Toni Hämeenniemi

Teollisen internetin esimerkkisovellus

Opinnäytetyö
Kevät 2017
SeAMK Tekniikka
Automaatiotekniikka, sähköautomaatio

SeAMK
SEINÄJOEN AMMATTIKORKEAKOULU
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

SEINÄJOEN AMMATTIKORKEAKOULU

Opinnäytetyön tiivistelmä

Koulutusyksikkö: Tekniikka

Tutkinto-ohjelma: Automaatiotekniikka

Suuntautumisvaihtoehto: Sähköautomaatio

Tekijä: Toni Hämeenniemi

Työn nimi: Teollisen internetin esimerkkisovellus

Ohjaaja: Petteri Mäkelä

Vuosi: 2017 Sivumäärä: 51 Liitteiden lukumäärä: 0

Seinäjoen Ammattikorkeakoululla oli tarve järjestelmälle, joka mahdollistaa teollisen internetin laboratorion laitteiston sisältämän prosessidatan säilömisen ja reaaliaikaisen tai myöhemmän analysoinnin sekä hyödyntämisen muissa sovelluksissa. Tämä opinnäytetyö käsittää tarpeiden mukaisen, yleiskäyttöisen järjestelmän suunnittelu- sekä toteutusosion.

Opinnäytetyössä käydään läpi aiheeseen liittyvä historia, teoria, sekä esimerkkisovelluksen toteutus, rakenne ja työn lopputulokset. Eniten työssä keskitytään tiedonsiirtoprotokolliin, dokumenttipohjaisiin tietokantoihin, rajapintapalveluihin ja web-käyttöliittymäsovellusten toteuttamiseen liittyviin teknologioihin.

Työn tuloksena Seinäjoen Ammattikorkeakoululle luotiin esimerkkisovellus, jonka avulla teollisen internetin laboratorion sisäistä prosessidataa voidaan tallentaa dokumenttipohjaiseen tietokantaan aikaleimoilla varustettuna. Dataa voidaan myös lukea ja muokata erillisen rajapintasovelluksen kautta, joka mahdollistaa työn tulosten laajan hyödyntämisen muissa tulevaisuuden opiskelijaprojekteissa ja harjoitustöissä.

Avainsanat: esineiden internet, protokollat, ohjelmointi, rajapinnat, ohjelmoitava logiikka

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Thesis abstract

Faculty: School of Technology

Degree programme: Automation Engineering

Specialisation: Electric Automation

Author: Toni Hämeenniemi

Title of thesis: Industrial internet example application

Supervisor: Petteri Mäkelä

Year: 2017 Number of pages: 51 Number of appendices: 0

Seinäjoki University of Applied Sciences had a need for a system that would make it possible to gather and analyze the internal process data created by the industrial internet laboratory of SeAMK. This thesis contains both the design and the creation of such a generic system.

This thesis goes through the history and theory behind the subject and the related technologies. Also the whole design and implementation process of the example application is explained. The theory chapters mainly focus on data transfer protocols, document based databases, application interface services and the technologies that lie behind implementing an interactive web application on top of this software stack.

As the result of this thesis, an example application was made for SeAMK that can be used to save the internal process data of the industrial internet laboratory into a document based database with timestamps. The data can also be read and edited afterwards via the RESTful interface service which makes the wide usage of the results of this work possible in future student projects and exercises.

Keywords: internet of things, protocols, programming, interfaces, programmable logic controller

SISÄLTÖ

Kuvaluettelo

Käytetyt termit ja lyhenteet

API *Application Program Interface* – Rajapinta ohjelmiston sisäisille toiminnoille, joiden hyödyntäminen on tehty mahdolliseksi muille täysin ulkoisille ohjelmille.

Asiakas Kutsuva ohjelmisto tai käyttäjä asiakas/palvelin-yhteydessä.

DCOM *Distributed Component Object Model* – Microsoftin teknologia, kokoelma konsepteja ja rajapintoja, joita käyttäen asiakasohjelma voi pyytää palveluita palvelinohjelmilta, joita ajetaan verkossa olevilla toisilla tietokoneilla.

Git Versionhallintaohjelmisto, joka toisin kuin SVN, toimii hajautetusti ja mahdollisimman tehokkaasti. Git on alun perin suunniteltu Unix-tyyppisille käyttöjärjestelmille joihin kuuluu muun muassa Linux, macOS ja tietysti Unix.

JSON *Javascript Object Notation* – Tekstipohjainen, käyttäjälle helposti luettava formaatti, jolla pystytään esittämään tietorakenteita sekä objekteja. Käytetään nykyään hyvin yleisesti web-ohjelmoinnissa.

LGPL-lisenssi Vapaan lähdekoodin ohjelmistolisenssi. Hyvin samanlainen MIT-lisenssin kanssa, mutta yhtenä erottavana rajoitteena on se, että jos LGPL-lisensoitua ohjelmistokoodia muokkaa ja käyttää, tulee muokattu versio LGPL-lisensoidusta koodista julkaista.

MIT-lisenssi Vapaan lähdekoodin ohjelmistolisenssi, joka kehitettiin Massachusetts Institute of Technologyssä. Se sallii tuotoksen käytön jopa kaupallisissa suljetun lähdekoodin ohjelmistoissa.

OLE *Object Linking and Embedding* – Microsoftin teknologia, kokoelma rajapintoja, jotka mahdollistavat compound-dokumenttien luonnin sekä näyttämisen.

Olio Yksittäinen ilmentymä tietorakenteesta, jolla on omat, sille kuuluvat arvot jäsenmuuttujilla.

OPC UA OPC *Unified Architecture* – Nykyinen, yleiskäyttöisempi ja järjestelmäarkkitehtuurista riippumaton versio OPC-protokollasta.

OPC *OLE for Process Control* – Tiedonsiirtoprotokolla teollisuuden automaatiolaitteille ja sovelluksille.

Palvelin Tietokoneohjelma, joka antaa palvelunsa muiden ohjelmistojen ja käyttäjien hyödynnettäviksi ja joka voi sijaita samalla tai eri tietokoneella.

RDP *Remote Procedure Call* – Protokolla, jonka avulla tietokoneohjelma voi kutsua toisen *palveluohjelman* jotain toiminnallisuutta. Tämä voidaan kuvitella funktio- tai aliohjelmakutsuna.

Relaatio Kahden muuttujan riippuvaisuus toisistaan.

REST *Representational State Transfer* – Arkkitehtuurillinen lähestymistapa web-pohjaisiin kommunikointijärjestelmiin.

Sarjallistaminen Olion tai tietorakenteen lukeminen datavirrasta olion tai tietorakenteen instanssiksi.

SOAP *Simple Object Access Protocol* – Viestintäprotokolla, joka mahdollistaa eri käyttöjärjestelmillä suoritettavien ohjelmien välisen kommunikoinnin käyttäen HTTP-protokollaa ja sen XML-kieltä.

SQL	*Structured Query Language* – 1970-luvulla luotu ja nykyään standardoitu ohjelmointikieli relaatiotietokantojen hallinnointia varten.
SSL	*Secure Sockets Layer* – Verkkoliikenneprotokolla, jonka tarkoituksena on varmistaa verkkoyhteyden molemminpuolinen tietoturva ja salaus muutoin tietoturvallisesti epävarmassa verkossa.
SVN	*Subversion* – Versionhallintaohjelmisto, joka toimii keskitetysti. Tämä tarkoittaa sitä, että projektin repository sijaitsee yhdellä palvelimella ja kommitoitaessa muutoksia projektiin on muutokset lähetettävä tälle keskitetylle palvelimelle verkon ylitse.
TCP/IP	*TCP/*IP tai *Stack* – Tällä viitataan verkkoliikenteen eri kerroksiin, joiden kautta data kulkee sekä asiakasohjelman että palvelinohjelman puolella tiedonvälitystapahtumien aikana.
Tietorakenne	Abstrakti muuttujista ja niiden arvoista koostuva yhtenäinen rakenne tietokoneen muistissa.
WAN	*Wide Area Network* – Suomeksi laajaverkko, on geograafisesti hajautettu, mutta silti yksityinen verkko, joka yhdistää useita LAN-verkkoja toisiinsa.
X.509-sertifikaatti	Digitaalinen sertifikaatti, joka käyttää yleisesti hyväksyttyä, kansainvälistä X.509 julkisen avaimen infrastruktuuristandardia varmistamaan, että julkinen avain kuuluu käyttäjä-, tietokone- tai palveluidentiteetille, joka on osana sertifikaattia.

1 Johdanto

1.1 Työn tausta

Opinnäytetyö tehtiin Seinäjoen Ammattikorkeakoululle ja työn kohteena oli teollisen internetin laboratorio.

Teollisen internetin hyödyntäminen automaatioprosessien ohjaamisessa sekä tarkkailussa lisääntyy jatkuvasti. Prosessien sisältämän informaation tallentamisella ja analysoinnilla voidaan muun muassa ennustaa ja jopa ehkäistä ongelmatilanteiden syntymistä. Tätä informaatiota hyödyntäen voidaan myös etsiä pullonkauloja prosessin eri välivaiheista, laskea arvioita seuraaville mahdollisille laitteiston huoltotarpeille tai jopa esimerkiksi säätää prosessille kuuluvia parametreja tilanteen mukaisesti.

