja kuvien lähettäminen

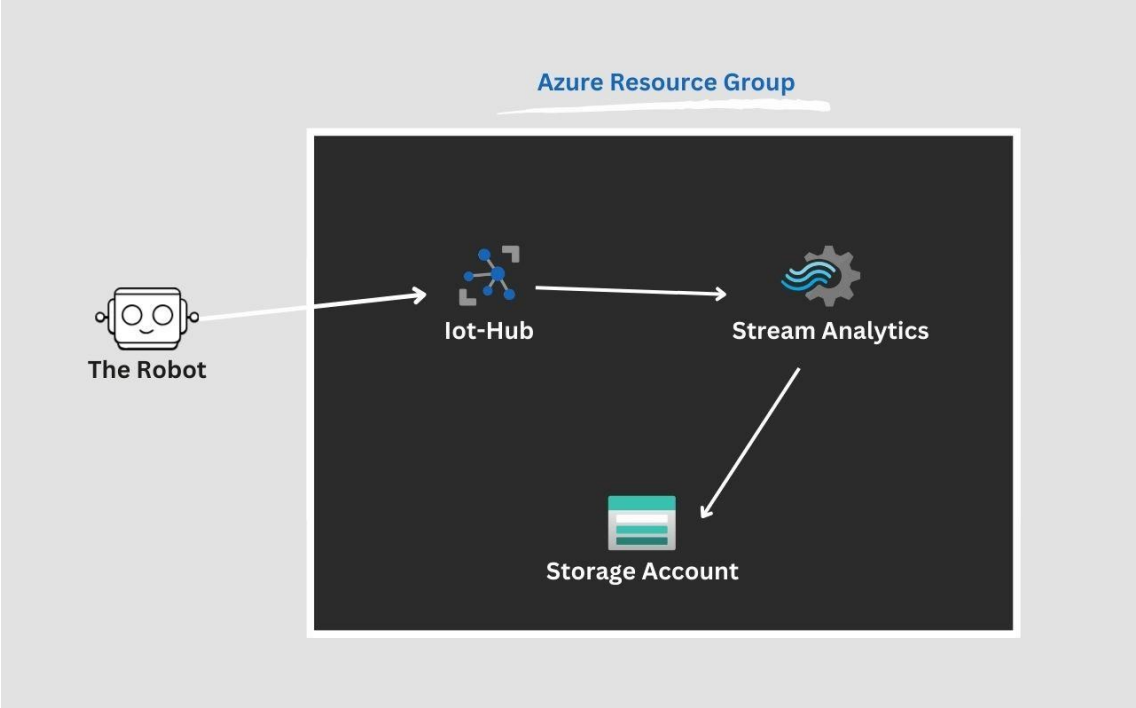
Azure Stream Analytics Job -palvelussa voidaan nähdä, kuinka lähetetty dataa tulee string-muotoisena taulukkona nimeltä image_data Azure Iot Hubiin. Seuraavassa kuviossa 25 nähdään, miltä tämä näyttää Azure Stream Analytics Jobsin Query -osiossa.



Kuvio 25. Datan vastaanottaminen Azure Stream Analytics Job -palvelussa

4.6 Yhteyden muodostaminen sivustolta Azureen

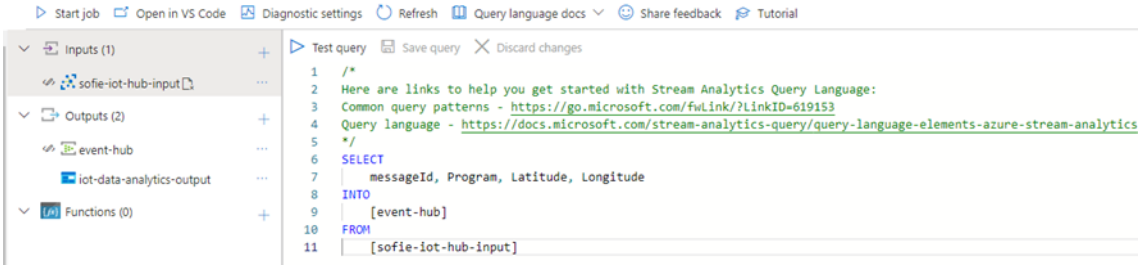
Projektin tässä vaiheessa on luotu yhteys Azuren ja robotin välille niin, että robotin on mahdollista lähettää keräämäänsä base64-muotoista dataa Azureen. Seuraava kuvio 26 ilmaisee tähän asti luotua yhteyttä ja käytettyjä Azuren palveluita. Robotti lähettää datan IoT-Hubiin, josta IoT-hub lähettää datan eteenpäin Stream Analyticsiin. Stream Analytics on palvelu Azuressa, joka analysoi reaaliajassa sen lävitse kulkevaa dataa. Palvelun analysoitua Iot-Hubista vastaanotettu data se lähettää datan eteenpäin Storage Accountin omaan blob-storageen, jossa data voidaan tallentaa ja käyttää myöhemmin.



