Tampereen ammattikorkeakoulu

Pilvitietokannan datan integraatio Rejlers Accelerated Operations -palveluun

Mikael Forstén

OPINNÄYTETYÖ
Toukokuu 2020

Konetekniikka
Koneautomaatio

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Konetekniikan koulutusohjelma
Koneautomaatio

FORSTÉN, MIKAEL:
Pilvitietokannan datan integraatio Rejlers Accelerated Operations -palveluun

Opinnäytetyö 50 sivua, joista liitteitä 9 sivua
Toukokuu 2020

Opinnäytetyön aiheena on pilvitietokannan datan integraatio Rejlers Accelerated Operations (AOS) -palveluun. Työn tilaaja on Rejlers, joka on pohjoismainen insinööritoimisto. AOS on Rejlersin palvelu, joka hyödyntää digitaalista kaksosta ja tarjoaa visuaalisen käyttöliittymän eri lähteistä kootulle datalle. Opinnäytetyön tavoitteena oli toteuttaa dynaamisen datan esittäminen AOS:n käyttöliittymässä.

Opinnäytetyössä käsitellään teollisen internetin perusteita, laitteiden yhdistettävyydessä käytettyjä protokollia sekä muutamia pilvialustoja ja niiden tarjoamia ominaisuuksia. Työssä ohjelmoitiin Siemens S7-1500 -ohjelmoitava logiikka lähettämään dataa MQTT:n välityksellä Azure IoT Hubiin. IoT Hubista data ohjattiin funktiosovelluksen avulla Azure Cosmos DB -tietokantapalveluun. Datan hakeminen toteutettiin tietokannan REST-rajapinnan avulla. Datan hakeminen ja esittäminen 2D- ja 3D-ympäristöissä tehtiin luomalla prototyypit React- ja A-Frame -käyttöliittymäkomponenteista. Komponentteihin tehtiin rajapinta niiden integroimiseksi ohjelmakoodiin.

Opinnäytetyön tuloksena syntyivät käyttöliittymäkomponentit ja niiden rajapintaa kuvaava dokumentaatio. Komponentit saatiin onnistuneesti lisättyä AOS:n ohjelmakoodiin toimintojen demonstroimiseksi. Komponentteja voidaan myös muokata ja jatkokehittää soveltumaan muihin vastaavanlaisiin tarpeisiin.

Dynaamisen datan esittäminen täydentää entisestään AOS:n ominaisuuksia dataintegraattorina. Visuaalisen esittämistavan avulla ajantasainen tieto on helposti saatavilla eri sidosryhmille. Työhön tarvittavan tiedon ollessa koottuna yhteen paikkaan sen etsimiseen kulutettu aika saadaan minimoitua, parantaen toiminnan tehokkuutta.

Asiasanat: teollinen internet, iot, pilvialusta

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Mechanical Engineering
Machine Automation

FORSTÉN, MIKAEL:
Integration of Cloud Database Data into Rejlers Accelerated Operations Service

Bachelor's thesis 50 pages, appendices 9 pages
May 2020

The subject of this thesis is integration of cloud database data into Rejlers Accelerated Operations Service (AOS). The thesis was done for Rejlers Finland, which is part of the Nordic Rejlers engineering group. AOS is a service by Rejlers that utilizes the concept of digital twin and presents a visual user interface for displaying data from various sources. The goal of the thesis was to develop a method for displaying dynamic data in the AOS user interface.

The thesis includes an overview of the basic principles of industrial internet, protocols used for device connectivity, and a selection of cloud platforms and the services they offer for industrial internet. In order to reach the goal of the thesis, a practical scenario was constructed. A Siemens S7-1500 PLC was programmed to send data to Azure IoT Hub via MQTT. Within Azure, a function app was used to direct the data flow from IoT Hub to Cosmos DB. Cosmos DB REST API was used to retrieve the data from the database. In order to fetch and display data in both 2D and 3D environments, prototypes of React and A-Frame user interface components were made. The components include an interface for integration into program code.

The components and a documentation of their interface are presented as the result of the thesis. The components were successfully implemented into AOS program code as a proof of concept. The components can also be modified and further developed to suit various other use cases.

Displaying dynamic data further enhances the capabilities of AOS as a data integrator. Visual presentation of data allows different stakeholders to easily access up-to-date information. When the required information is available in one place, the time spent looking for the information can be minimized, improving the overall efficiency.

Key words: industrial internet, iot, cloud platform

SISÄLLYS

LYHENTEET JA TERMIT

AMQP	Advanced Message Queuing Protocol
AOS	Accelerated Operations Service
API	Application programming interface
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
IoT	Internet of Things
I-IoT	Industrial Internet of Things
JSON	JavaScript Object Notation
MQTT	Message Queue Telemetry Transport
OPC UA	Open Platform Communications Unified Architecture
PLC	Programmable Logic Controller
POI	Point of Interest
PaaS	Platform as a Service
REST	REpresentational State Transfer
SaaS	Software as a Service
SASL	Simple Authentication and Security Layer
SSL	Secure Sockets Layer
SQL	Structured Query Language
TLS	Transport Layer Security

1 JOHDANTO

Teollinen internet tarjoaa yrityksille lukemattomia liiketoimintamahdollisuuksia laitteiden etähallinnasta ennakoivaan huoltoon ja älykkäisiin tehtaisiin. Keskeinen seikka teollisen internetin sovelluksissa on data, ja sen liikkuvuus järjestelmien välillä ja tehokas hyödyntäminen.

Opinnäytetyön tarkoituksena on dynaamisen datan esittäminen Rejlersin AOS-palvelussa. AOS on palvelu, joka integroi datan eri järjestelmistä ja esittää sen visuaalisessa käyttöliittymässä. Käyttökohteita dynaamiselle datalle on esimerkiksi ennakoivassa kunnossapidossa, kiinteistöjen ilmanvaihdon hallinnassa ja rakennustöiden seurannassa. Käytännön tavoitteeksi opinnäytetyölle asetettiin pilvialustasta haetun dynaamisen datan esittäminen AOS:n käyttöliittymässä.

Opinnäytetyössä tutustutaan teollisen internetin perusteisiin ja esitellään joitakin AOS:n ominaisuuksia. Lisäksi perehdytään teollisen internetin yhdistettävyydessä käytettyihin teknologioihin ja protokolliin, ja tutustutaan muutamaan valikoituun pilvialustaan ja niiden ominaisuuksiin.

Toteutusosiossa ohjelmoidaan ohjelmoitava logiikka, yhdistetään se Microsoft Azure -pilvialustaan ja toteutetaan datan hakeminen. Lopuksi luodaan datan esittämistä varten käyttöliittymäkomponentit ja rajapinta niiden käyttämiseksi ohjelmakoodissa. Viimeisessä luvussa arvioidaan opinnäytetyön tuloksia ja niiden hyödynnettävyyttä, sekä esitetään kehittämismahdollisuuksia ja loppupäätelmät.

2 TEOLLINEN INTERNET

2.1 Teollinen internet

Esineiden internetillä (IoT) tarkoitetaan esineiden yhdistämistä toisiinsa internetin välityksellä. Esineiden yhdistäminen internetiin mahdollistaa saumattoman yhteyden ihmisten ja laitteiden välillä. Pilvipalvelujen, analytiikan ja mobiiliteknologian ansiosta esineet ja järjestelmät keräävät ja hyödyntävät dataa, yhdistäen fyysisen ja digitaalisen maailman. (Oracle)

Teollisuuden kontekstissa esineiden internetistä käytetään käsitettä I-IoT tai suomeksi teollinen internet. Teollinen internet täydentää olemassa olevaa automaatiota ja auttaa yrityksiä luomaan uusia liiketoimintamalleja ja kasvattamaan liiketulosta. (Oracle)

Teollinen internet tuo mukanaan lukuisia hyötyjä eri toimialoille. Teollisuudessa datan keräämisellä ja pilvipalveluiden kehittyneillä analysointimenetelmillä voidaan tehostaa tuotantoa ja saavuttaa säästöjä. Laitekannan toiminta-aikoja voidaan optimoida seuraamalla antureilla laitteiden toimintaa ja ennakoimalla huollon tarvetta. Lisäksi teollinen internet mahdollistaa kokonaan uusien digitaalisten tuotteiden ja palvelujen syntymisen. (Collin & Saarelainen 2016, 18)

Yksi teollisen internetin hyödyntämiskohteista on rakennuksien LVI-tekniikan etävalvonta ja -hallinta. Esimerkiksi lämpötilaa, ilmankosteutta ja CO2-pitoisuutta mittaavien antureiden datalla voidaan parantaa energiatehokkuutta, paljastaa erilaisia vikoja ja kosteusvaurioita sekä ehkäistä terveyshaittoja. (Collin & Saarelainen 2016, 116-117)

2.2 Pilvialustat

Alusta toimii datavirtojen, yritysten tietojärjestelmien, tietokantojen, analytiikan ja sovelluskehityksen yhteen liittäjänä. Alusta voi tarjota myös teollisen internetin toteuttamiseen vaaditun teknologisen infrastruktuurin. Alusta mahdollistaa eri

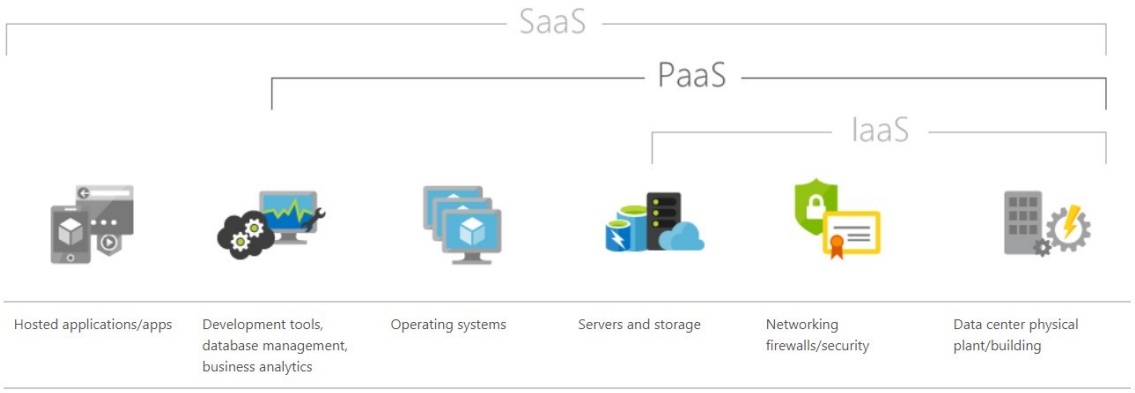
viestintäprotokollien tulkitsemisen sekä päätepisteiden löydettävyyden ja kommunikaation. (Collin & Saarelainen 2016, 228)

Pilvialustoja on lukuisia, sekä kaupallisia että avoimeen lähdekoodiin perustuvia. Osa alustoista on suunnattu kuluttajille, osa teollisen internetin tarpeisiin ja osa molempiin. (Collin & Saarelainen 2016, 228)

Alustoilla on vaihtelevia ominaisuuksia. Huomioitavia asioita alustan valinnassa ovat muun muassa liitettävyys ja eri protokollien tukeminen, tietokantaratkaisut, analytiikka, visualisointi- ja raportointityökalut sekä integroitavuus ulkoisiin järjestelmiin. Oleellisia seikkoja ovat myös tietoturva, kustannukset sekä tulevaisuuden tarpeisiin vastaaminen. (Collin & Saarelainen 2016, 230-233)

2.2.1 Palvelumallit

Pilvialustojen tarjoamat palvelumallit voidaan jakaa kolmeen pääluokkaan: IaaS, PaaS ja SaaS (kuva 1). Ostettava palvelu riippuu asiakkaan tarpeista. (AWS)



KUVA 1. Pilvialustojen palvelumallit (Microsoft).

IaaS eli Infrastructure as a Service tarkoittaa palvelumallia, jossa pilviyritys tarjoaa palveluna fyysisen infrastruktuurin, kuten verkkoyhteydet, palvelimet ja tallennustilaa. IaaS tarjoaa pohjan, jonka päälle asiakas voi rakentaa oman IT-ympäristönsä. (AWS) Pilvi-infrastruktuuria palveluna tarjoavat muun muassa suuryritykset kuten yhdysvaltalaiset Amazon Web Services ja Microsoft Azure, sekä kiinalainen Alibaba Cloud (Terraform).