Monet laitevalmistajat kehittävät omia teollisen internetin ratkaisujaan prosessidatan hyödyntämistä varten, ja näissä ratkaisuissa on hyvät ja huonot puolensa. Siemensillä on esimerkiksi MindSphere-pilvipalvelu ja Beckhoffilla TwinCAT IoT Communication- sekä TwinCAT Analytics -tuoteperheet. Vaikka nämä tuotteet alkavatkin jo olla varsin kypsiä ja tarjoavat laajan valikoiman eri toiminnallisuuksia, niin silti yleisenä ongelmana näissä ratkaisuissa on se, että ne ovat parhaiten yhteensopivia vain saman laitevalmistajan ohjelmistojen sekä laitteiden kanssa.

Laitevalmistajasta ja kohdealustan käyttöjärjestelmästä riippumattoman ratkaisun toteutus on siksi hyvin ajankohtaista.

1.2 Työn tavoitteet

Työn tavoitteena on suunnitella ja toteuttaa sovellus, joka mahdollistaa reaaliaikaisen informaation tallentamisen Seinäjoen Ammattikorkeakoulun teollisen internetin laboratorioon kuuluvista PLC-laitteista. Tämän lisäksi tavoitteena on myös toteuttaa tämä informaation tallennuksen hoitava sovellus siten, että se on mahdollisimman yleiskäyttöinen.

Tutkimusongelmana on suunnitella ja toteuttaa esimerkkisovellus, joka ei ole riippuvainen sovelluksen suoritusympäristön käyttöjärjestelmästä tai kohdeprosessin PLC-laitteiden valmistajista.

1.3 Työn rakenne

Luvussa 2 kerrotaan lyhyesti teollisesta internetistä, SeAMKin teollisen internetin laboratoriosta ja työssä syntyneestä esimerkkisovelluksesta. Luvussa 3 käydään läpi työhön liittyvien tiedonsiirtoprotokollien historiaa ja rakennetta. Luvussa 4 esitellään yleiset tietokantatyypit. Luvussa 5 kerrotaan työssä käytetyistä teknologioista. Luvussa 6 kerrotaan työn tuloksena syntyneen esimerkkisovelluksen rakenteesta, käytetyistä ohjelmointikielistä ja kirjastoista, konfiguroinnista ja toimintalogiikan perusteista. Luvussa 7 on esitelty esimerkkisovelluksen jatkokehitysideoita, sekä Git-versionhallinta-repositoryt. Luvussa 8 on työkokonaisuuden yhteenveto ja työn tulosten esittely.

2 Teollisen internetin laboratorio

2.1 Teollinen internet

Teollinen internet tarkoittaa fyysisten laitteiden kykyä viestiä internetyhteyden välityksellä ja tämän viestinnän ympärille rakennettua liiketoimintaa. (Teknologiateollisuus [Viitattu 3.4.2017]).

Tähän viestintään liittyy yleensä monenlaisia sensoreita, joiden luomaa dataa laite jakaa verkossa oleville muille laitteille tai esimerkiksi yhdelle keskitetylle palvelimelle. Esineiden internetin laitteet sisältävät yleensä myös toimintalogiikkaa, joka on osittain riippuvainen muiden laitteiden verkkoon jakamasta datasta. (Quva [Viitattu 25.3.2017].)

Pääasiassa teollisen internetin hyödyt yritystoiminnassa sijaitsevat sen avulla tuotetun tiedon hyödyntämisen sektorilla. Laitteet voivat päivittää tilannettaan verkon välityksellä esimerkiksi yhteen keskitettyyn tietokantaan ja sen kautta saadaan aikaleimoilla varustettua dataa analysoitavaksi. Tämä datan analysointi taas mahdollistaa monenlaisia asioita teollisuuden eri osa-alueilla. Voidaan muun muassa optimoida tuotanto- huolto- tai muita toimintaketjuja yrityksessä siten, että käyttökatkot eri laitteilla saadaan mahdollisimman minimaalisiksi. Laitteiden käyttöastetta saadaan kasvatettua ja tehtyä huomattavia säästöjä muun muassa huollossa ja materiaalikustannuksissa. (Quva [Viitattu 25.3.2017].)

Konkreettisempi esimerkki teollisen internetin soveltamisesta teollisuudessa on ennakoiva kunnossapito esimerkiksi laitteiden värinä- tai kiihtyvyysdatan analysoinnin kautta. Rikkoutumispistettä lähellä oleva laakeri voi aiheuttaa värinää, joka taas voidaan havaita laitteen tallentamasta datasta selvästi. Tämä myös mahdollistaa esimerkiksi osan vaihdon ajoittamisen juuri oikeaan aikaan, jos aiemman datan perusteella tiedetään suunnilleen osan todellinen rikkoutumisraja. (Mäkelä & Ristimäki 2016, 469.)

2.2 SeAMKin Teollisen internetin laboratorio

Seinäjoen Ammattikorkeakoulun teollisen internetin laboratorio sai syntynsä osittain Eläköön Automaatio -palkinnon voittaneen Digital Factory -projektin menestyksen johdosta. Digital Factory muun muassa mahdollistaa virtuaalisten 3D-mallien ohjaamisen fyysisellä PLC-laitteistolla. Tämä tarjoaa monia mahdollisuuksia kokonaisen tuotantoprosessin, prosessin osien tai yksittäisten laitteiden testaamiseen simuloidussa virtuaaliympäristössä, jota kuitenkin ohjataan aidolla fyysisellä laitteistolla. (SeAMK 9.3.2016.)

Digital Factory -projektista saatu palaute ja sen kautta hankittu osaaminen toivat esille tarpeen ympäristöstä, jossa on mahdollista opetella teollisen internetin hyödyntämistä automaatiossa. Näin syntyi teollisen internetin laboratorio -projekti, oppimisympäristö, joka on tärkeä opiskelijoille, opettajille sekä alueen yrityksille alan tulevaisuutta ajatellen. (SeAMK 9.3.2016.)

Kuva 1. Teollisen internetin laboratorion 3D-malli (SeAMK 9.3.2016).

Laboratorion tarkoituksena on olla käytännön tason opetuksen mahdollistava ympäristö, jonka avulla voidaan kehittää ja tutkia tuotannon eri parametreja säätäviä sekä analysoivia teollisen internetin sovelluksia. (SeAMK 9.3.2016).

Kuvassa 1 on SeAMKin teollisen internetin laboratoriossa oleva tuotantolinja. Tuotantolinjan on toimittanut Festo ja se koostuu kolmesta moduulista. Varasto-asemaa

ohjaa Siemensin PLC ja itse varastosta tavaraa haetaan robotilla, jonka tarttuja liikkuu varaston sisällä X, Y- ja Z-suunnissa. 3D-mallin kuvan keskimmäinen asema koostuu konenäöstä ja porausasemasta. Konenäöllä tarkastetaan robotin kokoamat tai purkamat osat, tätä asemaa ohjaa Beckhoffin PLC. Robottiasemalla on ABB:n robotti ja tuotannon ohjauksen hoitaa Siemensin PLC. (SeAMK 9.3.2016.)

Laboratorion tuotannonohjausjärjestelmänä toimii Feston MES4-sovellus. Se pitää kirjaa tuotannon tilauksista Microsoft Access -tietokannan avulla. Viestintä PLC-laitteiden ja tuotannonohjauksen välillä tapahtuu binääripohjaisella socket-yhteydellä, MES4 toimii palvelimena, johon PLC-laitteet yhdistävät asiakasohjelman roolissa. Jos MES4 sammutetaan tai yhteys katkeaa jostain muusta syystä, yrittävät PLC-laitteet aktiivisesti yhdistää uudestaan PLC-ohjelmassa määriteltyyn MES4-sovelluksen osoitteeseen.

2.3 OPC UA Data -esimerkkisovellus

Tämän työn tuloksena syntyi esimerkkisovellus, joka mahdollistaa reaaliaikaisen datan lukemisen PLC-laitteilta ja tämän luetun datan säilömisen keskitettyyn tietokantaan. Tietokannan kanssa viestimiseen luotiin RESTful-rajapinta, jonka kautta käyttäjä pystyy tallentamaan, päivittämään, sekä hakemaan dataa tietokannasta. Rajapintakyselyt tukevat myös suodatinparametreja, joilla yksittäisen kyselyn käsittämää kohdedatajoukkoa voi rajoittaa.

Esimerkkisovellus on kokonaisuus, joka koostuu OPC UA Data Gateway-, OPC UA Data REST-, OPC UA Data Visualizer-, sekä OPC UA Data Server -sovelluksista. Gateway toimii tiedonvälittäjänä OPC UA -palvelinten ja RESTful-rajapinnan välillä. REST toimii tietokannan rajapintana, muokkaustoiminnot ovat Basic Auth -autentikoinnin takana. Visualizer on tietokannan datan tarkastelujärjestelmä, joka käyttää myös samaa yhteistä RESTful-rajapintaa. Server on websocket-palvelin, joka mahdollistaa reaaliaikaisen datan lukemisen tietokannasta esimerkiksi Visualizer -sovelluksessa. Sovellukset keskustelevat keskenään HTTP-protokollan ja suoran TCP/IP-yhteyden välityksellä.