Kuvio 26. Robotin Azure-palvelut

Seuraava vaihe kehityksessä on luoda yhteys web-sivuston sekä Azuren välille. Tätä varten tarvitaan vielä yksi Azure-palvelu lisää: Azure-Event Hubin. Azure Event Hub on Azuren palvelu, joka mahdollistaa suurien datamäärien keräämisen, striimaamisen ja käsittelyn. Opinnäytetyön aikana ja sen jälkeen on mahdollista, että robotin keräämä anturidata kulkee suurina määrinä Azuren lävitse. Event Hubin kaltainen suurien tietomäärien käsittelyyn soveltuva palvelu on juuri oivallinen opinnäytetyön tarpeisiin. Azuressa dataliikennettä muokataan niin, että IoT-Hubista data menee Stream Analyticsiin, mutta Storage Accountin sijasta

data ohjataan Event Hubiin. Käytännössä määräys dataliikenteestä tehdään Azuren Stream Analytics -palvelussa. Seuraavassa kuviossa 27 nähdään Azuren määrittelyt. Query eli kysely hakee Iot-Hubista datan ja syöttää sen Event Hubiin. Tämän määritelmän yläpuolella nähdään muuttujat, jotka valitaan Iot-Hubiin vastaanotetusta datasta ja lähetetään eteenpäin Azure Event Hubiin.



Kuvio 27. Azuren dataliikenne

Yhteyden muodostamisen demonstroimista varten luotiin Python-tiedosto, joka hakee Event Hubista yhteysmerkkijonon avulla robotilta Azuren Iot-Hubiin lähetettyä dataa. Yhteyden muodostaminen onnistuu yhteysmerkkijonojen avulla. Seuraavassa kuviossa 28 käytetään luodun blob-varaston ja Event Hubin yhteysmerkkijonoa, ja Event Hubin sekä blob-säiliön nimeä yhteyden muodostamiseksi Azureen.

```python
async def on_event(partition_context, event):
    # Print the event data.
    print(
        'Received the event: "{}" from the partition with ID: "{}"'.format(
            event.body_as_str(encoding="UTF-8"), partition_context.partition_id
        )
    )

    # Update the checkpoint so that the program doesn't read the events
    # that it has already read when you run it next time.
    await partition_context.update_checkpoint(event)

async def main():
    # Create an Azure blob checkpoint store to store the checkpoints.
    checkpoint_store = BlobCheckpointStore.from_connection_string(
        BLOB_STORAGE_CONNECTION_STRING, BLOB_CONTAINER_NAME
    )

    # Create a consumer client for the event hub.
    client = EventHubConsumerClient.from_connection_string(
        EVENT_HUB_CONNECTION_STR,
        consumer_group="$Default",
        eventhub_name=EVENT_HUB_NAME,
        checkpoint_store=checkpoint_store,
    )
    async with client:
        # Call the receive method. Read from the beginning of the
        # partition (starting_position: "-1")
        await client.receive(on_event=on_event, starting_position="-1")
```