PaaS eli Platform as a Service -mallissa myytävänä palveluna on ympäristö, jossa asiakas voi ajaa omia sovelluksiaan. PaaS-mallissa tarjotaan valmis infrastruktuuri sisältäen laitteiston, käyttöjärjestelmän sekä näiden hallinnoinnin kuten päivitykset ja tietoturvan. Tällöin asiakkaan tarvitsee keskittyä vain oman sovelluksensa hallinnoimiseen ja liiketoiminnan harjoittamiseen. (AWS)

SaaS eli Software as a Service -mallissa asiakkaalle tarjotaan valmis digitaalinen palvelu, joka on palveluntarjoajan hallinnoima. Asiakkaan huolehdittavaksi jää ainoastaan palvelun käyttäminen ja sen hyödyntäminen omiin tarkoituksiinsa. (AWS)

2.3 Analytiikka

Data on keskeinen resurssi teollisessa internetissä. Teollisuusautomaation anturit, laitteet ja hallintaliittymät sekä niiden muodostamat verkot muodostavat rungon, jonka päälle teollisen internetin jatkuva datavirta on mahdollista toteuttaa. Prosesseista syntyvä raakadata viedään analytiikka-alustalle, jossa datasta jalostetaan arvokasta tietoa. (Collin & Saarelainen 2016, 50)

Jotta data voidaan käyttää hyödyksi, sitä täytyy osata tarkastella oikein. Tärkeää on tuntea datan konteksti, kuten tuotantoympäristö ja normaalista toiminnasta kertovat arvot. (Collin & Saarelainen, 206)

Analytiikkaa automatisoidaan algoritmien eli matemaattisiin kaavoihin perustuvien ohjelmakoodien avulla. Tehokkaampien algoritmien avulla analytiikkaa voidaan viedä kuvailevasta analyysistä, aiemmin tapahtuneen tarkastelusta, ohjailevaan analyysiin, jossa pyrkimyksenä on laitteen tai prosessin toiminnan optimointi reaaliaikaisesta analyysistä saadun tiedon perusteella (kuva 2). (Collin & Saarelainen 2016, 208)

KUVA 2. Analytiikan tasot (Datatiede).

Tietoliikenteessä paikallisten laitteiden ja pilven välillä esiintyy aina viivettä, joka ei kaikissa tapauksissa ole suotavaa. Lisäksi kaiken datan vieminen pilveen voi kuormittaa verkkoa suhteettoman paljon. Siksi suuntana on viime aikoina ollut suorittaa osa analytiikasta paikallisesti pilven sijaan. (Collin & Saarelainen 2016, 214-215)

Edge computing tarkoittaa analytiikan viemistä lähemmäs datan syntypaikkaa. Suomeksi voidaan puhua hajautetusta laskennasta tai lähilaskennasta. Lähilaskennan etuna on vähentynyt tietoliikenteen ja tallennustilan tarve sekä alhaisemmat viiveet. Lähilaskentaa on järkevää hyödyntää esimerkiksi toiminnoissa, joissa analysoidaan suuria datamääriä ja poikkeamat on havaittava nopeasti. (Collin & Saarelainen 2016, 215)

Fog computing eli usvalaskenta yhdistää pilvi- ja lähilaskennan. Usvalaskennassa suuri osa analytiikasta suoritetaan lähilaskennan tavoin paikallisesti, mutta osa datasta viedään pilveen. Syynä analysoida osa datasta pilvessä voi olla erityistä suorituskykyä vaativa analyysi tai tarve suhteuttaa se pilvessä olevaan suureen datamäärään. (Collin & Saarelainen 2016, 215-216)

2.4 Datan visualisointi

Jotta data ja analytiikan tulokset saadaan ihmiselle ymmärrettävämpään muotoon, tarvitaan visualisointia. Esittämällä data graafisesti erilaisten mittarien ja kaavioiden muodossa siitä on helpompi havaita malleja ja trendejä. Pilvialustat sisältävät usein valmiita työkaluja analytiikkaan ja tulosten visualisointiin tai mahdollisuuden viedä tietoa erillisiin visualisointiohjelmistoihin. (Collin & Saarelainen 2016; 214, 232)

Visualisoinnissa voi hyödyntää myös digitaalista kaksosta. Digitaalinen kaksonen tarkoittaa fyysisen tuotteen digitaalista mallia, jossa on kuvattuna fyysisen tuotteen ominaisuuksia. Digitaalinen kaksonen voi olla esimerkiksi 3D-malli kokonaisesta tehtaasta. (Collin & Saarelainen 2016, 214) Digitaalisen kaksosen avulla voidaan seurata tuotteen ominaisuuksia reaaliaikaisesti sekä havainnollistaa erilaisten muuttujien vaikutuksia tuotteeseen ennen fyysisten muutosten tai investointien tekemistä. (Siemens)

Myös lisätyn todellisuuden hyödyntäminen on lisääntymässä eri teollisuuden aloilla. Lisättyä todellisuutta voidaan käyttää esimerkiksi kokoonpanotyössä työohjeiden tai muun tiedon tuomiseen suoraan näkökenttään, tai huoltotyössä laitteen huollontarpeen havaitsemiseen ilman laitteen purkamista. (Wright 2017)

2.5 Tietokannat

Tietokanta on organisoitu varasto, jonne tallennetaan jäsenneltyä dataa. Yhdessä hallintajärjestelmän kanssa ne muodostavat kokonaisuuden, jonka avulla dataa voidaan helposti tarkastella, lisätä, muokata ja poistaa. Data on usein jäsenneltynä erilaisiin tauluihin riveille ja sarakkeille. Tietokannat käyttävät datan käsittelyyn tyypillisesti IBM:n 1970-luvulla kehittämää SQL-kieltä. (Oracle)

SQL-malli edellyttää, että tietokantaan vietävä data noudattaa ennalta määrättyä rakennetta, mikä voi muodostua pullonkaulaksi datamäärän kasvaessa. NoSQL eli ei-rakenteellinen tietokanta on tietokantatyyppi, joka sallii datan tallentamisen ilman ennalta määrättyjä rakenteita. NoSQL on yleistynyt etenkin esineiden internetin monimuotoisen datan tallentamisessa. Datan hakeminen on NoSQL:ssä hitaampaa ja vaikeampaa, mutta se voi kuitenkin sisältää SQL-rajapinnan, jolla

saadaan käyttöön SQL:n tehokkaita hakutoimintoja. (Collin & Saarelainen 2016, 196-198)

Pilvipalvelut sisältävät usein teollisen internetin tarpeisiin soveltuvia tietokantapalveluja, jotka ovat toimivat yhdessä erilaisten laitehallinta-, analytiikka- ja visualisointipalvelujen kanssa. Pilvitietokannan etuina ovat edullisuus, skaalattavuus ja käyttöönoton helppous. Pilven kanssa esiintyy kuitenkin väistämättä viiveitä ja mahdollisia katkoksia yhteyksissä, joten ajan suhteen kriittisiin sovelluksiin se ei välttämättä sovellu. (Oracle; Collin & Saarelainen 2016, 202)

2.6 Tietoturva

Tietoturva on teollisen internetin suurimpia jarruttavia tekijöitä. Perinteisesti teollisuuden laitteita ei ole kytketty suoraan internetiin, joten niiden tietoturvaa ei välttämättä ole mietitty kovin pitkälle. Teollisen internetin myötä laitteet ovat usein kytkettyinä internetiin erilaisten hallintaohjelmistojen ja toiminnanohjausjärjestelmien välityksellä, mikä voi altistaa ne hyökkäyksille. (Collin & Saarelainen 2016, 242-243)

Lähtökohta järjestelmien tietoturvan varmistamiselle on verkkotopologian ja datavirtojen hahmottaminen. Järjestelmiä monimutkaistavien uusien kerrosten lisäämiset on syytä suunnitella tarkasti, jotta ne voidaan toteuttaa tietoturvan kannalta kestävällä tavalla. Teollisen internetin tietoturvaan pätee pohjimmiltaan samat periaatteet kuin yleisestikin tietoturvassa, kuten palomuurit, monikerroksinen suojaus ja tunnistautuminen. (Collin & Saarelainen 2016, 245-246)

Esineiden internetin tietoturvaan liittyy lukuisia haasteita. Esineiden internetin ollessa nopeasti kehittyvä markkina, kiire saada uusia laitteita markkinoille saattaa asettaa tietoturvan taka-alalle. Laitteiden resurssit eivät aina ole riittäviä vahvojen tietoturvamenetelmien käyttämiseen. Riskejä tietoturvan kannalta ovat myös vanhentuneet ja päivittämättömät laitteet sekä tietoturvastandardien pirstaleisuus. (Rouse 2018)

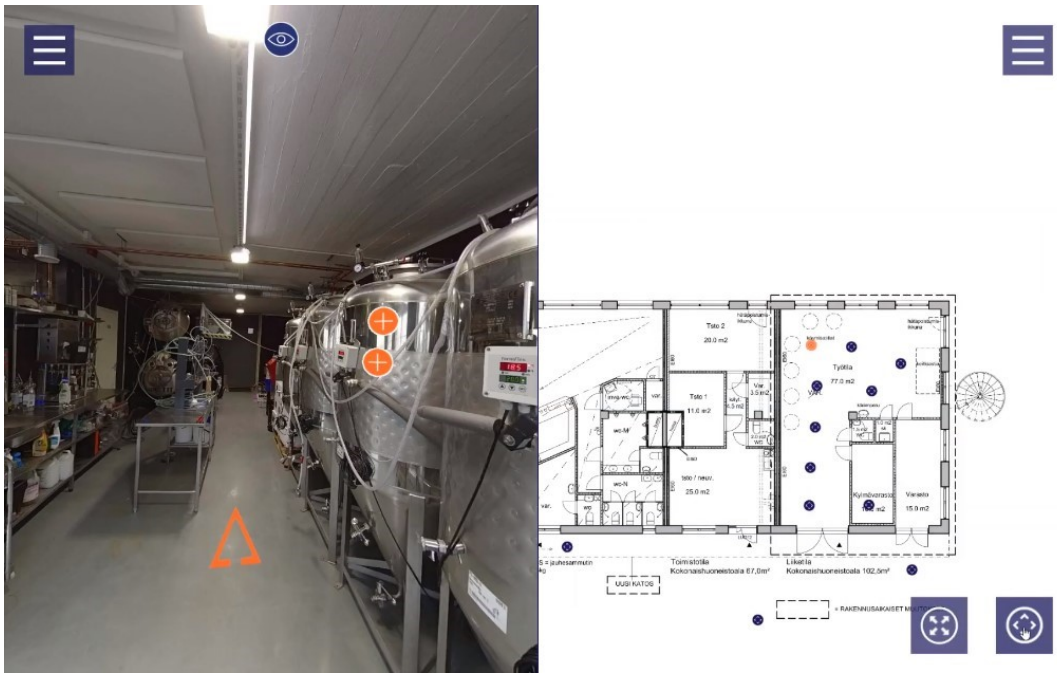
3 REJLERS ACCELERATED-OHJELMA

Rejlersin Accelerated-ohjelma on digitaalisten ratkaisujen kokonaisuus. Ratkaisut tehostavat liiketoimintaa lisätyn todellisuuden, oppivien algoritmien ja digitaalisten mallien avulla. Sovellettavia toimialoja ovat teollisuus, rakentaminen, energia ja infra. (Rejlers)

3.1 Accelerated Operations Service

AOS on digitaalista kaksosta ja lisättyä todellisuutta hyödyntävä kokonaisratkaisu huolto- tuki- ja suunnittelutoimintojen kehittämiseen. AOS on visuaalinen käyttöliittymä digitaaliselle sisällölle. AOS integroi datan toiminnanohjaus-, hallinta- ja huoltojärjestelmistä ja esittää sen fyysisestä tilasta luodussa digitaalisessa mallissa. Digitaalinen malli voidaan luoda laserkeilauksella, fotogrammetrialla tai drone-kuvauksella. AOS toimii selaimessa ja on saatavilla sekä mobiili- että työpöytälaitteessa. (Rejlers)

Navigoiminen AOS:ssä tapahtuu kolmiulotteisessa mallissa klikkaamalla 360-kuvassa ja pohjakuvassa sijaitsevia navigointipisteitä (kuva 3). Näkymän voi asettaa koko ruudun kokoiseksi tai jaetuksi mallin ja pohjakuvan välillä.



KUVA 3. Navigoiminen AOS:ssä.