PLC-laitteen ja esimerkkisovelluksen välisessä kommunikoinnissa voidaan käyttää joko OPC-, OPC UA- tai ADS-protokollaa. ADS-protokolla ei ole käyttökelpoinen tässä sovelluksessa, sillä sitä voidaan käyttää vain Beckhoffin PLC-laitteiden kanssa. Luonnollisena, järkevimpänä vaihtoehtona jäljelle jäi vain OPC UA, alustasta ja laitevalmistajasta riippumattoman suunnittelun, sekä vahvan standardoinnin takia.

3 Protokollat

3.1 OPC

OPC on lyhenne sanoista *OLE for Process Control* ja se pohjautuu Microsoftin OLE- ja DCOM-teknologioihin. OPC rakentuu monesta erillisestä itsenäisestä protokollasta. Viestintä tapahtuu asiakas/palvelin-periaatteella ja siitä on ajan myötä tullut käytetyin tapa automaatiosovellusten välillä käytettävään kommunikointiin. OPC asiakas/palvelin -yhteys voi toimia joko siten, että asiakas pyytää haluamiaan tietoja palvelimelta tai sitten tapahtumapohjaisena, missä palvelin toimittaa tietoja asiakkaalle tietojen muutostilanteissa. (Novotek [Viitattu 7.12.2016].)

OPC-palvelintuki missä tahansa järjestelmässä pienentää mahdollisia muiden ohjelmistojen ja laitteistojen integrointi- sekä kehityskustannuksia ja parantaa yleistä tiedonvälityksen turvallisuutta. (Prosys OPC [Viitattu 17.12.2016]).

OPC-protokollan muita hyötyjä ovat muun muassa vahvasti standardoidut ohjelmistokomponentit, joista OPC koostuu. Edellä mainittujen ominaisuuksien ansiosta tuotekehitys OPC-pohjaisille ratkaisuille on nopeaa ja tuloksena syntyvät ohjelmistot ovat luotettavia. Standardin mukainen kommunikaatiorajapinta mahdollistaa myös luonnostaan valmistajariippumattoman sovellusten ja laitteiden integroinnin. (Wapice [Viitattu 13.3.2017].)

3.1.1 OPC Data Access

Eniten käytetty OPC-protokolla on OPC Data Access, sillä lähetetyssä ja vastaanotetussa datassa on jokaisessa tietueessa itse haetun arvon lisäksi muuttujan nimi, aikaleima ja tyyppi. OPC DA -protokollaa käytetään lähinnä kommunikoitaessa ohjaus- ja automaatiojärjestelmien kanssa. Se on tarkoitettu reaaliaikaiseen datan tarkasteluun sekä manipulointiin. (Novotek [Viitattu 7.12.2016].)

3.1.2 OPC Alarm & Events

OPC AE -protokolla otettiin osaksi OPC-standardia OPC DA:n jälkeen. Se on tilauspohjainen protokolla ja yhdistetty asiakasohjelma vastaanottaa aina kaikki tilaamansa tapahtumat. Vastaanotetun OPC AE -tapahtuman tietueissa ei ole muuttujien nimeä eikä tyyppiä. Kuten OPC DA, OPC AE käsittelee myös vain muuttujien hetkellisarvoja ilman minkäänlaista historiatietojen varastointia. Nimensä mukaisesti OPC AE on tarkoitettu hälytys- ja tapahtumatietojen välitykseen asiakkaalle. Asiakas voi halutessaan myös suodattaa tilaamistaan tapahtumista loogisilla ehdoilla halutut tapahtumat pois. (Novotek [Viitattu 7.12.2016].)

3.1.3 OPC Historical Data Access

OPC HDA mahdollistaa kerätyn datan historiatiedon tarkastelun OPC-palvelimen tietokannasta. Protokolla tukee useamman kuin yhden muuttujatagin historian tarkastelua aikaleimarajojen avulla. (Novotek [Viitattu 7.12.2016].)

3.2 OPC Unified Architecture

OPC UA on OPC-standardin pohjalta jalostettu alustariippumaton binääriprotokolla, jonka ensimmäinen versio julkaistiin vuonna 2008. Siinä on päätavoitteina alustariippumattomuus sekä arkkitehtuuri, joka mahdollistaa kaikkien OPC-protokollien toiminnallisuuksien tukemisen samassa paketissa. (OPC Foundation [Viitattu 7.12.2016].)

OPC UA Hyväksyttiin IEC standardiksi 62541 vuonna 2011. (Prosys OPC [Viitattu 17.12.2016]).

3.2.1 Alustariippumattomuus

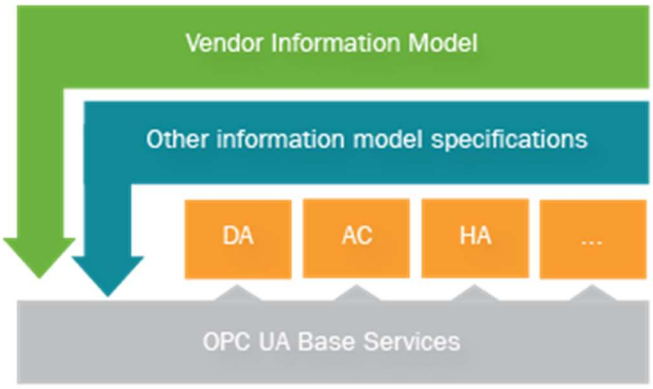
OPC UA ei hyödynnä Microsoftin OLE- tai DCOM-teknologiaa kuten edeltäjänsä. Tämän ansiosta se on täysin laitteistoalustasta ja käyttöjärjestelmästä riippumaton

ratkaisu. Tuettuja käyttöjärjestelmiä ovat mm. Microsoft Windows, Apple OSX/macOS, Android tai mikä tahansa Linux/Unix-jakelu. (OPC Foundation [Viitattu 7.12.2016].)

OPC UA korvaa vanhan OPC-standardin käyttämän DCOM-pohjaisen kommunikaation TCP/IP-, HTTP/HTTPS- ja SOAP-protokollilla. Uuden standardin tuoma lisäturvallisuus mahdollistaa myös muun muassa WAN-yhteydet. (Prosys OPC [Viitattu 17.12.2016].)

3.2.2 Tiedon mallinnus

OPC UA mahdollistaa hyvin monimutkaistenkin tietueiden sekä näiden välisten yhteyksien mallintamisen hyödyntämällä esimerkiksi BACNet-protokollaa tai XML-skeemoja. OPC UA -palvelimen rakenne koostuu noodeista, jotka sijaitsevat puurakenteessa. Tämän puurakenteen eri noodien tai noodiryhmien välillä voidaan määritellä monimutkaisia yhteyksiä ja relaatioita. (Novotek [Viitattu 7.12.2016].)



Kuva 2. OPC UA -palvelimen tietuemallit
(Novotek [Viitattu 7.12.2016]).

Tavallisesti OPC UA -palvelimen puurakenne koostuu OPC-protokollapinon tietorakenteista, muista mahdollisista tietorakenteista sekä laitteiston ja ohjelmiston valmistajan omista tietorakenteista. (Novotek [Viitattu 7.12.2016]).

3.2.3 Turvallisuus

Protokolla tukee istuntojen ja palvelimen välisen viestinnän salausta 128- tai 256-bittisellä salauksella. OPC UA käyttää X509-tyypin sertifikaatteja viestien allekirjoittamisessa. Tämä tarkoittaa sitä, että viestin vastaanottaja voi tarkistaa lähettäjän oikeellisuuden tarkistamalla viestistä löytyvän X509-sertifikaatin allekirjoituksen. OPC UA -yhteydessä käytettävä sertifikaatti koostuu julkisesta salausavaimesta, yksityisestä salausavaimesta sekä sertifikaatin omistajan tiedoista. Lähetettävä viesti voidaan näin salata julkisella avaimella ja sitä ei pysty lukemaan selkokielisenä kuin ainoastaan yksityisen salausavaimen avulla. (Kepware, 25.9.2014.)

Autentikointi on tuettu sekä palvelin- että käyttäjätasoilla. Yksityisten käyttäjien käyttöoikeuksia palvelimella on mahdollista hallita, jokaisen käyttäjän toiminnot kirjoitetaan lokiin myöhempää tarkastelua varten. (OPC Foundation [Viitattu 7.12.2016].)

Yhteys OPC UA -asiakkaan ja -palvelimen välillä rakentuu kahdesta eri tasosta, jotka puolestaan käyttävät standardoitua TCP/IP-pinoa. Sessiotasolla käsitellään autentikointiin liittyvät pyynnöt ja vastaukset sekä hoidetaan muita sessioiden käsittelyyn liittyviä pyyntöjä. Tiedonvälitystasolla taas käytetään joko SSL-, HTTP- tai HTTPS-yhteyttä pääosin datan välitykseen. Tämä taso käyttää hyväkseen aiemmin mainittuja X509-tyypin sertifikaatteja pyynnöissään, mikä esimerkiksi antaa turvan käynnissä olevan yhteyden kaappaus- ja manipulointiyrityksiltä. (Novotek [Viitattu 17.12.2016].)

3.3 HTTP

HTTP tulee sanoista *Hypertext Transfer Protocol*. Se on sovellustason tiedonsiirtoprotokolla ja maailmanlaajuisen *World-Wide Web*in tiedonsiirron ehdoton tukipilari. Ensimmäinen HTTP:n versio, HTTP/0.9, oli yksinkertainen protokolla paljaan datan lähettämistä varten internetin välityksellä. (Fielding, Irvine, Gettys, Mogul, Frystyk, Masinter, Leach & Berners-Lee 1999, 6.)