Kuvio 28. Python-demo Azure-yhteyden luomiseen

Koodin alussa on määritelty omat muuttujat, jotka sisältävät nämä tiedot. Näitä ei näytetä kuviossa turvallisuussyistä. Funktiossa on_event luodaan toiminnallisuus, joka suoritetaan aina kun uusi event vastaanotetaan. Tapahtuman viesti tulostuu ja checkpoint päivittyy, jotta ohjelma ei lue samoja tapahtumia uudelleen ohjelman uudelleen käynnistyessä. Main-funktiossa luodaan BlobCheckpointStore-olio, joka käyttää annettua Blob Storagen yhteysmerkkijonoa sekä säiliön nimeä. Tämä varasto tallentaa checkpointin tiedot, jotka kertovat ohjelmalle, mihin kohtaan ohjelmassa ollaan jo päästy. Tämän jälkeen luodaan Event Hub-client, joka vastaanottaa eventtejä eli tapahtumia Azure Event Hubista. Tämän luomiseen käytetään Event Hubin yhteysmerkkijonoa, kuluttajaryhmää, Event Hubin nimeä sekä hetki aiemmin luotua checkpoint-varastoa. Kokonaisuutena main-funktio valmistelee yhteydet Blob Storageen ja Event Hubiin, luo tarvittavat resurssit ja aloittaa tapahtumien vastaanottamisen asynkronisesti.

Ongelma tässä ohjelmassa on se, että se on luotu Python-ohjelmointikieltä käyttäen. Web-sivusto ei ilman erilaisia tulkkeja ymmärrä Python-kieltä sellaisenaan, joten tämä ohjelma on käännettävä JavaScriptille. Koska aiemmin luodussa Vue-kehykseen luodussa web-sivustossa käytetään myös WebSocketia, on implementoitava se myös samaan JavaScript-tiedostoon Azure-palvelun kanssa. Luodaan uusi JavaScript-tiedosto nimeltään server.js. Tiedostoon tuodaan Azuren Event Hub-kirjasto sekä WebSocket-kirjasto. Tämän lisäksi määritellään aiemmassa Python esimerkissä mainitut yhteismerkkijonot. Tällä kertaa kuitenkin käytetään vain Azure Event Hubin yhteysmerkkijonoa, Event Hubin nimeä sekä $Default-kuluttajaryhmää.

Luodaan WebSocket-palvelin, joka kuuntelee porttia 3000. Kun uusi asiakas yhdistää WebSocket-palvelimeen, tulostetaan konsoliin “Client Connected.” Tämän lisäksi lyödään Event Hub-clientit, joita ovat consumerClient sekä producerClient, jotka käyttävät aiemmin määriteltyjä Event Hub-asetuksia. Seuraavassa kuviossa 29 voidaan nähdä, kuinka ohjelma luo tilauksen Event Hubin viestien vastaanottamiseksi. Funktio processEvents kutsutaan aina, kun Event Hubista saapuu viestejä. Funktio vastaanottaa kaksi parametria, jotka ovat events-taulukko saapuneista viesteistä ja context, joka sisältää lisätietoja viestien vastaanottamisen kontekstista. Tämän lisäksi funktio muokkaa viestit JSON-muotoon ja lähettää ne eteenpäin WebSocket-asiakkaalle.

```
// Luo WebSocket-palvelin
const wss = new WebSocket.Server({ port: 3000 });

wss.on('connection', ws => {
  console.log('Client connected');

  const consumerClient = new EventHubConsumerClient(consumerGroup, connectionString, eventHubName);
  const producerClient = new EventHubProducerClient(connectionString, eventHubName);

  // Event Hubin viestien vastaanotto
  const subscription = consumerClient.subscribe({
    processEvents: async (events, context) => {
      console.log(`Received ${events.length} event(s)`);
      for (const event of events) {
        const message = {
          body: event.body,
          partitionId: context.partitionId,
          consumerGroup: context.consumerGroup,
          direction: 'Message Received' // Lisätään viestin suunta
        };
        ws.send(JSON.stringify(message));
        console.log('Message sent to WebSocket client:', message);
      }
    },
    processError: async (err, context) => {
      console.error(`Error: ${err}`);
    }
  });
```