AOS:ssä käytettävissä olevat toiminnot riippuvat käyttäjälle asetetusta tasosta (taulukko 1). Esimerkiksi Viewer-tason käyttäjä voi tarkastella mallia ja dataa, mutta ei lisätä tietoja. Operator-tason käyttäjällä on oikeudet lisätä ja muokata POI-pisteitä ja dataa. (Rejlers)

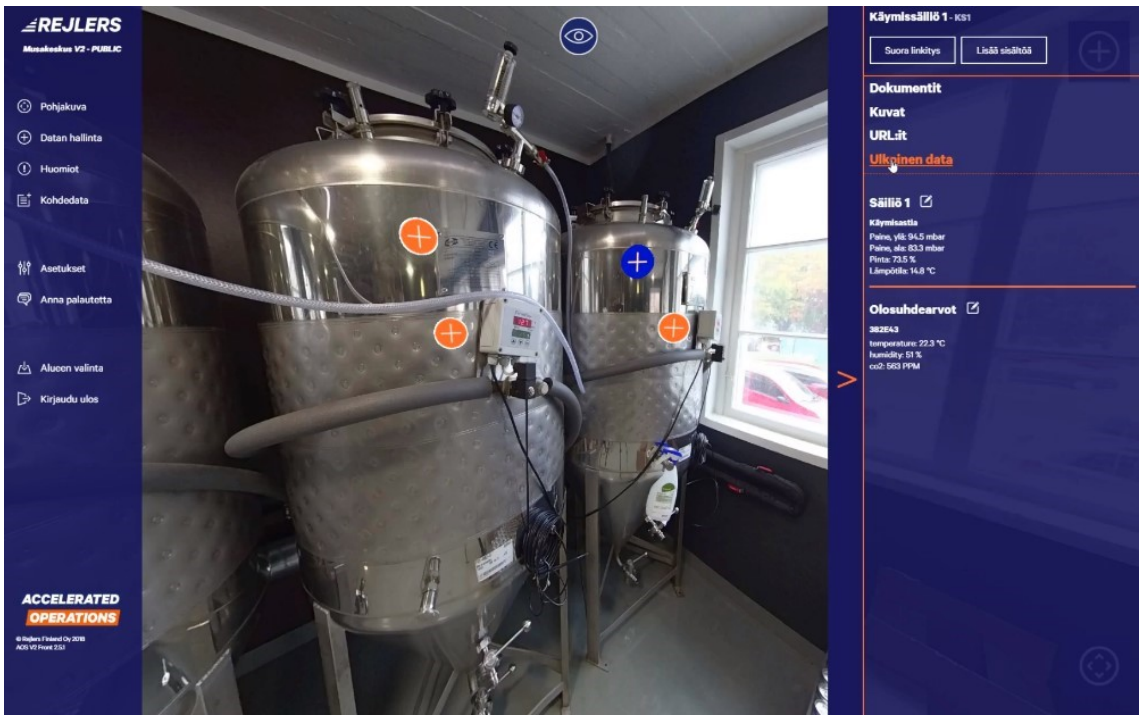
TAULUKKO 1. AOS:n käyttäjätasot. (Rejlers)

User levels for AOS					
	Guest	Viewer	User	Operator	Editor
360 image view/watch	x	x	x	x	x
Poi, item and data view/watch		x	x	x	x
Markers add/edit			x	x	x
Poi, ext360, item and data add/edit				x	x
Model editor					x

3.1.1 Datamalli ja POI-pisteet

Palveluun lisättävät mallit sisältävät kohteita, joista tehdään POI-pisteitä kuviin. POI-pistettä klikkaamalla saa auki kyseisen kohteen tietoja (kuva 4). Kohteisiin voi olla tallennettuna esimerkiksi dokumentteja, kuvia ja linkkejä. Kohteen tiedoissa voidaan esittää myös suoraan ulkoisesta lähteestä haettua dataa. Tietoa on mahdollista esittää myös suoraan 360-kuvassa (kuva 5).

Malliin tuotu data on tarkasteltavissa hierarkkisesti jäsenneltynä datan hallinnassa. Kohteita voidaan tuoda palveluun esimerkiksi massana tuotannonohjausjärjestelmästä.



KUVA 4. POI-pisteen data AOS:ssä.

KUVA 5. 360-kuva ja datan esittäminen.

3.1.2 Huomioiden kirjaaminen ja käsittely

Erilaiset huomiot, kuten vikatilanteet ja tiedot huoltotoimenpiteistä näkyvät kootusti huomiot-ikkunassa (kuva 6). Avaamalla tietyn huomion näkee siihen liittyvät tarkemmat tiedot. Huomioihin on mahdollista liittää viestejä ja tiedostoja, ja tiedot voidaan myös lähettää suoraan sähköpostilla halutuille kontakteille.

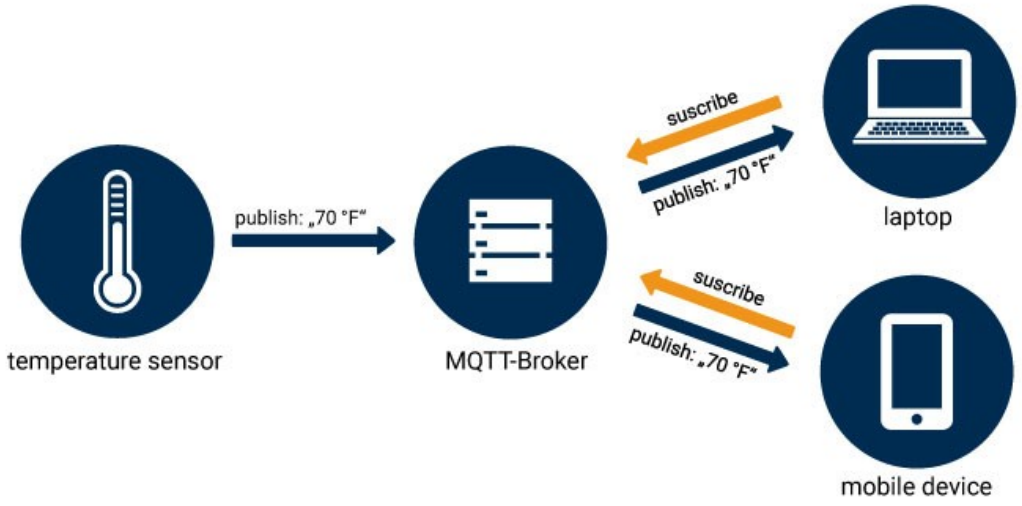
KUVA 6. Huomiot-ikkuna.

4 TEOLLISUUSAUTOMAATION YHDISTETTÄVYYS

4.1 MQTT

MQTT on yksinkertainen ja kevyt viestintäprotokolla laitteiden väliseen kommunikaatioon. Se on suunniteltu toimimaan erityisesti rajallisten resurssien ja epäluotettavien verkkojen kanssa. MQTT-liikenne on mahdollista salata SSL-protokollalla. Ominaisuuksiensa vuoksi se soveltuu hyvin teollisen internetin sovelluksiin. (mqtt.org)

MQTT toimii *publish/subscribe* -periaatteella, jossa asiakasohjelma *client* voi sekä julkaista viestejä että vastaanottaa niitä (kuva 7). Viestit kuuluvat aina tietyn aiheen eli *topicin* alle, jonka client tilaa vastaanottaakseen siihen liittyviä viestejä. Clientit ovat yhteydessä *brokeriin,* joka toimii viestien välittäjänä. Saman topicin voi tilata useampi client. (Mosquitto)



KUVA 7. MQTT:n toimintaperiaate (OPC-Router).

4.2 AMQP

AMQP on alun perin finanssialalla kehitetty avoimeen lähdekoodiin perustuva viestintäprotokolla, joka MQTT:n tavoin toimii publish/subscribe -periaatteella. AMQP:n vahvuuksia ovat yksinkertaisuus, varmuus, tietoturva ja yhteensopivuus eri järjestelmien välillä. (amqp.org; Collin & Saarelainen 2016, 188)

Verrattuna MQTT:hen AMQP sisältää enemmän ominaisuuksia ja käyttää myös enemmän resursseja. AMQP:n ominaisuuksia ovat muun muassa SASL- ja TLS-salaus sekä vahvistuksien lähettäminen viestejä vastaanottaessa, mikä takaa viestien perillepääsyn. (Hackernoon 2017; Rouse 2018)

4.3 OPC UA

OPC UA on standardisoitu, vuonna 2008 julkaistu protokolla, joka mahdollistaa tiedonsiirron eri yhteysprotokollia käyttävien laitteiden ja järjestelmien välillä. Verrattuna edeltäjäänsä OPC Classiciin, OPC UA on alustasta riippumaton ja skaalattavissa, mikä tarkoittaa, että sitä voidaan käyttää erilaisissa ympäristöissä, sekä suljetuissa verkoissa että internetissä. OPC UA:ta kehittää ja ylläpitää yhteistyössä eri laitevalmistajien kanssa OPC Foundation. (OPC Foundation)

4.4 Node-RED

Node-RED on NodeJS-pohjainen ohjelmointityökalu flow-muotoiseen ohjelmointiin, jossa ohjelman toiminta kuvataan virtauksenomaisena. Node-RED sisältää selaimessa esitettävän graafisen käyttöliittymän, jolla ohjelman luominen on tehty yksinkertaiseksi ilman tarvetta ymmärtää varsinaista ohjelmakoodia. Flow-ohjelmoinnissa data ohjataan nodelle, joka suorittaa datalle määrätyn funktion, minkä jälkeen käsitelty data välitetään eteenpäin. Ohjelma luodaan yhdistämällä nodeja toisiinsa. (Node-RED)

Node-RED on laajennettavissa asentamalla uusia nodeja. Node-RED-kirjastossa on saatavilla nodeja esimerkiksi S7-, Modbus- sekä OPC UA -protokollilla tapahtuvaan kommunikaatioon. Myös moniin pilvipalveluihin voidaan yhdistää suoraan tätä varten kehitettyjen nodejen avulla. (Node-RED)

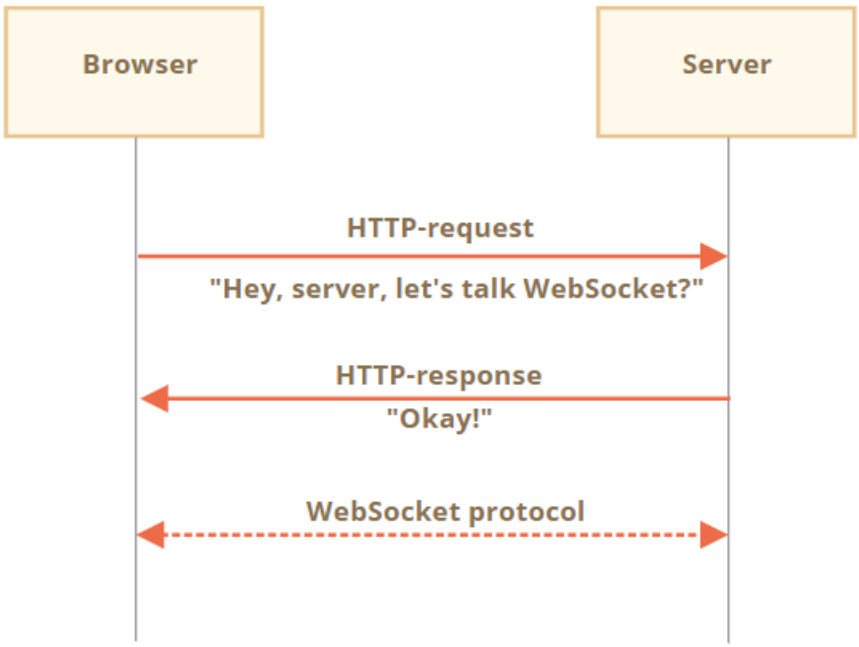
4.5 REST API

REST on paradigma laitteiden verkon välityksellä tapahtuvan kommunikaation kehittämiseen. Se on suosittu yksinkertaisuutensa ja järjestelmäyhteensopivuutensa takia. Rajapinta on RESTful, kun se täyttää tietyt REST-paradigman asettamat vaatimukset. (Rouse 2019)

REST perustuu vuorovaikutukseen verkkopalvelimella olevien resurssien kanssa hyödyntäen HTTP-metodeja, kuten GET, PUT, POST ja DELETE. Resurssien tulee olla tunnistettavissa URL-osoitteen perusteella ja manipuloitavissa käyttäen ainoastaan HTTP-metodeja. REST on tilaton, eli rajapinta ei tallenna tietoa pyyntöihin liittyen, vaan pyyntöjen prosessointiin tarvittava informaatio on pyynnöissä itsessään. (Rouse 2019)

4.6 WebSocket

WebSocket on protokolla selaimen ja palvelimen väliseen kommunikaatioon pysyvän yhteyden välityksellä. Dataa voidaan lähettää molempiin suuntiin paketteina ilman yhteyden katkaisemista tai erillisten HTTP-pyyntöjen tekemistä (kuva 8). WebSocket on hyödyllinen erityisesti käyttösovelluksissa, joissa tarvitaan jatkuvaa datavirtaa. (javascript.info)



KUVA 8. WebSocket-yhteyden muodostusta kuvaava kaavio (javascript.info).