Käytännölliset ratkaisut vaativat enemmän toiminnallisuuksia kuin mitä HTTP/0.9-versiolla on tarjota. HTTP/1.0-julkaisun myötä protokolla tukee muun muassa metainformaation lähetystä datan ohella, mikä kertoo vastaanottajalle lähetetyn datan rakenteesta, sekä pyynnön ja vastauksen muuttujien merkityksistä. (Fielding, Irvine, Gettys, Mogul, Frystyk, Masinter, Leach & Berners-Lee 1999, 6.)

Protokollaa käytetään myös proxyjen, yhdyskäytävien ja selainten geneeriseen kommunikointiin muiden internet palvelujen kanssa. Nämä palvelut voivat muun muassa tukea SMTP-, NNTP-, FTP-, Gophe- ja WAIS -protokollia. Tämän ansiosta HTTP voi sallia pääsyn hypermediaresursseihin monenlaisten sovellusten kautta. (Fielding, Irvine, Gettys, Mogul, Frystyk, Masinter, Leach & Berners-Lee 1999, 7.)

3.3.1 Kyselymetodit

HTTP Määrittelee joukon metodeja, joilla jokaisella on oma tarkoituksensa HTTP-resurssien käsittelyssä. Seuraavista metodeista kaikki paitsi *PATCH* määritellään RFC 7231 -spesifikaatiossa: (Fielding & Reschke 2014, 24 & 28.)

GET: Metodi edustaa tietyn resurssin tietojen pyyntöä palvelimelta. Se on tarkoitettu käytettäväksi vain datan hakemista varten.

POST: Metodi edustaa tietyn resurssin tietojen lähettämistä palvelimelle.

PUT: Sama periaate kuin POST-metodilla, mutta PUT on tarkoitettu palvelimen puolella olemassa olevan resurssin tietojen päivitykseen.

DELETE: Resurssien poistaminen, parametrit esimerkiksi poistettavien resurssien valitsemista varten annetaan URL:n perään.

HEAD: Sama periaate kuin GET-metodilla, mutta palvelin ei lähetä pyydetyn resurssin tietoja, vaan ainoastaan kyselyn HEADER-osuuden, joka sisältää metatietoa resurssista.

TRACE: Palvelin palauttaa pyynnön kaikuna takaisin, tällä tavalla kyselyn lähettänyt asiakasohjelma voi tarkistaa, miten kyselyn käsitelleet palvelut muokkasivat alkuperäisen kyselyn tietoa.

OPTIONS: Metodi pyytää palvelimelta listan tuetuista HTTP-metodeista annetulla URL-osoitteelle.

CONNECT: Metodi muuttaa alkuperäisen pyynnön yhteyden TCP/IP-tunneliyhteydeksi. (Fielding & Reschke 2014, 24-32.)

PATCH (*RFC-2068*): Metodi on hyvin saman kaltainen kuin PUT, mutta sillä on tarkoitus tehdä lähinnä osittaisia muutoksia tietyn resurssin tietoihin. (Fielding 1997, 155.)

3.3.2 Statuskoodit

HTTP-protokollan 1.0-versiosta lähtien palvelimen lähettämässä vastauksessa ensimmäinen rivi kuuluu statuskoodille, joka kertoo asiakasohjelmalle, miten kyselymetodin suoritus meni ja tarvitaanko sen suhteen muutos- tai jatkotoimenpiteitä. Koodin ensimmäisestä luvusta voidaan päätellä koodin yleinen vastausluokka ja kaksi viimeistä lukua edustavat koodin varsinaista numerokoodia. (Fielding & Reschke, 2014. 17.)

1XX (*Informational*): Informatiivinen vastauskoodi. Indikoi asiakasohjelmalle, että vastaanotettu viesti ymmärrettiin ja sitä esimerkiksi tällä hetkellä prosessoidaan. Tällä voidaan myös viestiä, että asiakkaan protokollan vaihdos -pyyntö hyväksyttiin tai että palvelin antaa luvan asiakkaalle jatkaa pyynnön runko-osuuden lähetystä sellaisten pyyntöjen kanssa, joissa runko vaaditaan (esimerkiksi POST-pyyntö). (Fielding & Reschke 2014, 49-50.)

2XX (*Successful*): Pyynnön käsittelyn onnistumisen indikoiva vastauskoodi. Lopullinen koodi muodostuu alkuperäisen pyynnön tyypin ja rakenteen mukaisesti. Yleisesti käytetään *200 OK* -viestiä, mutta esimerkiksi onnistuneen uuden resurssin luomisesta ilmoitetaan *201 Created* -koodilla. Palvelin voi myös ilmoittaa *202 Accepted* -koodilla, että alkuperäinen pyyntö hyväksyttiin, mutta sen prosessointi ei ole vielä täysin valmis. (Fielding & Reschke 2014, 50-53.)

3XX (*Redirection*): Palvelin ilmoittaa, että pyynnön onnistunut käsittely vaatii asiakkaalta lisätoimenpiteitä, useissa tapauksissa näillä viesteillä tarkoitetaan URL-uudelleenohjausta. *300 Multiple Choises* -koodi kertoo, että asiakkaalla on vaihtoehtoina useita eri URL-lähteitä pyydetyn resurssin lataamiselle. Myös esimerkiksi proxyn pakollinen käyttö pyynnöissä jatkossa voidaan ilmoittaa *305 Use Proxy* -koodilla. (Fielding & Reschke 2014, 53-57.)

4XX (*Client error*): Nämä viestit indikoivat asiakkaalle, että lähetetty pyyntö oli jollain tapaa virheellinen. Yleisiä *4XX*-koodeja ovat muun muassa *400 Bad Request*, *401 Unhauthorized*, *403 Forbidden* ja *404 Not Found* -koodit. (Fielding & Reschke 2014, 57-62.)

5XX (*Server error*): Palvelin ilmoittaa, että se itse epäonnistui sisäisesti sellaisen pyynnön käsittelyssä, jonka alkuperäinen rakenne oli täysin oikeaoppinen. Yleisesti käytettyjä koodeja ovat *500 Internal Server Error* tai *501 Not Implemented*. Näistä ensimmäinen indikoi, että jonkun pyydetyn tai lähetetyn resurssin käsittelyn aikana tapahtui virhe josta ei voida palautua ja jälkimmäinen sitä, että pyydetylle resurssille ei ole implementoitu alkuperäisen pyynnön tyyppiä (Voi esimerkiksi olla, että PUT ei ole ollenkaan tuettu tietylle resurssille). (Fielding & Reschke 2014, 62-63.)

4 Tietokannat

Tässä luvussa käsitellään relaatiotietokantoja ja NoSQL-tietokantoja.

Tietokannat mahdollistavat informaation kokoamisen samaan paikkaan tietokoneen muistissa sellaiseen muotoon, että sitä on tehokasta täydentää, päivittää ja lukea. (Oracle [Viitattu 18.12.2016]).

4.1 Relaatiotietokannat

Relaatiotietokannat edustavat informaatiota säilöttynä tauluihin, joissa on rivejä ja sarakkeita. Informaatioon taulussa voidaan viitata yhteisillä avaimilla muiden taulujen sarakkeiden kanssa. Mahdollisuus hakea tietyn taulun relaatioiden dataa on relaatiotietokantojen perusperiaate. (Oracle [Viitattu 18.12.2016].)

Relaatiopohjaiset tietokannat tulivat käyttöön 1970-luvulla. Tällöin dataskeemat olivat vielä paljon yksinkertaisempia kuin nykyisin, joten olioiden käsittely sarjana relaatioita oli järkevä ja hyvin yleispätevä ratkaisu. Esimerkiksi tietokantaan säilöttävä artikkeliolio voisi olla yhteydessä myös kategoria- ja kommenttitauluihin, koska artikkeli sisältää informaation kategoriasta ja siihen liittyvistä kommenteista, jotka toisaalta ovat myös itsenäisiä olioita. Koska kaikki relaatiot määritellään tietokannan rakennetta esittävään tietokantaskeemaan, voidaan relaatiotietokantoihin kohdistaa kyselyitä standardilla SQL-kielellä. (MongoDB [Viitattu 18.12.2016].)

4.1.1 Konfigurointi ja suunnittelu

Relaatiotietokannoissa on tiettyjä yhteneväisyyssääntöjä. Konfiguroitaessa relaatiotietokantaa käyttäjä voi aluksi valita, sallitaanko kaksoisrivejä ollenkaan vai ei. Jos niitä ei sallita, niin tietokanta estää oletuksena sellaisten rivien lisäämisen tauluihin. Valittaessa taululle pääavainsaraketta, tulee myös ottaa huomioon se, että pääavainsarakkeen sisältämän informaation tulee olla yhtenäistä, eikä se saa sisältää tyhjiä arvoja tai kaksoisrivejä. Tämä sääntö juontuu siitä, että yksittäisen rivin tulee

olla löydettävissä vähintään taulun pääavaimen arvon avulla. (Oracle [Viitattu 18.12.2016].)