Kuvio 29. Event Hub- ja WebSocket-yhteyden luominen

Samaan server.js-tiedostoon on myös lisätty toiminallisuus liittyen TextFieldButton-komponenttiin. Käyttäjä kykenee lähettämään WebSocketin avulla viestejä käyttöliittymästä Azure Event Hubiin. Seuraavassa kuviossa 30 nähdään, kun WebSocket-asiakas lähettää viestin, se parsitaan eli erotellaan ymmärrettäviksi osiksi ja tulostetaan konsoliin. Viesti lähetetään Azure Event Hubiin käyttäen “producerClient”-clienttia. Lopuksi WebSocket-asiakkaalle lähetetään vahvistusviesti lähetetystä viestistä.

```
// Käsittele viestit asiakkaalta
ws.on('message', async (message) => {
  try {
    const parsedMessage = JSON.parse(message);
    console.log('Received message from client:', parsedMessage);
    // Lähetä viesti Azure Event Hubiin
    const batch = await producerClient.createBatch();
    batch.tryAdd({ body: parsedMessage.body });
    await producerClient.sendBatch(batch);
    console.log('Message sent to Azure Event Hub:', parsedMessage.body);

    // Lähetä viesti takaisin WebSocket-asiakkaalle vahvistuksena
    ws.send(JSON.stringify({ ...parsedMessage, direction: 'Message Sent' }));
  } catch (err) {
    console.error('Error sending message to Azure Event Hub:', err);
  }
});

ws.on('close', () => {
  console.log('Client disconnected');
  subscription.close();
});
});

console.log('WebSocket server is running on ws://localhost:3000');
```

Kuvio 30. Viestien lähettäminen Azureen WebSocketin avulla

Vue-projektiin lisättiin aiemmin TextFieldButton- sekä ChatBox-komponentit. ChatBox-komponentti on se komponentti, johon Azuresta haetut viestit sekä käyttäjän lähettämät viestit tulevat näkyviin. Seuraavassa kuviossa 31 nähdään ChatBox-komponentin template-rakenne. “connectionStatus” kertoo WebSocket-yhteyden tilan. Ohjelma käy läpi messages-taulukon ja luo jokaiselle viestille oman -säiliön, luo iteroinnin messages-taulukon jokaiselle message -objektille sekä ilmoittaa onko annettu viesti lähetetty vai vastaanotettu käyttöliittymässä.

```
<template>
  <div class="chatbox">
    <div v-if="connectionStatus" class="status">
      <p>{{ connectionStatus }}</p>
    </div>
    <div v-for="(message, index) in messages" :key="index" class="message">
      <p v-if="message.direction === 'Message Received'"><strong>Partition:</strong> {{ message.partitionId }}</p>
      <p v-if="message.direction === 'Message Received'"><strong>Group:</strong> {{ message.consumerGroup }}</p>
      <p><strong>Message:</strong> {{ message.body }}</p>
      <p>
        <strong>Status: </strong>
        <span :class="message.direction === 'Message Sent' ? 'sent' : 'received'">{{ message.direction }}</span>
      </p>
    </div>
  </div>
</template>
```

Kuvio 31. ChatBox-komponentin rakenne

ChatBox-komponentti käyttää TextFieldButtonin tavoin WebSocketia Azure Event Hub -yhteyden luomista varten. Seuraavassa kuviossa 32 voidaan nähdä, kuinka WebSocketilta saapuvat viestit sisällytetään messages-taulukkoon. “connectionStatus” kertoo WebSocket-yhteyden tilan, joka on oletuksena “Connecting…” Luodaan Vue-elinkaarimetodi mounted(), jota kutsutaan, kun ChatBox.vue-komponentti on luotu. mounted-metodi kutsuu connectWebSocket-metodia aloittaakseen WebSocket-yhteyden. “connectWebSocket”-metodissa luodaan yhteys paikalliseen porttiin 3000 ja määritellään tapahtumakäsittelijät “socket.onopen”, “socket.onmessage”, “socket.onerror” sekä “socket.onclose”. Tapahtumankäsittelijä “socket.onopen” tarkastelee WebSocketin tilaa, ilmoittaa tilan ChatBoxin sisällä käyttäjälle ja tulostaa viestin konsoliin. “socket.onmessage” käsittelee WebSocketin kautta vastaanotettuja viestejä, parsii ne ja lisää messages-taulukkoon.