WebSocketilla voi lähettää ja vastaanottaa binääristä ja tekstimuotoista dataa. WebSocket-protokollan metodeja ovat datan lähettäminen ja yhteyden sulkeminen. WebSocket kuuntelee tapahtumia, joita ovat *open, message, error* ja *close.*

Protokolla ei itsessään sisällä mekanismeja tunnistautumiseen tai yhteyden uudelleenmuodostamiseen, vaan ne on toteutettava erillisillä kirjastoilla tai manuaalisesti ohjelmakoodissa. (javascript.info)

5 TEOLLISUUDEN PILVIALUSTOJA

5.1 Amazon Web Services

Amazon Web Services (AWS) on Amazonin kehittämä pilvialusta. AWS tarjoaa luotettavan ja skaalattavan pilvialustan alhaisilla kustannuksilla. AWS on käytössä eri alojen yrityksissä 190 maassa. AWS tarjoaa muun muassa infrastruktuurin asiakkaan omien sovellusten ja verkkosivujen pyörittämiseen sekä tallennustilaa ja tietokantaratkaisuja. AWS:n palvelujen hinnoittelu määräytyy käytön mukaan ilman etukäteismaksuja tai pitkäaikaisia sitoumuksia. (AWS)

Tietokantapalveluja on tarjolla erilaisiin käyttötarkoituksiin. Esimerkiksi Aurora, RDS ja Redshift perinteisen relaatiotietokannan ylläpitoon, DynamoDB avainarvo-tyyppiseen tietokantaan sekä Timestream aikasarjadatan tallentamiseen. (AWS)

AWS IoT Core on palvelu, joka sallii laitteiden yhdistettävyyden muihin AWS:n palveluihin sekä datan tallennuksen ja prosessoinnin. Tuettuja yhteysprotokollia ovat MQTT ja HTTP, joista MQTT myös WebSocketin välityksellä. AWS IoT Device SDK sisältää asiakasohjelmakirjastoja ja ohjeita laitteiden yhdistämiseen. SDK tukee C:tä, JavaScriptiä ja Arduinoa. (AWS)

5.2 Microsoft Azure

Azure on Microsoftin pilvialusta. Azure tarjoaa laajan valikoiman palveluja esimerkiksi terveydenhuoltoon, kaupan alalle, teollisuuteen ja valtiollisille toimijoille. Azuren palvelut ovat hinnoittelultaan verrattavissa AWS:ään lukuun ottamatta Windows Server ja SQL Server -palveluita, jotka se tarjoaa halvemmalla. (Microsoft)

AWS:n tavoin Azure tarjoaa erilaisia tietokantapalveluja erilaisiin käyttötarpeisiin. Vaihtoehtoina ovat muun muassa globaalisti hajautettu Cosmos DB ja tekoälyominaisuuksia sisältävä Azure SQL Database. (Microsoft)

Azuren IoT-palveluihin lukeutuvat muun muassa IoT Hub ja IoT Central. IoT Hub mahdollistaa laitteiden yhdistämisen ja viestien ohjaamisen muihin Azuren palveluihin. IoT Central tarjoaa samanlaiset toiminnot, mutta yksinkertaistettuna ja valmiita malleja hyödyntäen, vähentäen hallinnoinnin tarvetta. Tuettuja yhteysprotokollia ovat MQTT ja AMQP, jotka ovat tuettuja myös WebSocketin välityksellä, sekä HTTP. (Microsoft)

5.3 Siemens MindSphere

MindSphere on Siemensin kehittämä pilvialusta, joka tarjoaa ratkaisuja datan keräämiseen ja sen reaaliaikaisen tarkastelemiseen ja analytiikkaan. MindSphere tarjoaa palveluina erilaisia työkaluja tehokkuuden ja tuottavuuden parantamiseen sekä alustan omien sovelluksien kehittämiselle ja julkaisemiselle. (Siemens)

MindSphere tarjoaa erilaisia käyttömahdollisuuksia riippuen käyttäjän roolista. Loppukäyttäjälle tarjotaan SaaS-ratkaisuja teollisen internetin hyödyntämiseen omassa liiketoiminnassaan, ohjelmistokehittäjät voivat kehittää omia sovelluksiaan ja myydä niitä MindSpheren sovelluskaupassa ja operaattori voi rakentaa ja hallinnoida omia palveluja MindSpheressä ja myydä niitä asiakkailleen. (Siemens)

Datan viemiseen MindSphereen on useita vaihtoehtoja. MindConnect Nano ja MindConnect IoT2040 ovat sulautettuja tietokoneita, jotka keräävät dataa kentältä ja vievät sen salattua internet-yhteyttä käyttäen MindSphereen. Datan kerääminen onnistuu S7-protokollaa käyttäen, mahdollistaen esimerkiksi vanhempien S7-300 -sarjan ohjelmoitavien logiikoiden liittämisen MindSphereen. Tuettuja protokollia ovat myös OPC UA, Modbus ja Rockwell. (Siemens)

Siemensin S7-1500 ja S7-1200 -sarjan CPU:t voidaan yhdistää MindSphereen MQTT:llä TIA Portaliin lisättävällä *LMindConn_MQTT*-kirjastolla. Tässä tapauksessa brokerina toimii MindConnect IoT Extension, joka on MindSpherestä erikseen aktivoitava lisäpalvelu. (Siemens)

Yhteys MindSphereen on mahdollista toteuttaa myös MindSpheren yhteisön kehittämillä avoimen lähdekoodin työkaluilla ja kirjastoilla. Saatavilla on sekä NodeJS-kirjasto että Node-RED -node. (Siemens)

5.4 Muita pilvialustoja

Google Cloud on Googlen pilvialusta. Google Cloud tarjoaa IoT-ratkaisuja varten Cloud IoT Core -palvelun. Cloud IoT Core mahdollistaa laitteiden telemetria- ja tilatietojen lähettämisen pilveen ja ohjaamiseksi muihin palveluihin. Tuettuja yhteysprotokollia ovat MQTT ja HTTP. (Google)

Oracle tarjoaa IoT-ratkaisujen toteuttamiseen sekä valmiita SaaS-palveluja IoT Applications -palvelussaan että ekosysteemin omien ratkaisujen toteuttamiselle. Laitteita voidaan yhdistää MQTT:llä ja REST-rajapinnalla sekä eri ohjelmointikielillä. (Oracle)

IBM:n pilvialusta IBM Cloud tarjoaa IoT-ratkaisuja varten Watson IoT Platform -palvelua. Watson IoT Platform kokoaa joukon palveluja laitteiden yhdistämiseen, datan tallentamiseen ja analytiikkaan, ja tarjoaa ne kokonaisvaltaisena SaaS-ratkaisuna. Laitteita voidaan yhdistää MQTT:n välityksellä. (IBM)

6 INTEGRAATION TOTEUTTAMINEN

Suunnitelmana integraation suhteen oli toteuttaa koko ketju datalähteestä datan esittämiseen AOS:ssä. Datalähteeksi valittiin Siemens S7-1500 PLC ja pilvialustaksi Microsoft Azure. Pilvialustana oli aluksi tarkoitus käyttää Siemens MindSphereä, mutta kokeiluversion hankkimisessa ilmenneistä ongelmista johtuen se jouduttiin hylkäämään.

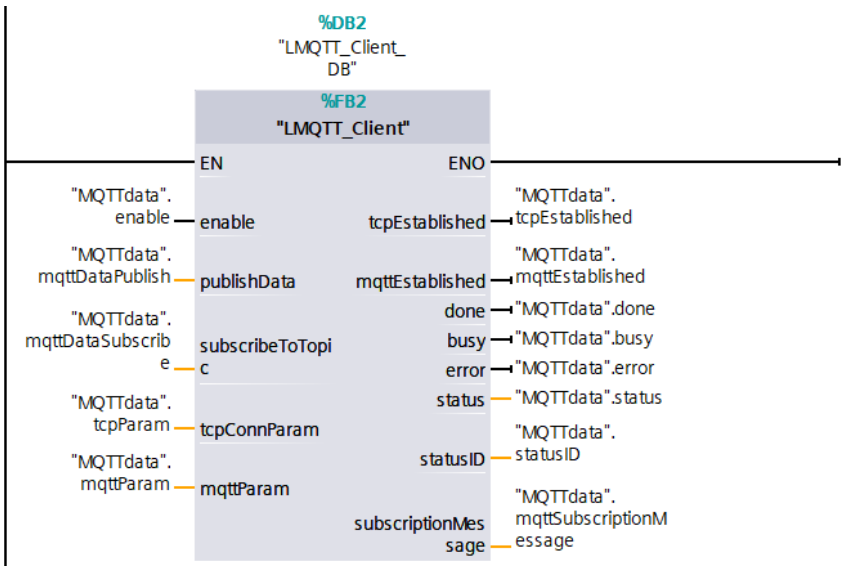
Dataa lähetettiin PLC:stä Azureen MQTT:n välityksellä ja datan hakeminen Azuresta toteutettiin REST-rajapinnan avulla. Lopuksi luotiin erilliset prototyypit React- ja A-Frame -komponenteista, jotka olisivat sellaisenaan integroitavissa AOS:ään.

6.1 Datan vieminen Azureen

6.1.1 Laitteen yhdistäminen Azureen

Laitteen yhdistämiseksi ja datan viemiseksi Azureen käytettiin Azuren IoT Hub -palvelua. Aluksi luotiin IoT Hub -tili, minkä jälkeen IoT Hubiin luotiin uusi PLC:tä kuvaava laite.

Siemens S7-1500 yhdistettiin Azureen käyttämällä Siemensin yleistä MQTT-kirjastoa, joka on ladattavissa Siemensin tukisivustolta. MQTT-kirjaston tuominen TIA Portal -projektiin tapahtui kirjaston dokumentaation osoittamalla tavalla. Parametreja varten luotiin data block, jonka arvot yhdistettiin function blockiin kuvan 9 mukaisesti.



KUVA 9. MQTT-client function block.

TLS-salauksen käyttäminen vaatii sertifikaatin asentamisen projektiin sekä CPU:hun. Tässä tapauksessa käytettiin Baltimore CyberTrust Root -sertifikaattia. CPU:n päivämäärä ja aika täytyi asettaa nykyhetkeen.

MQTT-clientin TCP-parametrit asetettiin kuvan 10 osoittamalla tavalla. Brokerin osoitteeksi asetettiin IoT Hubin osoite. MQTT-client asetettiin *useQdn*-parametrillä käyttämään verkkotunnusta IP-osoitteen sijaan, minkä takia CPU:n asetuksiin oli määritettävä reitittimen ja DNS-palvelimen IP-osoitteet.

MQTTdata

	Name	Data type	Start value
1	Static		
2	enable	Bool	true
3	tcpParam	"LMQTT_typeTcpCo.	
4	useQdn	Bool	true
5	hwIdentifier	HW_ANY	0
6	connectionID	CONN_OUC	16#0001
7	qdnAdressBroker	String[254]	'iothubjee123.azure-devices.net.'
8	ipAdressBroker	Array[0..3] of USInt	
9	localPort	UInt	0
10	mqttPort	UInt	8883
11	activateSecureConr	Bool	true
12	validateSubjectAlt...	Bool	false
13	idTlsServerCertifica.	UDInt	0
14	idTlsOwnCertificate	UDInt	0

KUVA 10. TCP-parametrit.

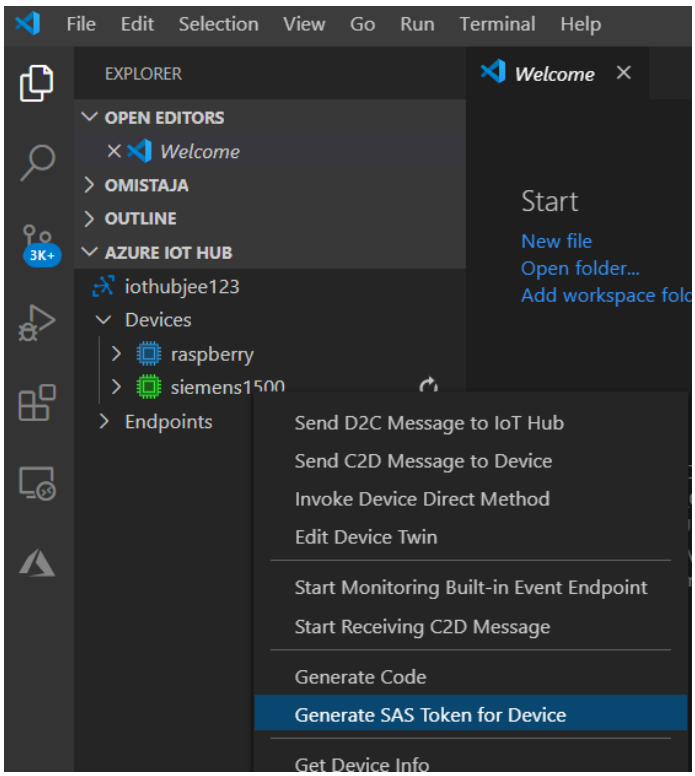
MQTT-parametrit asetettiin kuvan 11 mukaisesti. Laitteen tunnuksen on oltava sama kuin laitteen *deviceId* IoT Hubissa. Yhdistäminen Azureen vaatii käyttäjä-

tunnuksen ja salasanan. Käyttäjätunnus on muotoa *{iothubhostname}/{device_id}/?api-version=2018-06-30.* Salasana on SAS token, jonka voi generoida esimerkiksi Visual Studio Codessa kuvan 12 mukaisesti.

Parametriin *publishTopic* asetettiin IoT Hubin sisäiseen päätepisteeseen viittaava osoite. Jotta laitteen lähettämä viesti näkyy oikein Azuressa, topicin perään täytyi lisätä parametrit, jotka kertovat viestin olevat JSON-muotoista ja UTF-8 koodattua. Lähetettävä viesti kirjoitetaan *publishMessageData*-kohtaan ohjelmallisesti.