Tietokannan rakenteen suunnittelu on hyvin tärkeää. Etenkin relaatiokantojen kohdalla huono suunnittelu voi johtaa jälkikäteen muun muassa informaation rikkoutumiseen, epäsäännöllisiin relaatioihin, jatkuvaan ylläpidon tarpeeseen ja tarpeettoman monimutkaisten SQL-kyselyiden kehittämiseen. Tietokannan suunnittelun aikana on hyvä käydä seuraavat vaiheet läpi:

- Käsiteanalyysi: tietokannan rakenteen karkea luonnostelu.
- Tarveanalyysi: testataan tehtyä käsitemallia ja täydennetään sitä testitulosten perusteella.
- Normalisointi: tietokantamallin tarkistus turhaan toistuvan informaation varalta.
- Taulujen muodostaminen: käsitemallin muunnos relaatiokannan tauluiksi.
- Suorituskyvyn tarkistus: testataan tietokannan suorituskykyä kuorman alla, käydään läpi kaikki indeksoitavat sarakkeet, yleinen optimointi. (Hovi 2004, 10.)

4.1.2 Tietokantakyselyt

SQL-kielen yleisimmät komennot ovat SELECT, INSERT, UPDATE, DELETE ja DROP -komennot. Näillä komennoilla käsiteltävä datajoukko valitaan komennon perään tulevan WHERE-lauseen loogisilla ehdoilla. Kaikki SQL-komennot voidaan luokitella ryhmiin komentojen tarkoituksien mukaisesti. (Tutorials Point [Viitattu 11.4.2017].)

Ohessa yleisimmät komennot luokiteltuna kolmeen eri ryhmään:

- DDL – Data Definition Language
  - CREATE: Luo kohdetietokantaan uusi taulu tai muu objekti. Käytetään myös tietokannan luontiin (CREATE DATABASE databasename).
  - ALTER: Muokkaa olemassa olevaa objektia.
  - DROP: Poista objekti, esimerkiksi taulu tai tietokanta.
- DML – Data Manipulation Language

- SELECT: Valitse yksi tai useampi rivi kohdetietokannan kohdetaulusta.
- INSERT: Luo uusi tauluun uusi rivi.
- UPDATE: Päivitä olemassaolevia rivejä.
- DELETE: Poista rivejä.

- DCL – Data Control Language
  - GRANT: Myönnä käyttöoikeudet kohdekäyttäjälle esimerkiksi kohdetietokantaan tai kohdetauluun.
  - REVOKE: Poista käyttöoikeudet kohdekäyttäjältä. (Tutorials Point [Viitattu 11.4.2017].)

	username	sha_pass_hash	gmlevel	sessionkey	v	s	joindate	last_ip	failed_logins		last_login				
1	ADMINISTRATOR	a34b29541b87b7e4823683ce6c7bf6ae68beaaac	3		0	0	2006-04-25 13:18:56	127.0.0.1	0	0	0000-00-00 00:00:00	0	0	0	0
2	GAMEMASTER	7841e21831d7c6bc0b57fbe7151eb82bd65ea1f9	2		0	0	2006-04-25 13:18:56	127.0.0.1	0	0	0000-00-00 00:00:00	0	0	0	0
3	MODERATOR	a7f5fbff0b4eec2d6b6e78e38e8312e64d700008	1		0	0	2006-04-25 13:19:35	127.0.0.1	0	0	0000-00-00 00:00:00	0	0	0	0

Kuva 3. SELECT-kyselyllä valittuja esimerkkirivejä tietokantataulusta

Kuvassa 3 on esimerkkiotanta verkkopelin account-taulusta. Valinta tehtiin kuvassa 4 näkyvällä SELECT-tyyppisellä SQL-kyselyllä.

```
SELECT * FROM realmd.account WHERE gmlevel != 0;
```

Kuva 4. Esimerkkikysely verkkopelin tietokannan account-tauluun

Kuvassa 4 valitaan kaikki (*) sarakkeet (realmd.account)-taulusta sillä ehdolla, että rivin (gmlevel)-sarakkeen arvo ei ole 0. SQL-kieli on siis yleisesti ottaen hyvin selvä ohjelmointikieli ja tarpeeksi yksinkertaisten kyselyiden idea avautuu käyttäjälle normaalisti nopeasti.

4.2 Dokumenttipohjaiset tietokannat

Dokumenttipohjaiset tietokannat kehitettiin vastaamaan ongelmiin, joita relaatiotietokantojen kanssa saattaa ilmetä. Yleisesti ei-relaatiomallisiin tietokantoihin viitataan termillä *NoSQL,* ja ne sallivat informaation luokittelun luontevammin sekä loogisemmin verrattuna perinteisiin relaatiotietokantoihin. Yksi yleisimmistä NoSQL–

tietokantamalleista on dokumenttipohjainen tiedonsäilöntämalli. Siinä jokainen tallennettu tietokannan tietue esitetään dokumenttina, joka karkeasti sanottuna on C-kielen structin-tyylinen tietorakenne tai jopa luokka oliopohjaisessa ohjelmointikielessä, mutta ilman jäsenfunktioita. Tämän ansiosta kaikki relaatiot dokumenttiin löytyvät itsenäisinä dokumentteina itse dokumenttiobjektin sisältä. (MongoDB [Viitattu 18.12.2016].)

```
{
    "_id" : ObjectId("582b0c02f0be8f06d5a1eb15"),
    "_class" : "io.github.harha.rest.model.OPCUAVariable",
    "identifier" : "dbRfidData.ID1.Mes.iResourceId",
    "nsIndex" : 4,
    "type" : "int16_t",
    "serverId" : 2,
    "value" : "0",
    "serverTimeStamp" : ISODate("2016-11-15T14:20:13.135Z"),
    "localTimeStamp" : ISODate("2016-11-15T13:22:10.016Z")
}
```

Kuva 5. Esimerkki MongoDB-dokumentista tekstimuodossa

Kuvan 5 esimerkkidokumentissa on OPCUAVariable-nimisen Java-luokan instanssi esitettynä MongoDB-dokumenttina. Dokumentit voivat sisältää myös listoja, sellaisia ei tässä esimerkkitapauksessa ole. (MongoDB [Viitattu 18.12.2016].)

Yleisesti dokumenttipohjaisen tietokannan käytössä on ainakin seuraavat edut:

- Dokumentit ovat itsenäisiä tietueita.
  - Tämä tarkoittaa sitä, että suorituskyky on yleisesti ottaen parempi, koska relaatiodata on kiinni samassa muistialueessa kuin itse dokumentti.
- Sovelluslogiikka on mahdollista kirjoittaa yksinkertaisemmin.
- Dokumentit voidaan muuttaa olioiksi tai toisinpäin todella helposti, koska dokumentti itsessään rakenteeltaan muistuttaa hyvin paljon perinteistä luokkaa olio-ohjelmoinnissa.
- Epäsäännöllisen datan tallentamisen yksinkertaisuus, eli samassa kokoelmassa olevien dokumenttien ei tarvitse rakenteeltaan vastata toisiaan.
- Dokumentti voi sisältää mitä tahansa avaimia ja arvoja sovelluslogiikka tarvitsee, eikä erillisten samassa ryhmässä sijaitsevien dokumenttien tarvitse olla rakenteeltaan identtisiä. (MongoDB [Viitattu 18.12.2016].)

4.2.1 Tietokantakyselyt

Dokumenttipohjaisissa tietokannoissa on yleisesti hyvin suorituskykyiset kyselyominaisuudet, jotka mahdollistavat hyvin optimoitujen kyselyjen nopean suorituksen. Yleisestikin tällaisten tietokantojen suuri markkinointivahvuus on kyselykielen monipuoliset ominaisuudet. (MongoDB [Viitattu 18.12.2016].)

```
// get db collection instance
var collection = db.collection('opc_ua_variable');

// find latest variable
var results = collection.find({
            $and: [
                        {identifier: data.identifier},
                        {serverId: data.serverId},
                        {nsIndex: data.nsIndex}
            ]
}).sort({
            $natural: -1
}).limit(1);
```

Kuva 6. Esimerkkikysely MongoDB-tietokantaan Javascript-kielellä

Kuvan 6 esimerkkikyselyssä käskytetään Javascript-kielellä MongoDB-tietokantaa hakemaan tietokannan (opc_ua_variable)-nimisestä kokoelmasta dokumentteja loogisten ehtojen perusteella, joita jokaista erottaa (AND, &&)-operaattori. Haku myös järjestetään alkamaan uusimmasta dokumentista, ja haun tulosten määrä rajoitetaan yhteen dokumenttiin. Results-objektista saadaan tulokset haluttaessa normaaliin listaan results.toArray-funktiota kutsumalla.

5 Teknologiat

Tässä luvussa käydään läpi työn ohjelmistoissa hyödynnettyjä teknologioita ja kirjastoja.

5.1 Open62541

Open62541 on avoimen lähdekoodin OPC UA -toteutus. Se on ohjelmoitu C-kielellä käyttäen kielen C99-versiota. Kirjastoa on mahdollista käyttää C-projektien lisäksi myös C++-projekteissa ja itse kirjasto sisältää OPC UA -binääriprotokollan toteutuksen lisäksi asiakas- ja palvelintoiminnallisuuksien toteutukset. Open62541 on lisensoitu LGPL-lisenssin alla staattisen linkityksen poikkeuksella, joka antaa kirjaston käyttäjälle LGPL-lisenssin myöntämät oikeudet riippumatta siitä, linkitetäänkö kirjasto dynaamisesti vai staattisesti ohjelmaan, jossa sitä käytetään. Tämä mahdollistaa kirjaston käyttämisen jopa suljetun lähdekoodin projekteissa, sillä poikkeuksella, että kaikki muutokset itse kirjastoon tulee julkaista saman lisenssin alla. (Open62541 26.12.2016, 1.)