Viestin suunta muutetaan samalla “Message received.” “socket.onerror” kutsutaan, mikäli WebSocket-yhteydessä ilmenee virheitä. Tällöin tapahtumankäsittelijä päivittää yhteyden tilan “WebSocket error”-tilaan ja tulostaa virheilmoituksen. “socket.onclose” kutsutaan, kun WebSocket-yhteys suljetaan. Tapahtumankäsittelijä päivittää yhteyden tilan suljetuksi ja tulostaa viestin konsoliin. Metodi nimeltä “sendMessageToChatBox(message)” lisää viestin ‘messages’-taulukkoon. Tämän ohjelma käyttää hyväkseen aiemmin luotua server.js-tiedostoa, missä luotiin WebSocket-palvelin ja Event Hub-clienit. Komponentti toimii luodun WebSocketin WebSocket-asiakkaana, joka vastaanottaa WebSocket-palvelimelta viestejä. ChatBox-komponentti käsittelee viestit, näyttää niiden tilan ja mahdollistaa viestien näyttämisen asiakkaille.

```
<script>
export default {
  data() {
    return {
      messages: [],
      connectionStatus: 'Connecting...' // Oletustila
    };
  },
  mounted() {
    this.connectWebSocket();
  },
  methods: {
    connectWebSocket() {
      const socket = new WebSocket('ws://localhost:3000');
      socket.onopen = () => {
        this.connectionStatus = 'Connected to WebSocket';
        console.log("WebSocket connection established");
      };
      socket.onmessage = (event) => {
        const message = JSON.parse(event.data);
        console.log('Received message from Event Hub:', message);
        this.messages.push({ ...message, direction: 'Message Received' });
      };
      socket.onerror = (error) => {
        this.connectionStatus = 'WebSocket error';
        console.error('WebSocket error:', error);
      };
      socket.onclose = () => {
        this.connectionStatus = 'WebSocket connection closed';
        console.log("WebSocket connection closed");
      };
    },
    sendMessageToChatbox(message) {
      this.messages.push(message);
    }
  }
};
</script>
```

Kuvio 32. WebSocket-yhteys ChatBox.vue-komponentissa

Vue-projekti käyttää Azure Event Hub-yhteyden luomiseen WebSocketia. WebSocket ja Azure Event Hub-clientien luonti toteutetaan server.js-nimisessä tiedostossa. Ongelma kuitenkin piilee siinä, että tämä ohjelma sekä paikallisen Vue-kehityspalvelimen käynnistyskomento pitää suorittaa samanaikaisesti, jotta yhteys toimii. Ratkaisu tähän on tehdä muutoksia projektin package.json-tiedostoon. Asennetaan ja käytetään concurrently-pakettia, jotta voidaan ajaa useita komentoja rinnakkain. Seuraavassa kuviossa 33 esitetään, miten näiden kahden komennon ajamisen onnistuu samanaikaisesti. Kirjoittamalla komennon *npm run*

dev on seuraavaksi mahdollista suorittaa komento *node src/server.js* sekä *npm run serve*, joka käynnistää sekä Vue-kehityspalvelimen että Node.js-palvelimen samanaikaisesti.

```
"scripts": {
  "serve": "vue-cli-service serve",
  "build": "vue-cli-service build",
  "lint": "vue-cli-service lint",
  "script": "node src/server.js",
  "dev": "concurrently \"npm run serve\" \"npm run script\""
},
```

Kuvio 33. Komentojen ajaminen samanaikaisesti

4.7 Robotti- ja sivustoyhteyden testaaminen

Tässä vaiheessa opinnäytetyötä on luotu yhteys robotilta Azureen ja yhteys sivustolta Azureen. Viimeisenä vaiheena on yhdistää Azuren avulla nämä kaksi, joka onnistuu tarkastelemalla ja muokkaamalla Azure Stream Analytics Job-osiota Azuressa. Stream Analyticsiin on määritelty, että Iot Hubiin tulevat viestit ja data lähetetään Azure Event Hubiin. Teoriassa tällä määritelmällä ja aiemmin luoduilla toteutuksilla sivuston ja robotin osalta pitäisi pystyä lähettämään ZED 2-kameran base64-muotoista dataa robotilta suoraan sivustolle ChatBox.vue-elementtiin. Azure Stream Analytics Job -palvelussa käyttäjän tulee painaa ”Start Job” -nappia, jotta yhteys käynnistyy.