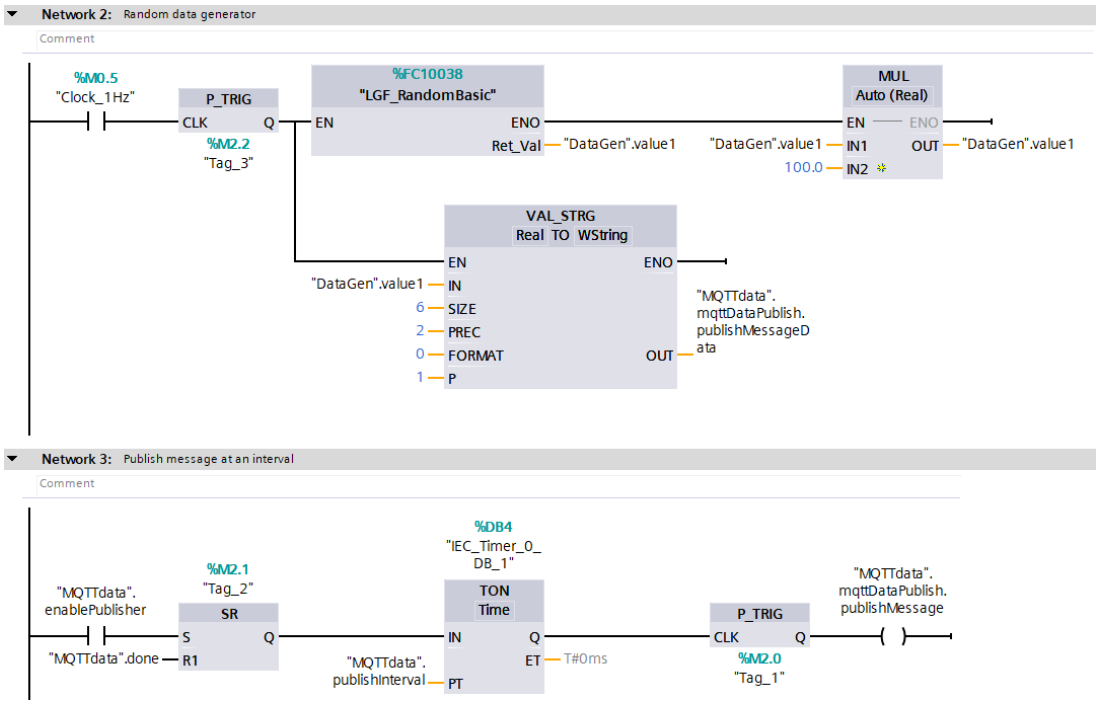
15	mqttParam	"LMQTT_typeParam.	
16	connectFlag	"LMQTT_typeConn...	
17	keepAlive	UInt	30
18	clientIdentifier	String[23]	'siemens1500'
19	willTopic	String[100]	''
20	willMessage	String[100]	''
21	userName	String[100]	'iothubjee123.azure-devices.net/siemens1500/?api-version=2018-06-30'
22	password	String[200]	'SharedAccessSignature sr=iothubjee123.azure-devices.net%2Fdevices%2Fsiemens1500&sig=
23	mqttDataPublish	"LMQTT_typePublis..	
24	publishMessage	Bool	false
25	publishTopic	WString[250]	WSTRING#'devices/siemens1500/messages/events/$$.ct=application%2Fjson&$$.ce=utf-8'
26	publishMessageDat	WString[1500]	WSTRING#''
27	publishQoS	Int	0
28	publishRetainFlag	Bool	false
29	mqttDataSubscribe	"LMQTT_typeSubsc..	
30	subscribeToTopic	Bool	false
31	unsubscribeFromT..	Bool	false
32	subscriptionTopic	WString[250]	WSTRING#'devices/siemens1500/messages/devicebound/#'
33	subscriptionQoS	Int	0
34	mqttSubscriptionMes...	"LMQTT_typeSubsc..	
35	newMessageRecie..	Bool	false
36	messageInvalid	Bool	false
37	receivingTopic	WString[400]	WSTRING#''
38	receivedMessageD.	WString[2400]	WSTRING#''

KUVA 11. MQTT-parametrit.



KUVA 12. SAS tokenin generoiminen Visual Studio Codessa.

Parametrien asettamisen jälkeen luotiin yksinkertainen ohjelma, joka lähettää tietyin aikavälein arvottuja lukuarvoja Azureen (kuva 13). Arvojen luomiseen käytettiin Library of general functions -kirjaston satunnaisarpojafunktiota. Viestien on oltava WString-muodossa, joten numerot muunnetaan käyttäen VAL_STRG-funktiota.

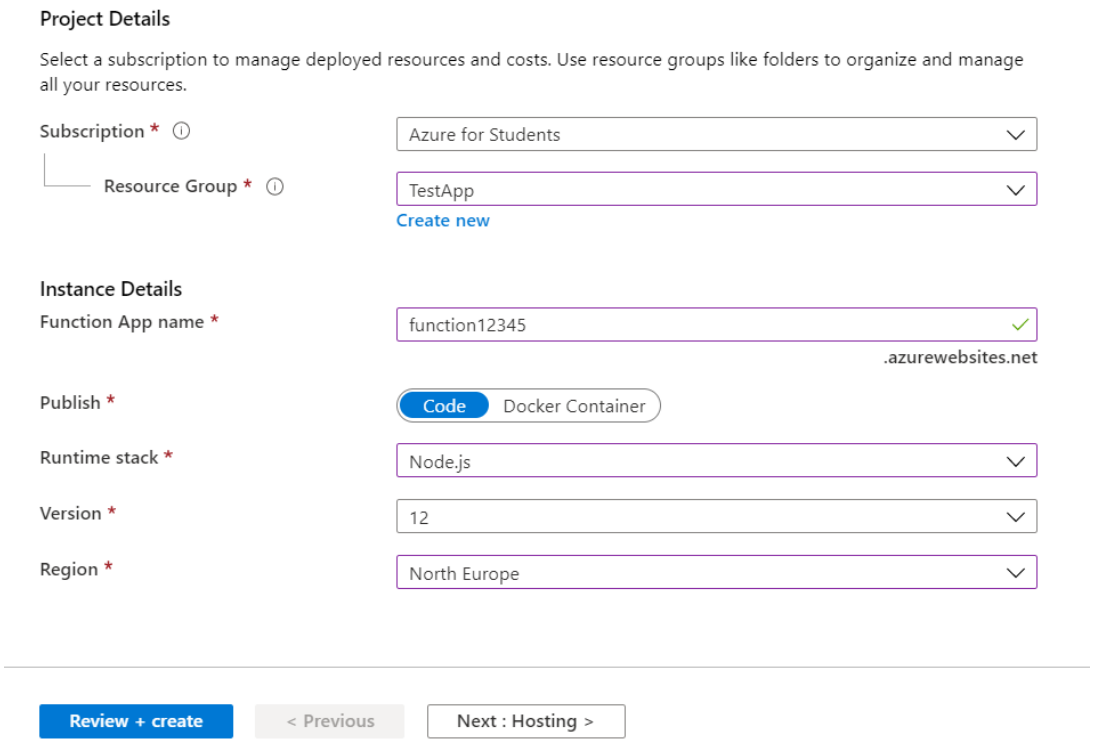


KUVA 13. Viestien generoiminen ja lähettäminen.

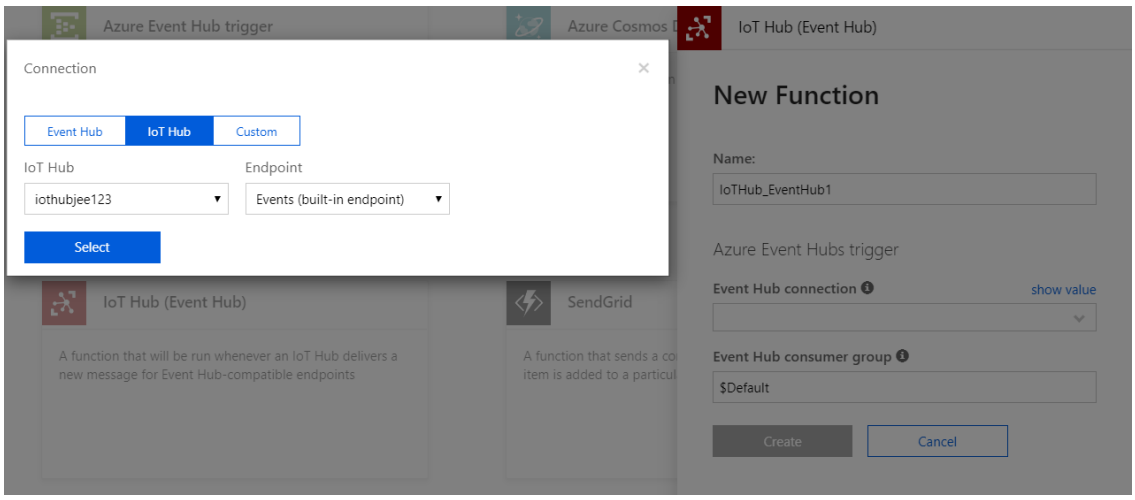
6.1.2 Datan ohjaaminen tietokantaan

Laitteen lähettämät viestit haluttiin ohjata tietokantaan. Tietokantana käytettiin Azuren Cosmos DB -palvelua ja Core SQL APIa. Tätä varten avattiin Cosmos DB -tili, jonne luotiin uusi tietokanta ja kokoelma. Kokoelman osiointiavaimeksi asetettiin laitteen tunnus.

Seuraavaksi täytyi luoda Azureen funktio, joka ohjaa viestit IoT Hubista tietokantaan. Tämä tapahtui luomalla uusi Function App kuvan 14 mukaisesti. Function Appiin luotiin uusi NodeJS-funktio käyttäen IoT Hub -templatea. Funktioon määritettiin yhteys IoT Hubin sisäänrakennettuun Events-päätepisteeseen (kuva 15).

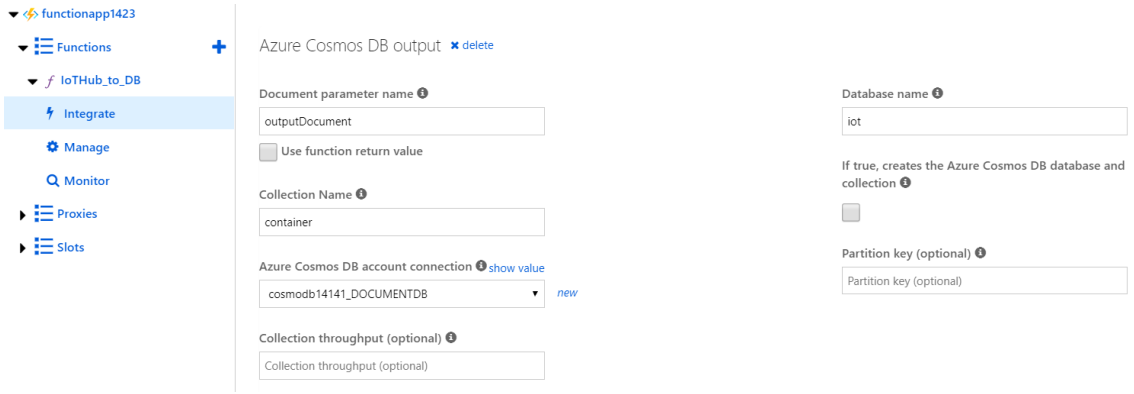


KUVA 14. Function Appin luominen.



KUVA 15. Funktion luominen ja yhdistäminen IoT Hubiin.

IoT Hub templatea käyttämällä saatiin funktio, joka ajetaan joka kerta, kun IoT Hubiin tulee uusi viesti laitteelta. Funktioon lisättiin kuvan 16 mukainen ulostulo tietokantaan.

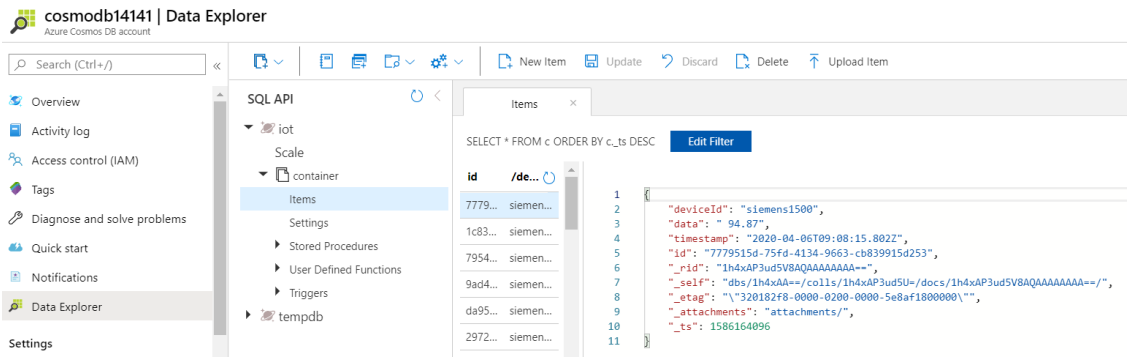


KUVA 16. Funktion yhdistäminen tietokantaan.