5.1.1 Tuetut toiminnallisuudet ja puutteet

Open62541 noudattaa OPC UA:n määrittelyjä niin tarkasti kuin mahdollista, mutta koska kirjasto on vasta alkuvaiheessa kehityksen osalta, siitä puuttuu toistaiseksi vielä jotain tiettyjä ominaisuuksia (Open62541 26.12.2016, 2).

Ohessa lista kirjaston 0.2-version tukemista ominaisuuksista:

- Kommunikointipino
  - OPC UA –protokolla.
  - Suurten viestien paloittelu pienempiin osiin.
  - Vaihdettavissa oleva verkkotaso lisäosaksi toteutettuna, mahdollistaa kustomoitujen verkkorajapintojen käytön.
- Informaatiomalli
  - Tuki kaikille OPC UA -noodien tyypeille.

    - Tuki noodien lisäämiselle ja poistamiselle, myös palvelimen ajon aikana.
    - Tuki olio- ja muuttujatyyppien alustukselle sekä olioiden perimiselle.
- Tilaukset
    - Tuki noodin arvon muutosilmoituksen tilaukselle.
    - Todella alhainen resurssien käyttö per monitoroitu arvo.
- Koodin generointi
    - Tuki datatyyppien generoimiselle standardin XML-määrittelyn mukaisesti.
    - Tuki palvelinpuoleisen infromaatiomallin generoinnille XML-määrittelyn mukaisesti. (Open62541 2016, 2.)

Versiosta 0.2 puuttuvat toiminnallisuudet ovat:

- Viestinnän salaus.
- Yksittäisten noodien ohjaus.
- Tapahtumat (olioiden lähettämät ilmoitukset).
- Tapahtuma-silmukka (taustatehtävät) ja asynkroniset palvelukutsut clientin puolella. (Open62541 26.12.2016, 2.)

5.2 Spring Boot

Spring Boot on Spring Frameworkiin pohjautuva sovelluskehys, joka automatisoi tuotantovalmiin Spring-sovelluksen kehityksessä vaadittuja toimenpiteitä. Sovelluksen Spring-kirjastoriippuvuudet yksinkertaistuvat. Sen voi kääntää valmiiksi suoritettavaksi JAR-ajotiedostoksi, jonka ajaminen ei vaadi erillistä isäntäsovellusta HTTP-palvelinta varten (esim. Apache Tomcat, Oracle Glassfish). Konfigurointivaatimukset ovat myös hyvin minimaaliset verrattuna normaaliin Spring-sovellukseen, koska Spring Boot konfiguroi sovelluksen hyvin pitkälti automaattisesti, joten kehittäjän ei tarvitse tehdä muuta kuin ohjeistaa sovellus tietynlaista konfiguraatiomallia kohti. (Spring 2017.)

Spring Framework ja Spring Boot pohjautuvat Java-ohjelmointikieleen, tuettuja käännösjärjestelmiä ovat Maven ja Gradle (Spring 2017).

5.3 AngularJS

AngularJS on vapaan lähdekoodin ohjelmistokomponenttikirjasto, joka on tarkoitettu helpottamaan dynaamisten Web-sovellusten kehitystä. Se antaa kehittäjälle vapauden käyttää HTML-kieltä sivuston pohjan luomiseen ja laajentaa HTML:n toiminnallisuutta dynaamisemmaksi antamalla mahdollisuuden kirjoittaa HTML-komponentteja Javascript-kielellä. Se myös tuo kehittäjälle käyttöön AngularJS:n omat HTML-direktiivit jotka mahdollistavat muun muassa for-silmukan upottamisen suoraan HTML-pohjaan, sekä tekee sovelluksen Javascript-puolen datan linkityksen HTML-elementteihin helpommaksi. (AngularJS [Viitattu 18.2.2017].)

5.3.1 Direktiivit

AngularJS siis muuttaa staattisten sivustojen kehitykseen tarkoitetun HTML-kielen ominaisuuksiltaan sopivammaksi dynaamisten Web-sovellusten kehityksen suhteen. Näitä HTML-kielen laajennuksia sanotaan AngularJS:n mukaisesti direktiiveiksi. (AngularJS [Viitattu 18.2.2017].)

Direktiiveille on aina määriteltävä HTML-pohja, se edustaa direktiivin ulkoista vaikutusta näkymään. HTML-pohja sisältää yleensä direktiivin sisäiset muuttujat sekä hieman sisäistä logiikkaa pohjan piirtämistä varten. Vakiona Angularista löytyvät seuraavat sisäänrakennetut direktiivit ja ominaisuudet:

- Datan kytkentä HTML-elementteihin, muotoa {{muuttuja}}.
- HTML-elementit DOM-manipulaatioon, esimerkiksi silmukat ja elementtien piilotus, muotoa .
- HTML-lomakkeiden datan oikeellisuuden tarkistus.
- HTML-elementtien tapahtumien käsittely.
- HTML-pohjien luominen uudelleenkäytettäviksi, ylemmän abstraktiotason komponenteiksi. (AngularJS [Viitattu 18.2.2017].)

5.3.2 Rakenne

AngularJS noudattaa *Model-View-Whatever*-mallia sillä toteutettavien sovellusten rakenteen suhteen. Se tarkoittaa arkkitehtuuria, joka muistuttaa *Model-View-Controller*-mallia, mutta antaa sovelluskehittäjälle vapauden päättää mallin controller-osuuden merkityksestä. *Whatever* mallin nimessä siis tarkoittaa *whatever works for you*. (AngularJS [Viitattu 15.3.2017].)

Pohjimmiltaan Angularilla tehty sovellus koostuu moduuleista. Moduuleja on montaa eri tyyppiä, ja sovelluskehittäjä pystyy luomaan omia moduuleitaan täysin vapaasti. Jokaiselle moduulille tulee määritellä lista muista moduuleista sekä metodeista, joista kyseinen moduuli on riippuvainen. (AngularJS [Viitattu 15.3.2017].)

angular.module: Moduulien tarkoitus on edustaa esimerkiksi kokonaista sovellusta tai pientä osaa sovelluksesta. Moduulit sisältävät riippuvuuksina alimetodeja joiden mahdolliset eri tyypit on lueteltu seuraavassa. Moduuli injektoidaan sovellukseen yleensä *ng-app*-direktiivillä esimerkiksi -elementin sisällä. (AngularJS [Viitattu 15.3.2017].)

angular.module.factory: Factoryt ovat metodeja, jotka sisältävät yksityistä dataa sekä alimetodeja ja palauttavat joukon alimetodeja joita taas puolestaan voidaan kutsua palauttamaan dataa. (AngularJS [Viitattu 15.3.2017].)

angular.module.service: Servicet ovat saman kaltaisia kuin Factoryt. Ne eroavat toisistaan lähinnä siten, että sovelluksen sisäiset palvelumoduulit ovat singleton-instansseja, joissa sisäinen data ja metodit ovat itse palvelu-instanssin jäseniä. Palvelu voi esimerkiksi edustaa rajapintaa, jonka metodeilla kommunikoidaan REST-rajapinnan kanssa. (AngularJS [Viitattu 15.3.2017].)

angular.module.directive: Directivet edustavat käytännössä sovelluksen sisäisiä kustomoituja HTML-direktiivejä. Kehittäjällä on siis mahdollisuus angular-direktiivien kautta luoda omia HTML-elementtejä, jotka piirretään DOM-objektiin niiden sisältämän logiikan ja datan mukaisesti. (AngularJS [Viitattu 15.3.2017].)

angular.module.filter: Metodeja, jotka ottavat sisääntulona dataa sekä vaihtoehtoisia parametreja, suodattavat datan moduulin sisäisillä ehdoilla ja palauttavat datan suodatettuna. (AngularJS [Viitattu 15.3.2017].)

angular.module.controller: Controllerit ovat kuin olio-ohjelmointikielen luokkia. Controller-instanssi liitetään aina johonkin näkymään, ja sillä on suora yhteys näkymän *$scope*-muuttujaan. Niiden tehtävänä on hallita näkymän dataa käyttäen hyväksi sovelluksen palveluita sekä muita moduuleja. (AngularJS [Viitattu 15.3.2017].)

6 OPC UA Data -esimerkkisovellus

Tässä luvussa kuvataan opinnäytetyötä varten tehty esimerkkisovellus. Esimerkkisovellus koostuu seuraavista ohjelmista:

Gateway: Toimii tiedonvälittäjänä PLC-laitteiden ja tietokannan välillä. PLC-viestintä tapahtuu OPC UA -protokollaa hyödyntäen ja tietokantaa käytetään REST-rajapinnan kautta HTTP-protokollalla.