Kun Job on käynnissä, on aika siirtyä käynnistämään sivusto tietokoneen Terminalissa komennolla *npm run dev*, joka käynnistää Vue-kehityspalvelimen ja npm-noden, joka luo WebSocket-palvelimen ja luo Event Hubiin yhteyden. Kun ChatBox.vue-komponentissa lukee robotin tietosivulla “WebSocket connection established”, on sivuston mahdollista vastaanottaa Event Hubista tulevia viestejä. Viimeisenä vaiheena yhteyden luomiseen on robotin käynnistäminen, ZED 2-kameran käyttöönotto ROS 2-ympäristössä komennolla *ros2 launch zed_wrapper zed_camera.launch.py camera_model:=zed2* ja suorittamalla ROS 2-noden käynnistyskomennon *ros2 run zed2_image_publisher zed2_image_publisher*. Tämä aiheuttaa sen, että robotti lähettää base64-muotoista dataa Azure Iot Hubiin. Data tulee perille Azure Iot Hubiin, mutta se ei kulje sieltä eteenpäin.

5 POHDINTA

5.1 Johtopäätökset

Opinnäytetyön tavoitteena oli luoda uusi ratkaisu ROS-järjestelmien käytölle samanaikaisesti ilman tarvetta käyttää useampaa kuin yhtä elektronista laitetta samanaikaisesti. Periaatteessa ratkaisu oli onnistunut, mutta käytännössä toteutus on hieman monimutkaisempi mitä ROS-alustan yhdelle laitteelle käynnistäminen olisi. Toteutuksessa täytyy käynnistää useampia ohjelmia kolmessa eri kohteessa, jotta yhteys toimii. Nettisivusto on käynnistettävä yhdellä komennolla, Azure Stream Analytics Job -palvelussa täytyy käydä laittamassa “Start Job” -nappi päälle, joka aloittaa datan keräämisen Azure Iot Hubista, analysoi saatua dataa ja lähettää sen eteenpäin Azure Event Hubiin.

Robotin päädyssä pitää käynnistää ensin ROS 2 -ympäristössä ZED 2 -kamera ja sen jälkeen ROS 2 -node, joka yhdistää Azureen ja lähettää topicista tulevaa dataa Azure Iot Hubiin. Jatkokehityksessä tämä kuitenkin helpottuisi muun muassa luomalla ZED 2 -kameran ja zed2_image_publisher-noden käynnistykseen oman launch -tiedoston ja implementoimalla sivuston omaan domainiin. Azuren Stream Analytics Jobsiin ei tarvitse koskea, jos sen pitää jatkuvasti päällä. Tämä voi kuitenkin aiheuttaa suuria kustannuksia Azuressa liikkuvista datamääristä riippuen. Projektin tässä vaiheessa järjestelmään on yhdistetty kokonaisesti vain yksi robotti, Snower. Tulevaisuudessa järjestelmään tulisi yhdistää ainakin neljä muuta robottia, jotka jokainen kaipaavat enemmän tai vähemmän muutoksia robotille luodun ROS 2 -noden koodiin. Samaa koodia voi kuitenkin käyttää yhden robotin kohdalla, joka käyttää Snowerin tavoin Jetson Orin Nanoa. Tämä robotti on Mini-ATV.

Vaikka keskitetyn ROS-palvelimen kehitysprojekti eteni monilta osin suunnitelmien mukaisesti, kohdattiin merkittävä tekninen haaste, joka estää kokonaisratkaisun toiminnan. Opinnäytetyön tavoitteena oli luoda Vue.js-pohjainen verkkosivusto, joka kommunikoi Azure Event Hubin ja Azure IoT Hubin avulla Snower-

robotin kanssa mahdollistaen reaaliaikaisen tiedonsiirron ja valvonnan. Ongelmana on kuitenkin ollut robotilta tulevan base64-muotoisen datan siirto Azure IoT Hubista Azure Event Hubiin ja edelleen verkkosivustolle.

Tämä ongelma osoittautui projektin kannalta kriittiseksi, sillä se esti koko tiedonsiirtoprosessin toiminnan. Vaikka base64-muotoisen datan siirron Azure IoT Hubiin onnistui, data ei kuitenkaan siirtynyt eteenpäin Azure Event Hubin kautta verkkosivustolle asti. Tämä voi johtua useista mahdollisista syistä, kuten yhteensopimattomuudesta käytettyjen palveluiden välillä, base64-datan käsittelyn haasteista Azure Event Hubissa tai puutteista tiedon reitityksessä.

Ongelmaa pyrittiin ratkaisemaan useilla eri tavoilla, kuten tarkistamalla tiedonsiirtoprosessin konfiguraatiot ja testaamalla datan käsittelyä. Valitettavasti nämä toimenpiteet eivät tuottaneet toivottua tulosta. Tämä korostaa tarvetta joko lisäosaamiselle Azure-pohjaisista tiedonsiirtoratkaisuista tai vaihtoehtoisille teknologisille ratkaisuille, jotka voisivat parantaa datan reitityksen ja käsittelyn luotettavuutta.