Seuraavaksi funktiota täydennettiin kuvan 17 mukaiseksi siten, että se kerää viestistä halutut tiedot, kokoaa ne JSON-objektiin, sekä lähettää objektin tietokantaan. Funktion toimivuus todennettiin tarkastelemalla tietokantaan syntyviä arvoja (kuva 18).

```
module.exports = function (context, IoTHubMessages) {
    context.log(`JavaScript eventhub trigger function called for message array: ${IoTHubMessages}`);

    var deviceId = "";
    var timestamp = "";
    var data = "";

    IoTHubMessages.forEach(message => {
        deviceId = context.bindingData.systemPropertiesArray[0]["iothub-connection-device-id"];
        timestamp = context.bindingData.systemPropertiesArray[0]["iothub-enqueuedtime"];
        data = message;

        context.log(`Time: ${timestamp}`);
        context.log(`DeviceId: ${deviceId}`);
        context.log(`Message: ${data}`);
    });

    var output = {
        "deviceId": deviceId,
        "data": data,
        "timestamp": timestamp,
    };

    context.bindings.outputDocument = output;

    context.done();
};
```

KUVA 17. Viestin käsittelevä funktio.



KUVA 18. Funktion lähettämät objektit tietokannassa.

6.2 Datan hakeminen Azuresta

Datan hakeminen toteutettiin Cosmos DB:n REST-rajapintaa käyttämällä. REST-pyyntöjen testaamiseen käytettiin Postman-ohjelmaa. Postmanin avulla luotiin kuvan 19 mukainen REST-pyyntö, joka hakee Cosmos DB:stä SQL-kyselyä käyttäen uusimman dokumentin. Headereihin *x-ms-date* ja *Authorization* tarvittavat arvot generoidaan pyynnön yhteydessä erillisessä scriptissä. Pyynnön luomisessa käytettiin apuna GitHub-käyttäjä MicrosoftCSA:n kokoelmaa *documentdb-postman-collection*.

Generated code for NodeJs - Request Contribute on GitHub Settings

```
var request = require('request');
var options = {
  'method': 'POST',
  'url': 'https://cosmodb14141.documents.azure.com:443/dbs/iot/colls/container/docs',
  'headers': {
    'Accept': 'application/json',
    'x-ms-version': '2018-12-31',
    'Authorization': '{{authToken}}',
    'x-ms-date': '{{RFC1123time}}',
    'x-ms-documentdb-isquery': 'true',
    'Content-Type': 'application/query+json',
    'x-ms-documentdb-query-enablecrosspartition': 'true'
  },
  body: "{\n    \"query\": \"SELECT TOP 1 * FROM c WHERE c.deviceId = @deviceId ORDER BY
      c._ts DESC\",\n    \"parameters\": [\n        {\n            \"name\":
      \"@deviceId\",\n            \"value\": \"siemens1500\"\n        }\n    ]\n}"

};
request(options, function (error, response) {
  if (error) throw new Error(error);
  console.log(response.body);
});
```

KUVA 19. Uusimman dokumentin hakeva REST-pyyntö.

6.3 Integraatio AOS:ään

AOS käyttää Reactia käyttöliittymäelementtien esittämiseen. Esimerkiksi POI-pisteen sivuvalikossa data esitettäisiin React-komponentissa. 360-ympäristö on toteutettu A-Frame -kirjastolla. Integraation tavoitteena oli saada pilvestä haettua dataa esitettyä sekä POI-pisteen sivuvalikossa että suoraan 360-kuvassa.

Toimintojen integroimiseksi AOS:ään luotiin erikseen prototyypit sekä React- että A-Frame -komponenteista, joita AOS:n ohjelmoijat voisivat käyttää pohjana lopullisessa toteutuksessa. Komponentit ohjelmoitiin siten, että ne sisältävät rajapinnan, jonka avulla halutut parametrit voidaan syöttää AOS:n käyttöliittymästä käsin tai tuoda muusta lähteestä. Komponentin käyttöönotto AOS:ssä tapahtuisi luomalla uusi POI-tyyppi, jota ladatessa kutsutaan kyseinen komponentti halutuilla parametreillä. Tällöin kullakin POI-pisteellä eli komponentin instanssilla olisi yksilöllinen tunnus.

Komponentteihin lisättiin rajapinnat seuraavien parametrien asettamiselle:

- Cosmos DB:n osoite ja master-avain
- Tietokannan ja kokoelman nimet sekä osiointiavain
- Laitteen tunnus
- Haettavan datan aikaväli
- Komponentin päivitystiheys

Komponenttien rajapinta on dokumentoitu liitteessä 1. Komponenttien ohjelmakoodit on esitetty kokonaisuudessaan liitteissä 2 ja 3.

6.3.1 React-komponentti

React-komponentin prototyyppi luotiin React/TypeScript-projektina käyttäen pohjana create-react-app -kirjastoa. Komponentti käyttää Postmanilla luotua REST-pyyntöä pohjana datan hakemiseksi. Osoite, josta REST-pyyntö lähetetään, täytyi lisätä Cosmos DB:n CORS-asetuksissa sallittuihin lähteisiin.

Komponentti tekee REST-pyynnön Axios-kirjastolla. Pyynnön yhteydessä luodaan *options*-objekti, joka sisältää tarvittavat tiedot pyynnön tekemiseksi, kuten osoitteen, headerit sekä lähetettävän SQL-kyselyn. Pyyntö toistetaan annetun päivitystiheyden välein, mikäli sellainen on annettu.

Pyynnön vastaus tallennetaan tilanhallintaan *responseData*-muuttujaan. Vastaus on lukujono, joka sisältää kaikki haetut dokumentit. Komponentti esittää haetun datan kuvan 20 mukaisesti. Kuvassa 21 on esitetty komponentin rajapinta sekä tilanhallinta.

Device ID	Value	UTC Time
siemens1500	72.07	2020-04-14T18:21:44.672Z
siemens1500	39.76	2020-04-14T18:20:44.654Z
siemens1500	1.42	2020-04-14T18:19:44.649Z
siemens1500	43.41	2020-04-14T18:18:44.643Z
siemens1500	15.23	2020-04-14T18:17:44.645Z
siemens1500	72.21	2020-04-14T18:16:44.642Z

KUVA 20. React-komponentin esittämä data.

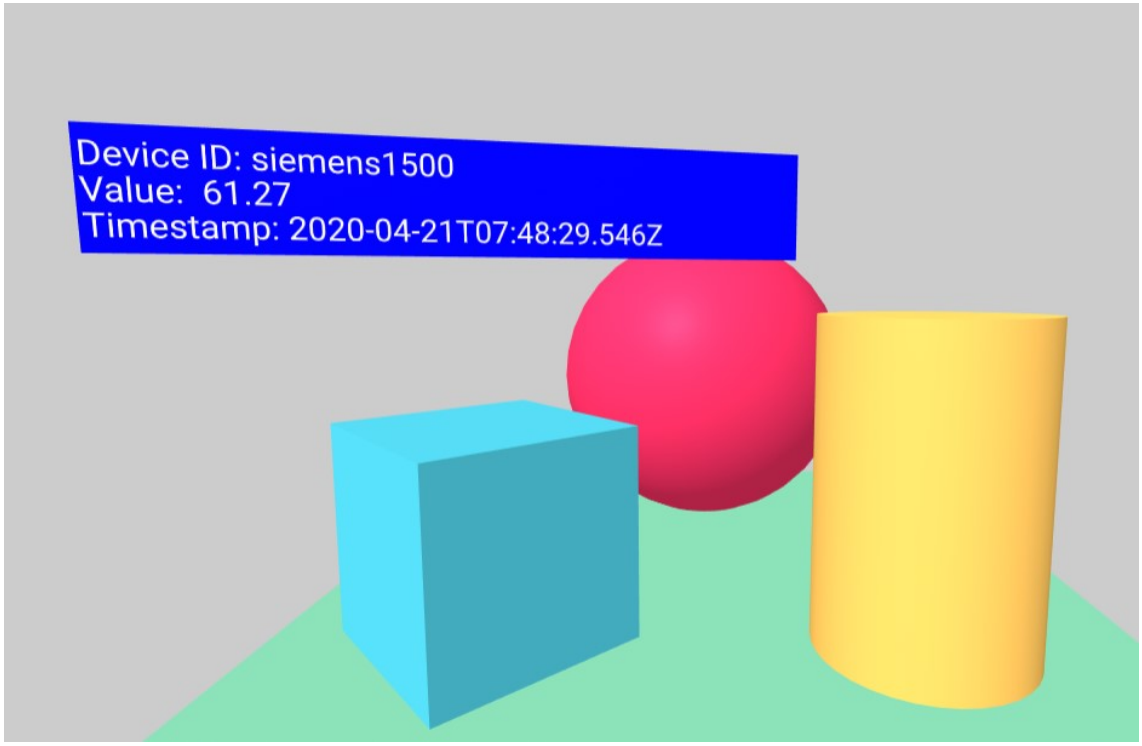
```
interface params {
  host: string, //database host uri, found in cosmosdb->settings->keys
  masterKey: string, //database master key, found in cosmosdb->settings->keys
  dbName: string, //database name
  collName: string, //collection name
  deviceId: string, //deviceId (only one supported)
  partitionKey: string, //collection partition key
  timeFrame: Number, //timeframe of data to be displayed, seconds
  interval: Number, //update interval, seconds
}
this.state = {
  responseData: [],
  isLoading: true,
  errors: null,
}
```

KUVA 21. React-komponentin rajapinta ja tilanhallinta.

6.3.2 A-Frame -komponentti

AOS käyttää 360-ympäristön esittämiseen A-Framea. A-Frame on avoimen lähdekoodin kirjasto, jonka avulla voidaan toteuttaa virtuaaliympäristöjä (A-Frame). Dynaamisen datan esittämiseksi suoraan 360-kuvassa oli luotava A-Frame -komponentti.

A-Frame -komponentti luotiin aiemmin luodun React/TypeScript-komponentin pohjalta. Komponentti hakee annettujen parametrien perusteella viimeisimmän dokumentin ja palauttaa 3D-ympäristön, jossa data esitetään kuvan 22 mukaisesti. Komponentti näyttää vain viimeisimmän dokumentin datan.

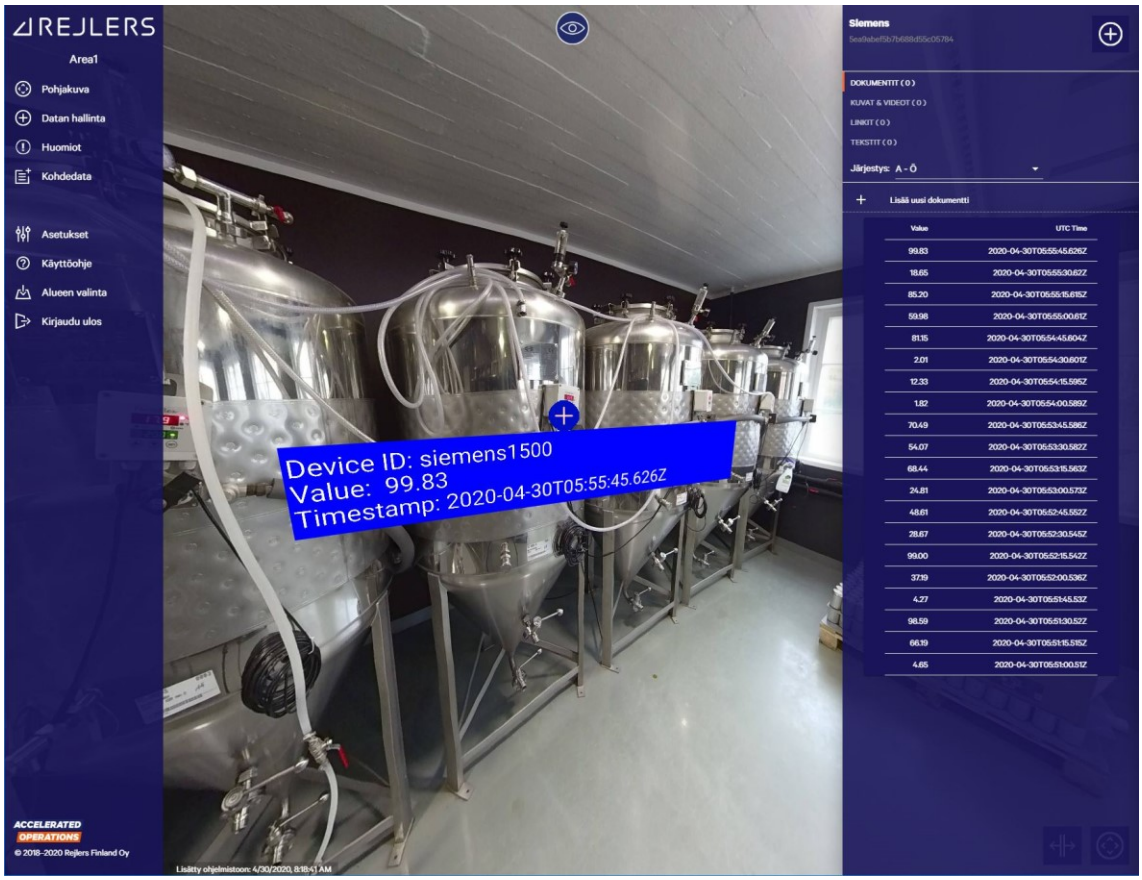


KUVA 22. Datan esittäminen A-Frame -komponentin 3D-maailmassa.