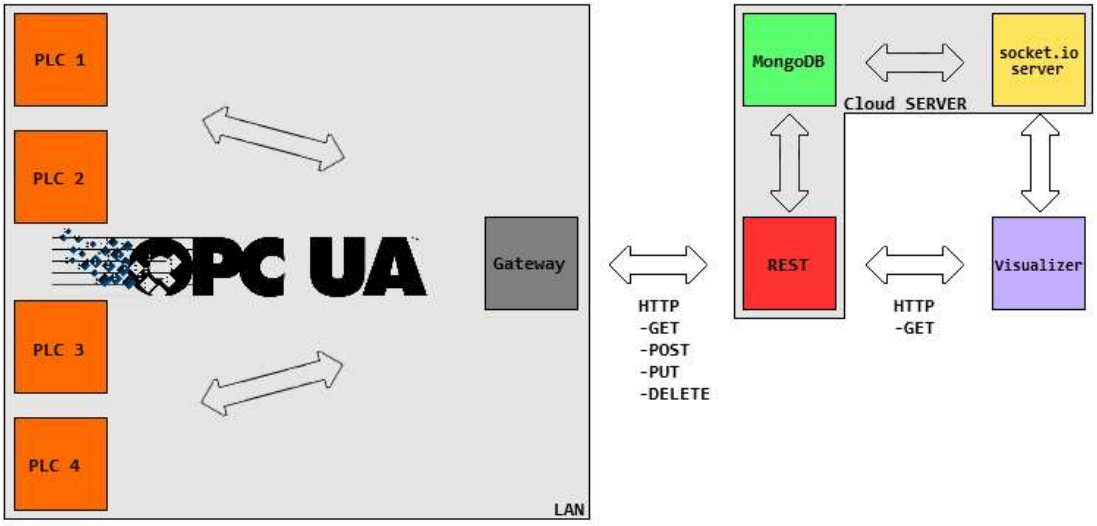
REST-palvelu: Autentikoi datan lähettäjän, varmistaa datan oikeellisuuden ja tallentaa varmistetun datan MongoDB-tietokantaan. Toteuttaa rajapintametodit myös tietokantadatan päivittämistä, poistamista, sekä hakemista varten.

Socket.io-palvelin: Mahdollistaa reaaliaikaisen katkeamattoman tiedonluvun MongoDB-tietokannasta websocket-yhteyden avulla.

Data Visualizer: Web-pohjainen käyttöliittymäsovellus, jonka avulla loppukäyttäjä voi tarkastella tietokantaan tallennettua dataa niin historiamuodossa kuin reaaliaikaisesti piirtyvänä X/Y-graafina.

6.1 Rakenne

Teollisen internetin esimerkkisovellus koostuu neljästä erillisestä ohjelmasta ja MongoDB-tietokannasta.



Kuva 7. Teollisen internetin esimerkkisovelluksen rakenne

Kuvassa 7 on kuvattu esimerkkisovelluksen arkkitehtuuria. Yksittäinen neliö edustaa joko erillistä tietokoneohjelmaa tai laitetta.

Gateway sijaitsee SeAMKin teollisen internetin laboratorion tapauksessa samassa lähiverkossa laboratorion PLC-laitteiden kanssa. Tämä on ideaali tilanne, joka mahdollistaa nopean viestinnän järjestelmän kriittisimmässä kohdassa.

Gateway-palvelun rooli tässä tapahtumaketjussa on lukea PLC-laitteiden OPC UA -kohdepalvelimilta dataa, välittäen samalla tätä luettua dataa HTTP-protokollalla REST-palvelulle. Alustusvaiheessa Gateway myös synkronoi konfiguraatiotietonsa REST-palvelun kautta tietokantaan, näitä tietoja käyttää myöhemmin hyödyksi Visualizer-sovellus.

REST-palvelu sijaitsee erillisellä palvelimella ulkoisessa verkossa. Samalla palvelimella on myös MongoDB-tietokanta ja Socket.io-palvelin. REST autentikoi jokaisen pyynnön, ainoastaan datan lukemispyynnöt eivät vaadi käyttäjätunnukselle erityisoikeuksia. REST myös olettaa datan tulevan JSON-muotoisena ja vastaa pyyntöihin

itse JSON-muotoisilla viesteillä. REST-palvelun tietolähteenä toimii MongoDB-tietokanta.

Socket.io-palvelin odottaa yhteyksiä Visualizer-sovelluksen puolelta. Se vastaa yksinkertaisiin loppukäyttäjän pyyntöihin palauttamalla pyynnön mukaisesti dataa MongoDB-tietokannasta.

Visualizer-sovellus sijaitsee samalla palvelimella REST-, MongoDB- ja Socket.io-palvelujen kanssa. Se on loppukäyttäjän selaimessa suoritettava sovellus, jonka takia sillä tehdyt pyynnöt tulevat lopulta loppukäyttäjän omasta IP-osoitteesta. Tämän takia se on kuvassa 7 eristettynä muista verkoista. Visualizer antaa loppukäyttäjälle mahdollisuuden lukea MongoDB-tietokannassa olevaa dataa joko REST-palvelulle tehtävien pyyntöjen avulla tai suoralla websocket-yhteydellä socket.io-palvelimen kautta. Datan visualisointi tapahtuu piirtämällä luettujen muuttujien arvot X/Y-graafille, dataa voi piirtää graafille ajan suhteen joko reaaliajassa tai historiamuodossa. Historiadataa on mahdollista suodattaa esimerkiksi aikaleimarajoilla.

Alusta lähtien ideana oli luoda yleispätevä ohjelmistokokonaisuus, jonka yksittäiset osat eivät ole täysin riippuvaisia toisistaan. Gateway-palvelu yhdistää konfiguraatiotiedoissa annettuihin OPC UA -palvelimiin ja alkaa lukemaan palvelimilla päivittyvää dataa. Samalla luettu data lähetetään HTTP-protokollalla ohjelman konfiguraatiotiedoissa määritellylle kohdepalvelulle. Ohjelman olettamat takaisin tulevat vastaukset REST-palvelulta ovat hyvin vähäiset ja yksinkertaiset, mikä mahdollistaa kyseisen kohdepalvelun korvaamisen toisella ohjelmistolla kohtuullisen helposti. Samoin REST-palvelu toimii vain mustana laatikkona, joka olettaa tietyt rakenteet sille lähetetyissä JSON-muotoisissa viesteissä, ja vastaa näihin viesteihin JSON-objektien muodossa. Kaiken kaikkiaan tämä kokonaisuus on siis hyvin modulaarinen ja yksittäisiä ohjelmistoja tässä ketjussa pitäisi olla mahdollista korvata kokonaan uusilla sovelluksilla suhteellisen vaivattomasti.

6.2 Gateway

Gateway-ohjelmiston tavoite on toimia palveluna, joka koostuu listasta OPC UA –client -instansseja ja yksittäisestä HTTP-client-instanssista. Ohjelma lukee aluksi annetun konfiguraation JSON-tiedostosta, jonka jälkeen se yhdistää OPC UA -clienteillä kaikille konfiguraatiossa määritellyille palvelimille. Kun yhteydet toimivat, ohjelma tilaa näiltä palvelimilta konfiguraatiossa määritellyiltä muuttujilta arvon muutostapahtumat. Samalla nämä tilatut tapahtumat linkitetään funktioon, joka luo arvoa vaihtaneesta muuttujasta JSON-objektin ja lähettää tämän koko JSON-objektin konfiguraatiossa määritellylle REST-palvelulle käyttäen HTTP-client-luokan instanssia.

Sovellus on kirjoitettu C++-kielellä, käyttäen hyväksi kielen C++11-version sisältämiä ominaisuuksia. Se hyödyntää Open62541-kirjastoa OPC UA -yhteyksiä varten, libcurl-kirjastoa HTTP-pyyntöjen teossa, sekä JSON for Modern C++ -single-header -kirjastoa JSON-datan käsittelyä varten.

6.2.1 Konfigurointi

Ohjelma lukee alustusvaiheessa ”./res/settings.json”-tiedostosta konfiguraation, joka määrittelee sen, miten ohjelman halutaan toimivan. Tässä alaluvussa käsitellään tuon tiedoston eri tietueiden vaikutusta ohjelman toimintaan. Kuvassa 8 on kuvattuna konfiguraatiotiedoston rakenne.

```
{
  "ua_service_config": {
    "quit_on_error":  true
  },
  "ua_rest_config": {
    "endpoint": "http://harha.us.to:9090",
    "username": "opc_ua_data_rest_admin",
    "password": "password",
    "output": "./res/libcurl_log.log",
    "verbose": false
  },
  "ua_client_config": [
    {
      "serverId": 10,
      "endpoint": "opc.tcp://harha-Laptop:48050",
      "identifier": "Beckhoff OPC UA, Toni Laptop",
      "username": "",
      "password": "",
      "subPublishInterval": 10,
      "subPublishPriority": 1,
      "subscriptions": [
        {
          "isFolder": true,
          "nsIndex": 6,
          "identifiers": [ "MAIN" ]
        }
      ]
    }
  ]
}
```

Kuva 8. Gateway-palvelun JSON-konfiguraatiotiedosto

Tiedosto edustaa yhtä JSON-objektia, joka koostuu pohjimmiltaan kolmesta eri pää-objektista, ”ua_service_config”, ”ua_rest_config” ja ”ua_client_config”.

- ua_service_config: JSON-objekti
  - quit_on_error: boolean-muuttuja. Määrittelee sen, suljetaanko palvelu, jos jossain OPC UA -clientissä tapahtuu virhe.
- ua_rest_config: JSON-objekti
  - endpoint: string-muuttuja. REST-palvelun URL-osoite.
  - username: string-muuttuja. REST-palvelin basic-auth-käyttäjätunnus.
  - password: string-muuttuja. REST-palvelun basic-auth-salasana.
  - output: string-muuttuja. libcurl-kirjaston virheviestien kohdetiedosto.
  - verbose: boolean-muuttuja. Lokituksen taso, määrittelee sen, minkä tasoiset HTTP-clientin viestit kirjoitetaan output-tiedostoon sekä palvelun ikkunaan.