5.2 Oman oppimisen pohdinta

Mitä tulee luomani opinnäytetyön opettavaisuuteen, koen työni olleen hyvin antoisaa ja opin paljon uutta. Käytin lähteenäni virallisia dokumentaatioita eri teknologioiden ymmärtämiseen, sisäistämiseen ja soveltamiseen käytännössä. Opin Vue-frameworkin ja ROS-järjestelmät seuraamalla Youtube-tutoriaaleja aiheesta asiantuntijoiden ohjaamana. Voi kuitenkin olla, että osa seuraamistani tutoriaaleista voi olla vanhentunutta tietoa muun muassa kirjastojen tai joidenkin menetelmien osalta, mutta lopputulos puhuu puolestaan.

Verratessani katsomiani tutoriaaleja ja virallista dokumentaatiota kuitenkin huomasin, ettei niissä ollut eroja muuten kuin sisällön osalta: tutoriaalit luovat paljon monimutkaisempia kokonaisuuksia, kun taas dokumentaatio näyttää yksinkertaisen version siitä, mitä on mahdollista tehdä. Kaikki nämä tiedonlähteet auttoivat minua sisäistämään Vue-frameworkin ja ROS-järjestelmien toiminnan niin, että kykenin luomaan niillä oman, sovelletun kokonaisuuden. Opin ymmärtämään miten framworkit toimivat web-kehityksessä. En ollut aiemmin päässyt kokeilemaan

frameworkien käyttöä nettisivujen luonnissa, ja sain huomata, miten paljon helpompaa ja käytännöllisempää on käyttää frameworkkeja kuin pelkästään HTML-, CSS- ja JavaScript-tiedostoja toiminnallisten nettisivujen luomiseen. Vue-framework loi minulle sellaisen ympäristön työskennellä, missä ei tullut ongelmia moduulien toiminnan kanssa.

Projektin aikana pääsin myös tutustumaan Webpackiin, jota en ollut koskaan ennen käyttänyt. Käytin Webpack-projektia sivuston moduulien testaamiseen, sillä yksinään se ei onnistunut JavaScriptissä. Kokonaisuudessaan koko opinnäytetyöprojekti oli opettavainen kokemus, josta saan varmasti hyvän pohjan alan työelämään lähtiessäni. Toivon, että tulevaisuudessa pääsen oppimaan yhä lisää ja soveltamaan oppimaani käytännössä.

5.3 Jatkokehittämisaiheet

Opinnäytetyön tuloksia voidaan hyödyntää jatkokehityksessä. Opinnäytetyöllä on luotu demonstraatio siitä, miten tällaisen yhteyden ROS-laitteiden ja nettisivuston välillä on mahdollista toteuttaa. Se vaatii vielä muutamia tärkeitä asioita, kuten kirjautumisen ja käyttäjätunnukset väärinkäytösten varalta sekä toimivan Azure-yhteyden. Tulevaisuudessa olisi hyvä, että kuka tahansa ei pääse käyttämään sivustoa vapaasti, muuten kuin ehkä katselutilassa. Se on robottien kannalta tietoturvariski, jos kuka tahansa nettisivuston osoitteen tietävä henkilö voi lähettää Azureen viestejä chatboxin avulla tai lähettää komentoja roboteille.

Projektin tässä vaiheessa ei ole mahdollista ohjata robotteja komennoilla, mutta tulevaisuudessa tämän voi implementoida yhdeksi ominaisuudeksi. Tämän voi tehdä käytännössä luomalla uuden Azure-palvelun Azure Functions, joka mahdollistaisi viestien lähettämisen Event Hubista Iot Hubiin ja sieltä robotille. Robotin aiheita hyödyntämällä olisi mahdollista säädellä sivustolta käsin muun muassa robotin renkaiden liikettä ja nopeutta.