7 POHDINTA

7.1 Tulokset ja niiden arviointi

Opinnäytetyön tuloksena syntyi prototyypit käyttöliittymäkomponenteista, joita soveltamalla dynaamisen datan esittäminen AOS:n käyttöliittymässä voidaan toteuttaa, sekä dokumentaatio niiden rajapinnasta. Opinnäytetyössä käytiin myös läpi prosessi ohjelmoitavan logiikan yhdistämiseksi Azureen sekä esiteltiin tapa, jolla Azuren palveluja voidaan hyödyntää tämänlaiseen käyttötarkoitukseen. Komponentteja saatiin Rejlersin ohjelmoijien avustuksella kokeiltua sellaisenaan myös AOS:ssä (kuva 23).



KUVA 23. React- ja A-Frame -komponentit AOS:ssä.

Opinnäytetyölle asetettu tavoite esittää dynaamista dataa AOS:ssä saatiin täytettyä. Komponenttien laajempi hyödyntäminen, kuten parametrien asettaminen käyttöliittymästä, POI-tyyppien määrittely ja datan sujuva esitystapa vaatii kuitenkin jatkokehitystä. Reaalielämän sovelluksina komponenteilla voitaisiin esittää

esimerkiksi valetun betonin kosteuden tilaa, kiinteistöjen ilmanvaihto-ongelmia tai laitteiden hälytystiloja.

Dynaamisen datan esittäminen AOS:ssä tuo monia hyötyjä ja mahdollisuuksia. AOS kokoaa tilaan tai projektiin liittyvän datan eri lähteistä ja esittää ne visuaalisessa käyttöliittymässä, jolloin ajantasainen tieto on helposti saatavilla mukana oleville eri sidosryhmille. Automaatiojärjestelmien datan integroiminen AOS:ään toiminnanohjaus- ja muiden järjestelmien datan lisäksi parantaisi entisestään kokonaiskuvaa.

Datan visuaalinen esitystapa hyödyttäisi esimerkiksi kunnossapitoa. Laitteen lähettämä hälytys voisi näkyä 360-kuvassa POI-pisteen muuttuneena värinä tai muotona, jolloin laitteen ulkonäkö ja sijainti ovat heti selvillä. Yhdistettynä POI-pisteen sisältämään muuhun dataan, kuten valokuviin ja erilaisiin dokumentteihin, kunnossapitohenkilöstön tarvitsema tieto on saatavilla yhdessä paikassa eikä sen etsimiseen tarvitse kuluttaa aikaa.

7.2 Kehitysmahdollisuuksia

Azuren palvelujen käyttöä voisi hyödyntää tehokkaammin joiltakin osin. IoT Hubin ilmaistilissä on rajoituksena 8000 viestiä päivässä, mikä voi koitua ongelmaksi, mikäli useampi laite lähettää jatkuvasti dataa. Useamman laitteen lyhyellä aikavälillä lähettämät viestit voisi välittää IoT Hubiin kootusti yhtenä viestinä jonkinlaisen välityspalvelimen kautta. IoT Hub mahdollistaa myös kaksisuuntaisen kommunikaation, jonka avulla laitteeseen voisi lähettää viestejä pilvestä tai AOS:stä käsin. Viesteillä voisi esimerkiksi käskeä laitetta suorittamaan jokin ennalta ohjelmoitu toiminto.

Laitteiden määrän kasvaessa myös tietokannan toimintaa voisi tehostaa suunnittelemalla datamallia tarkemmin, luomalla dokumentteja esimerkiksi päivämäärän mukaan ja hyödyntämällä dokumenttirakenteen sisäkkäisyyttä. Tietokannan tietoturvaa voisi parantaa käyttämällä datan hakemiseen master keyn sijaan resurssikohtaisia avaimia. Myös erilaisten tietokantatyyppien soveltuvuutta voisi arvioida tarkemmin.

7.3 Loppupäätelmät

Laitteiden yhdistettävyys näyttäisi olevan hyvällä tasolla. Monet pilvialustat ja laitteet tukevat vähintään MQTT-protokollaa, mikä mahdollistaa erilaisten laite- ja pilvialustayhdistelmien käyttämisen. Logiikkatoimittajan oman alustan käyttäminen voi tehdä laitteiden yhdistämisen suoraviivaisemmaksi abstraktoimalla alla olevaa protokollaa. Lisäksi esimerkiksi Siemensin MindConnect-laitteet mahdollistavat joidenkin sellaisten laitteiden yhdistettävyyden, jotka eivät sellaisenaan tue tarvittavia yhteysprotokollia.

Opinnäytetyöstä saadun kokemuksen perusteella tarvittava teknologia ja infrastruktuuri erilaisten teollisen internetin ratkaisujen toteuttamiseen on helposti saatavilla pilvialustoissa. Osaamista tarvitaan toteutuksen tarpeellisuuden ja taloudellisen kannattavuuden arviointiin sekä sopivimpien palvelujen valitsemiseen ja niiden tehokkaaseen hyödyntämiseen.

LÄHTEET

A-Frame. Introduction. Luettu 15.4.2020.
https://aframe.io/docs/1.0.0/introduction/

Amazon Web Services. About AWS. Luettu 14.4.2020.
https://aws.amazon.com/about-aws/

Amazon Web Services. AWS IoT Core features. Luettu 14.4.2020.
https://aws.amazon.com/iot-core/features/

Amazon Web Services. Cloud computing models. Luettu 27.2.2020.
https://aws.amazon.com/types-of-cloud-computing/

Amazon Web Services. Databases on AWS. Luettu 14.4.2020.
https://aws.amazon.com/products/databases/

Amazon Web Services. Protocols. Luettu 14.4.2020.
https://docs.aws.amazon.com/iot/latest/developerguide/protocols.html

Collin, J. & Saarelainen, A. (2016) Teollinen internet. Helsinki: Talentum.

Datatiede. Analytiikan tasot. Luettu 4.3.2020.
https://www.datatiede.fi/analytiikan-tasot/

Google Cloud. Cloud IoT Core overview. Luettu 15.4.2020.
https://cloud.google.com/iot/docs/concepts/overview

Hackernoon. RabbitMQ, AMQP, MQTT & Rest of the world. Luettu 14.4.2020.
https://hackernoon.com/rabbitmq-amqp-mqtt-rest-of-the-world-74433c5ff8c7

IBM. IBM Watson IoT Platform. Product Overview. Luettu 15.4.2020.
https://www.ibm.com/support/knowledgecenter/SSQP8H/iot/overview/overview.html

Javascript.info. WebSocket. Luettu 14.4.2020.
https://javascript.info/websocket

Microsoft Azure. Azure Cosmos DB. Luettu 14.4.2020.
https://azure.microsoft.com/en-us/services/cosmos-db/

Microsoft Azure. Azure IoT Hub. Luettu 14.4.2020.
https://azure.microsoft.com/en-us/services/iot-hub/

Microsoft Azure. Azure IoT Hub developer guide. Luettu 14.4.2020.
https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-protocols

Microsoft Azure. Azure SQL Database. Luettu 14.4.2020.
https://azure.microsoft.com/en-us/services/sql-database/

Microsoft Azure. What is Azure? Luettu 14.4.2020.

https://azure.microsoft.com/en-us/overview/what-is-azure/

Microsoft Azure. What is Azure IoT Central? 14.4.2020.
https://docs.microsoft.com/en-us/azure/iot-central/core/overview-iot-central

Microsoft Azure. What is PaaS? Luettu 2.3.2020.
https://azure.microsoft.com/en-us/overview/what-is-paas/

Mosquitto.org. Man. Luettu 24.2.2020.
https://mosquitto.org/man/mqtt-7.html

MQTT.org. Faq. Luettu 24.2.2020.
http://mqtt.org/faq

Nodered.org. About. Luettu 5.3.2020.
https://nodered.org/about/

Nodered.org. Node-RED Library. Luettu 14.4.2020.
https://flows.nodered.org/

OPC Foundation. OPC Unified Architecture. Luettu 10.2.2020.
https://opcfoundation.org/about/opc-technologies/opc-ua/

OPC-Router. What is MQTT? Luettu 25.2.2020.
https://www.opc-router.com/what-is-mqtt/

Oracle. Database. Luettu 16.4.2020.
https://www.oracle.com/database/what-is-database.html

Oracle. Internet of Things. Luettu 15.4.2020.
https://www.oracle.com/internet-of-things/

Oracle. Oracle Internet of Things Cloud Service. Get Started. Luettu 15.4.2020.
https://docs.oracle.com/en/cloud/paas/iot-cloud/index.html

Oracle. What is a cloud database? Luettu 16.4.2020.
https://www.oracle.com/database/what-is-a-cloud-database/

Oracle. What is IoT? Luettu 16.2.2020.
https://www.oracle.com/internet-of-things/what-is-iot.html

Rejlers. Accelerated-ohjelma. Luettu 12.3.2020.
https://www.rejlers.fi/Toimialat_ja_palvelut/ICT/accelerated-ohjelma/

Rejlers. AOS User Guide. Luettu 16.3.2020.
https://aos.rejlers.fi/documents/aos_user_guide.pdf

Rouse, M. 2018. Advanced Message Queuing Protocol. Luettu 14.4.2020.
https://whatis.techtarget.com/definition/Advanced-Message-Queuing-Protocol-AMQP

Rouse, M. 2018. IoT security. TechTarget. Luettu 13.3.2020.

https://internetofthingsagenda.techtarget.com/definition/IoT-security-Internet-of-Things-security

Rouse, M. 2019. REST (Representational State Transfer). TechTarget. Luettu 12.3.2020. https://searchapparchitecture.techtarget.com/definition/REST-REpresentational-State-Transfer

Rouse, M. 2019. RESTful API. TechTarget. Luettu 12.3.2020. https://searchapparchitecture.techtarget.com/definition/RESTful-API

Siemens. Digital Twin. Luettu 10.3.2020. https://www.plm.automation.siemens.com/global/en/our-story/glossary/digital-twin/24465

Siemens. Getting Connected to MindSphere. Luettu 11.3.2020. https://documentation.mindsphere.io/resources/pdf/getting-connected-en.pdf

Siemens. How MindSphere works. Luettu 9.3.2020. https://siemens.mindsphere.io/en/about

Siemens. LMindConn_MQTT. Luettu 25.2.2020. https://cache.industry.siemens.com/dl/files/284/109772284/att_1010248/v1/109772284_LMindConn_MQTT_DOKU_1-0_en.pdf

Siemens. MindConnect Nano and MindConnect IoT2040. Luettu 26.2.2020. https://www.plm.automation.siemens.com/media/store/en_in/Siemens-MIndSphere-MindConnect-Nano-and-MindConnect-IoT2040-fs-66300-A14_tcm76-17688.pdf

Siemens. MindSphere Open Source Tools and Libraries. Luettu 5.3.2020. https://opensource.mindsphere.io/index.html

Wright, I. 2017. What Can Augmented Reality Do for Manufacturing? Engineering.com. Luettu 13.3.2020. https://www.engineering.com/AdvancedManufacturing/ArticleID/14904/What-Can-Augmented-Reality-Do-for-Manufacturing.aspx

LIITTEET

Liite 1. Komponenttien rajapinta

Nimi	Tyyppi	Kuvaus
host	string	Cosmos DB Host name
masterKey	string	Cosmos DB Master key
dbName	string	Tietokannan nimi
collName	string	Kokoelman nimi
deviceId	string	Laitteen tunnus sellaisena kuin se on tietokantaan tallennettu. Vain yksi laite.
partitionKey	string	Kokoelman osiointiavain. Jos ei ole, niin haku tehdään cross partition querynä.
timeFrame	Number	Aikaväli nykyhetkestä taaksepäin, jolta dataa haetaan sekunteina. Null- tai 0-arvolla haetaan viimeisin dokumentti. (A-Frame-komponentissa tätä ei käytetä, vaan viimeisin haetaan aina)
interval	Number	Aikaväli datan uudelleenhakemiseen sekunteina. Null- tai 0-arvolla pyyntö tehdään ainoastaan komponentin ladatessa.