On myös tarpeellista, että tulevaisuudessa projektissa käytettävät Azuren yhteysmerkkijonot salataan esimerkiksi env-tiedoston sisälle. Tämä lisäisi osaltaan tietoturvaa, sillä projektin tässä vaiheessa yhteysmerkkijonot on kirjoitettu suoraan

muuttujiksi koodiin, mikä voi osaltaan aiheuttaa monia tietoturvariskejä. Jatkokehityksessä on siis lukemattomia mahdollisuuksia tämän projektin osalta, ja toivon, että tulevaisuudessa jatkokehittäjät kykenevät hyödyntämään opinnäytetyötä ja siihen liittyvää dokumentaatiota luodakseen projektistani vielä paremman, yhtenäisemmän kokonaisuuden monipuolisilla ominaisuuksilla.

LÄHTEET

Canva 2024a. Empowering the world to design. Viitattu 23.8.2024 https://www.canva.com/about/.

Canva 2024b. Add Text to a Photo for Free. Viitattu 23.8.2024 https://www.canva.com/features/add-text-to-photo/.

GitHub, Inc. 2024a. About GitHub and Git. Viitattu 31.7.2024 https://docs.github.com/en/get-started/start-your-journey/about-github-and-git.

GitHub, Inc. 2024b. About GitHub and Git. Viitattu 31.7.2024 https://docs.github.com/en/get-started/start-your-journey/about-github-and-git.

Kantor, I. 2022. An Introduction to JavaScript. Viitattu 15.04.2024 https://javascript.info/intro

Open Robotics 2022a. The ROS Ecosystem. Viitattu 10.4.2024 https://www.ros.org/blog/ecosystem/.

Open Robotics 2022b. ROS 2 In Science Robotics. Viitattu 10.4.2024 https://www.openrobotics.org/blog/2022/5/12/science-robotics-paper.

Open Robotics 2024c. Nodes. Viitattu 23.8.2024 https://docs.ros.org/en/humble/Concepts/Basic/About-Nodes.html.

Open Robotics 2024d. Understanding topics. Viitattu 23.8.2024 https://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html.

Open Robotics 2024e. Understanding services. Viitattu 23.8.2024 https://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Services/Understanding-ROS2-Services.html.

Open Robotics 2024f. Understanding actions. Viitattu 23.8.2024 https://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Actions/Understanding-ROS2-Actions.html.

Microsoft 2024a. What is Azure Iot Hub? Viitattu 2.7.2024 https://learn.microsoft.com/en-us/azure/iot-hub/iot-concepts-and-iot-hub.

Microsoft 2024b. Welcome to Azure Stream Analytics. Viitattu 2.7.2024 https://learn.microsoft.com/en-us/azure/stream-analytics/stream-analytics-introduction.

Microsoft 2024c. Azure Event Hub: A real-time data streaming platform with native Apache Kafka Support. Viitattu 2.7.2024 https://learn.microsoft.com/en-us/azure/event-hubs/event-hubs-about.

Microsoft 2024d. What is Azure? Viitattu 6.6.2024 https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-azure/.

Mozilla Corporation 2024a. CSS: Cascading Style Sheets. Viitattu 10.4.2024 https://developer.mozilla.org/en-US/docs/Web/CSS.

Mozilla Corporation 2024b. Introduction. Viitattu 23.8.2024 https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction.

Mozilla Corporation 2024c. Introduction to client-side frameworks. Viitattu 23.8.2024 https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Introduction.

Visual Studio Code 2024a. Getting Started. Viitattu 8.8.2024 https://code.visualstudio.com/docs.

Visual Studio Code 2024b. Why did we build Visual Studio Code? Viitattu 23.8.2024 https://code.visualstudio.com/docs/editor/whyvscode.

Vue.js 2024a. Introduction. Viitattu 30.7.2024 https://vuejs.org/guide/introduction.html.

Vue.js 2024b. Introduction. Viitattu 30.7.2024 https://vuejs.org/guide/introduction.html.

Vue.js 2024c. Introduction. Viitattu 23.8.2024 https://vuejs.org/guide/introduction.html.

Webpack 2024. Concepts. Viitattu 2.7.2024 https://webpack.js.org/concepts/.

W3C 2024. History, Viitattu 10.4.2024 https://www.w3.org/about/history/.

W3Schools 2024a. HTML Introduction, Viitattu 10.4.2024 https://www.w3schools.com/html/html_intro.asp.

W3Schools 2024b. JavaScript History, Viitattu 15.4.2024 https://www.w3schools.com/js/js_history.asp.