Esimerkit komponenttien kutsumisesta:

```
<ReactComponent
  host={'https://cosmosdbaccount.documents.azure.com:443/'}
  masterKey={'masterkey12345'}
  dbName={'mydatabase'}
  collName={'mycontainer'}
  deviceId={'mydevice'}
  partitionKey={'/partitionkey'}
  timeFrame={5*60}
  interval={20}
/>
```

```
<AFrameComponent
  host={'https://cosmosdbaccount.documents.azure.com:443/'}
  masterKey={'masterkey12345'}
  dbName={'mydatabase'}
  collName={'mycontainer'}
  deviceId={'mydevice'}
  partitionKey={'/partitionkey'}
  interval={15}
/>
```

Liite 2. ReactComponent.tsx


```
import React, { Component } from 'react'
import axios from 'axios'
import CryptoJS from 'crypto-js'
import TableContainer from '@material-ui/core/TableContainer';
import TableHead from '@material-ui/core/TableHead';
import Paper from '@material-ui/core/Paper';
import Table from '@material-ui/core/Table';
import TableBody from '@material-ui/core/TableBody'
import TableCell from '@material-ui/core/TableCell';
import TableRow from '@material-ui/core/TableRow';
import Container from '@material-ui/core/Container';

class ReactComponent extends Component<any, any> {
  constructor(params: any) {
    super(params);

    interface params {
      host: string, //database host uri, found in cosmosdb->settings-
>keys
      masterKey: string, //database master key, found in cosmosdb-
>settings->keys
      dbName: string, //database name
      collName: string, //collection name
      deviceId: string, //deviceId (only one supported)
      partitionKey: string, //collection partition key
      timeFrame: Number, //timeframe of data to be displayed, seconds
      interval: Number, //update interval, seconds
    }
    this.state = {
      responseData: [],
      isLoading: true,
      errors: null,
    }
  }
  interval: number | undefined;

  setOptions() {
    //here we set the options for the axios request
    var options: any = {
      method: 'POST',
      url:
`${this.props.host}dbs/${this.props.dbName}/colls/${this.props.collNam
e}/docs`,
      headers: {
        'Accept': 'application/json',
        'x-ms-version': '2018-12-31',
        'Authorization': '',
        'x-ms-date': '',
        'x-ms-documentdb-isquery': 'true',
        'Content-Type': 'application/query+json',
        'x-ms-documentdb-query-enablecrosspartition': 'false',
        'x-ms-partition-key': ''
      },
      data: ""
    }
```

```
//set partition key for the request if supplied, else enable cross
partition query
    if (this.props.partitionKey.trim()) {
      options.headers['x-ms-partition-key'] =
[`${this.props.partitionKey}`]
    } else {
      options.headers['x-ms-documentdb-query-enablecrosspartition'] =
'true'
    }

    //set the sql query according to parameters and update request
options
    var query = ``
    var queryParams = ``
    if (this.props.timeFrame > 0) {
      query = `
      "SELECT * FROM c
      WHERE (c.deviceId = @deviceId
      AND (c._ts BETWEEN @fromTime AND @toTime))
      ORDER BY c._ts DESC"
      `
      queryParams = `[
        { "name": "@deviceId", "value": "${this.props.deviceId}" },
        { "name": "@fromTime", "value": ${Date.now() / 1000 -
this.props.timeFrame} },
        { "name": "@toTime", "value": ${Date.now() / 1000} }
      ]`
      console.log(`Querying documents from last
${this.props.timeFrame} seconds`)
    } else {
      query = `
              "SELECT TOP 1 * FROM c
              WHERE c.deviceId = @deviceId
              ORDER BY c._ts DESC"
              `
      queryParams = `[
                { "name": "@deviceId", "value":
"${this.props.deviceId}" }
              ]`
      console.log(`Querying most recent document only`)
    }
    var requestBody = `{
      "query": ${query},
      "parameters": ${queryParams}
     }`
    options.data = requestBody

    // generate x-ms-date and Authorization headers for the request
    // taken from MicrosoftCSA documentdb-postman-collection
    var mastKey = this.props.masterKey;
    var today = new Date();
    var UTCstring = today.toUTCString();
    var url = options.url.trim();
    var strippedurl = url.replace(new RegExp('^https?://[^/]+/'),
'/');
    var strippedparts = strippedurl.split("/");
    var truestrippedcount = (strippedparts.length - 1);
    var resourceId = "";
    var resType = "";
    if (truestrippedcount % 2) {
      resType = strippedparts[truestrippedcount];
      if (truestrippedcount > 1) {
```

3 (4)

```
   var lastPart = strippedurl.lastIndexOf("/");
        resourceId = strippedurl.substring(1, lastPart);
      }
    }
    else {
      resType = strippedparts[truestrippedcount - 1];
      strippedurl = strippedurl.substring(1);
      resourceId = strippedurl;
    }
    var verb = options.method.toLowerCase();
    var date = UTCstring.toLowerCase();
    var key = CryptoJS.enc.Base64.parse(mastKey);
    var text = (verb || "").toLowerCase() + "\n" +
      (resType || "").toLowerCase() + "\n" +
      (resourceId || "") + "\n" +
      (date || "").toLowerCase() + "\n\n";
    var signature = CryptoJS.HmacSHA256(text, key);
    var base64Bits = CryptoJS.enc.Base64.stringify(signature);
    var MasterToken = "master";
    var TokenVersion = "1.0";
    var auth = encodeURIComponent("type=" + MasterToken + "&ver=" +
TokenVersion + "&sig=" + base64Bits);
    options.headers.Authorization = auth;
    options.headers["x-ms-date"] = UTCstring;

    console.log(options)
    return (options)
  }

  makeRequest() {
    this.setState({ isLoading: true })

    //make the request using the output of setOptions() as config
    axios(this.setOptions())
      .then(response => {
        this.setState({
          responseData: response.data, //save response to state
          isLoading: false //loading is done
        })
        console.log(this.state.responseData);
      })
      .catch(error => {
        this.setState({
          error,
          isLoading: false
        })
        console.log(error);
      })
  }

  componentDidMount() {
    // when component mounts -> make request
    this.makeRequest()
    // repeat if interval is set
    if (this.props.interval > 0) {
      this.interval = window.setInterval(() => this.makeRequest(),
1000 * this.props.interval);
    }
  }
```

```
componentWillUnmount() {
    clearInterval(this.interval);
  }

  render() {
    if (this.state.isLoading) {
      return (
        <p>Loading...</p>
      )
    }
    return (
      <Container maxWidth="sm">
        <TableContainer component={Paper} >
          <Table className='respone_container' size='small' aria-
label="a dense table">
            <TableHead>
              <TableRow>
                <TableCell align="right">Device ID</TableCell>
                <TableCell align="right">Value</TableCell>
                <TableCell align="right">UTC Time</TableCell>
              </TableRow>
            </TableHead>
            <TableBody>
              {this.state.responseData.Documents?.map((row: { id:
string | number | undefined; deviceId: React.ReactNode; value:
React.ReactNode; timestamp: React.ReactNode }) => (
                <TableRow key={row.id}>
                  <TableCell align="right">{row?.deviceId}</TableCell>
                  <TableCell align="right">{row?.value}</TableCell>
                  <TableCell
align="right">{row?.timestamp}</TableCell>
                </TableRow>))}
            </TableBody>
          </Table>
        </TableContainer>
      </Container>
    )
  }
}

export default ReactComponent
```

Liite 3. AFrameComponent.tsx


```
import 'aframe';
import { Entity, Scene } from 'aframe-react';
import React, { Component } from 'react';
import axios from 'axios'
import CryptoJS from 'crypto-js'

class AFrameComponent extends Component<any, any> {
  constructor(params: any) {
    super(params);

    interface params {
      host: string, //database host uri, found in cosmosdb->settings->keys
      masterKey: string, //database master key, found in cosmosdb->settings->keys
      dbName: string, //database name
      collName: string, //collection name
      deviceId: string, //deviceId (only one supported)
      partitionKey: string, //collection partition key
      interval: Number //update interval in seconds
    }
    this.state = {
      responseData: [],
      isLoading: true,
      errors: null,
      text: ''
    }
  }
  interval: number | undefined;

  setOptions() {
    //here we set the options for the axios request
    var options: any = {
      method: 'POST',
      url: `${this.props.host}dbs/${this.props.dbName}/colls/${this.props.collName}/docs`,
      headers: {
        'Accept': 'application/json',
        'x-ms-version': '2018-12-31',
        'Authorization': '',
        'x-ms-date': '',
        'x-ms-documentdb-isquery': 'true',
        'Content-Type': 'application/query+json',
        'x-ms-documentdb-query-enablecrosspartition': 'false',
        'x-ms-partition-key': ''
      },
      data: ""
    }
    //set partition key for the request if supplied, else enable cross partition query
    if (this.props.partitionKey.trim()) {
      options.headers['x-ms-partition-key'] = [`${this.props.partitionKey}`]
    } else {
      options.headers['x-ms-documentdb-query-enablecrosspartition'] = 'true'
    }
```

2 (4)

```
//update request body with given parameters
    var  query = `
              "SELECT TOP 1 * FROM c
              WHERE c.deviceId = @deviceId
              ORDER BY c._ts DESC"
              `
    var  queryParams = `[
                { "name": "@deviceId", "value":
"${this.props.deviceId}" }
              ]`
    var requestBody = `{
      "query": ${query},
      "parameters": ${queryParams}
     }`
    options.data = requestBody

    // this script generates x-ms-date and Authorization headers for
the request
    // from MicrosoftCSA documentdb-postman-collection
    var mastKey = this.props.masterKey;
    var today = new Date();
    var UTCstring = today.toUTCString();
    var url = options.url.trim();
    var strippedurl = url.replace(new RegExp('^https?://[^/]+/'),
'/');
    var strippedparts = strippedurl.split("/");
    var truestrippedcount = (strippedparts.length - 1);
    var resourceId = "";
    var resType = "";
    if (truestrippedcount % 2) {
      resType = strippedparts[truestrippedcount];
      if (truestrippedcount > 1) {
        var lastPart = strippedurl.lastIndexOf("/");
        resourceId = strippedurl.substring(1, lastPart);
      }
    }
    else {
      resType = strippedparts[truestrippedcount - 1];
      strippedurl = strippedurl.substring(1);
      resourceId = strippedurl;
    }
    var verb = options.method.toLowerCase();
    var date = UTCstring.toLowerCase();
    var key = CryptoJS.enc.Base64.parse(mastKey);
    var text = (verb || "").toLowerCase() + "\n" +
      (resType || "").toLowerCase() + "\n" +
      (resourceId || "") + "\n" +
      (date || "").toLowerCase() + "\n\n";
    var signature = CryptoJS.HmacSHA256(text, key);
    var base64Bits = CryptoJS.enc.Base64.stringify(signature);
    var MasterToken = "master";
    var TokenVersion = "1.0";
    var auth = encodeURIComponent("type=" + MasterToken + "&ver=" +
TokenVersion + "&sig=" + base64Bits);
    options.headers.Authorization = auth;
    options.headers["x-ms-date"] = UTCstring;

    console.log(options)
    return (options)
  }
```

3 (4)

```
makeRequest() {
    this.setState({
      isLoading: true,
      text: 'Loading...'
    })

    //make the request using the output of setOptions() as config
    axios(this.setOptions())
      .then(response => {
        this.setState({
          responseData: response.data, //save response to state
          isLoading: false //loading is done
        })
        console.log(this.state.responseData);

        var text = `Device ID:
${this.state.responseData.Documents[0].deviceId}
        Value: ${this.state.responseData.Documents[0].value}
        Timestamp: ${this.state.responseData.Documents[0].timestamp}`

        this.setState({ text: text })
      })
      .catch(error => {
        this.setState({
          error,
          isLoading: false
        })
        console.log(error);
      })
  }

  componentDidMount() {
    // when component mounts -> make request
    this.makeRequest()
    // repeat if interval is set
    if (this.props.interval > 0) {
      this.interval = window.setInterval(() => this.makeRequest(),
1000 * this.props.interval);
    }
  }

  componentWillUnmount() {
    clearInterval(this.interval);
  }
```

4 (4)

```
render() {
    return (
      <Scene background="color: #FAFAFA">
        <Entity primitive="a-box" position="-1 0.5 -3" rotation="0 45
0" color="#4CC3D9" shadow/>
        <Entity primitive="a-sphere" position="0 1 -5" radius="1"
color="#EF2D5E" shadow/>
        <Entity primitive="a-cylinder" position="1 0.75 -3"
radius="0.5" height="1.5" color="#FFC65D" shadow/>
        <Entity primitive="a-plane" position="0 0 -4" rotation="-90 0
0" width="4" height="4" color="#7BC8A4" shadow/>
        <Entity textbox
          id='textbox'
          geometry={{ primitive: 'plane', height: 0.5, width: 3 }}
          material={{ color: 'blue' }}
          position={{ x: -1, y: 2, z: -3 }}
          rotation={{ x: 0, y: 15, z: 0 }}
          text={{ width: 3, value: this.state.text }}>
        </Entity>
        <Entity primitive="a-sky" material="color: #ccc" />
      </Scene>
    );
  }
}

export default AFrameComponent;
```