- ua_client_config: lista JSON-objekteja
  - serverId: int-muuttuja. OPC UA -kohdepalvelimen uniikki identiteetti REST-palvelun tietokannassa.
  - endpoint: string-muuttuja. OPC UA -kohdepalvelimen pääteosoite.
  - identifier: string-muuttuja. OPC UA -kohdepalvelimen mielivaltainen nimi.
  - username: string-muuttuja. OPC UA -kohdepalvelimen käyttäjätunnus.
  - password: string-muuttuja. OPC UA -kohdepalvelimen salasana.
  - subPublisInterval: double-muuttuja. OPC UA -kohdepalvelimelta tilattujen muuttujien arvonmuutostapahtumien päivitysväli.
  - subPublishPriority: byte-muuttuja (8-bit). OPC UA -kohdepalvelimelta tilattujen muuttujien arvonmuutostapahtumien päivitysprioriteetti.
  - subscriptions: lista JSON-objekteja
    - isFolder: boolean-muuttuja. Onko tilattava muuttujanoodi ’kansio’, joka sisältää alinoodeja vai yksittäinen noodi. Jos kansio, tilaa client arvonmuutostapahtumien päivitykset tämän noodin kaikilta alinoodeilta. Muuten tilataan vain kyseisen noodin tapahtumat.
    - nsIndex: int-muuttuja. Tilattavien noodien nimiavaruus.
    - Identifiers: lista stringejä. Kaikkien tilaukseen kuuluvien noodien nimi, joka on uniikki valitussa nimiavaruudessa. Jos isFolder on ’true’, niin tähän tarvitsee määritellä vain noodit, joiden kaikki alinoodit tilataan.

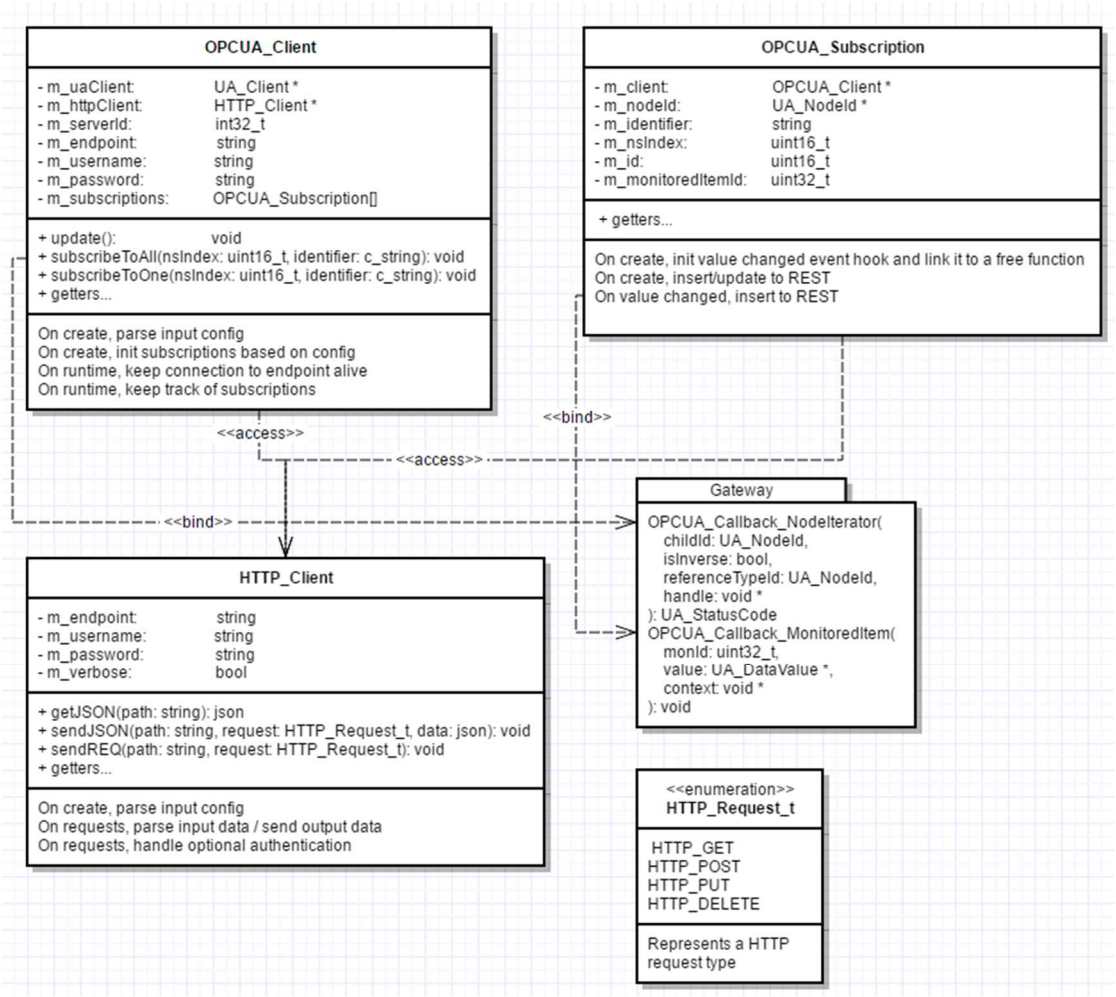
6.2.2 Rakenne ja toiminta

Ohjelma koostuu kolmesta eri pääkomponentista, OPCUA_Client-, OPCUA_Subscription- sekä HTTP_Client-luokista.

OPCUA_Client-luokan instanssien tehtävänä on edustaa asiakas-palvelin yhteyttä, eli jokaista erillistä kohdepalvelinta kohti on olemassa yksi OPCUA_Client-instanssi.

OPCUA_Subscription-luokan instanssit edustavat OPCUA_Client-olion noodi-tilausta, tarkemmin sanottuna OPC UA -palvelimella sijaitsevan tietyn noodin arvon muutostapahtumien tilausta.

HTTP_Client-luokka toimii helppokäyttöisenä rajapintana JSON-muotoisten pyyntöjen lähetykselle REST-palvelulle. Tästä luokasta on vain yksi instanssi koko sovelluksessa, jokainen client- sekä subscription-olio omaa jäsenmuuttujana osoittimen tähän yksittäiseen instanssiin. Kuvassa 9 on kuvattuna tämän ohjelman rakenne ja luokkien relaatiot keskenään.



Kuva 9. UML-luokkakaavio Gateway-sovelluksen olioiden yhteyksistä

6.3 REST-rajapinta

REST-palvelu tätä ohjelmistokokonaisuutta varten luotiin Spring Boot -sovelluskehyksellä Palvelun ideana on toimia HTTP-protokollan avulla käytettävänä rajapintana, jonka avulla voidaan lukea, tallentaa ja poistaa dataa yksittäisestä MongoDB-tietokannasta.

Rajapinta on kirjoitettu Java 8 -kielellä käyttäen Spring Boot -sovelluskehyksen lisäksi Apache Maven -pakettienhallinta / käännösohjelmistoa, lukuisia Spring Boot -alikirjastoja sekä joda-time- ja json-path-kirjastoja.

6.3.1 Konfigurointi

Ohjelman käännetty JAR-paketti sisältää application.properties-tiedoston, josta se lukee alustuksen aikana parametreja. Jos käyttäjä haluaa vaikuttaa ohjelman konfiguraatioon, niin kyseisestä tiedostosta voidaan luoda kopio, joka asetetaan samaan kansioon suoritettavan JAR-paketin kanssa. Tämä kopio ylikirjoittaa JAR-paketin sisällä olevan application.properties-tiedoston arvot.

```
# REST
spring.application.name=opc_ua_data_rest
app.encoding=@project.build.sourceEncoding@
app.java.version=@java.version@
server.port=9090

# MongoDB
spring.data.mongodb.uri=mongodb://harha.us.to:27017/iotgateway

# Security
authentication.adminUser=opc_ua_data_rest_admin
authentication.adminPass=password
authentication.readOnlyUser=opc_ua_data_rest_readonly
authentication.readOnlyPass=password

# JSON serialization
spring.jackson.serialization-inclusion=non_null

# Logging
logging.file=logs/opc_ua_data_rest.log
logging.level.root=INFO
```

Kuva 10. REST-palvelun application.properties-konfiguraatiotiedosto

- REST
  - spring.application.name: Palvelun nimi.
  - app.encoding: Tekstin enkoodaustapa, määritellään pom.xml:ssä käännösvaiheessa, oletuksenta UTF-8.
  - app.java.version: Java-versio, määritellään pom.xml:ssä käännösvaiheesa, oletuksena Java 8.
  - server.port: Portti, jonka alla palvelu odottaa HTTP-kutsuja.
- MongoDB
  - spring.data.mongodb.uri: Kohdetietokannan URI, sisältää osoitteen, portin sekä mahdolliset lisäparametrit, kuten tietokannan nimen, käyttäjätunnuksen ja salasanan.
- Security
  - authentication.adminUser: Pääkäyttäjän tunnus.
  - authentication.adminPass: Pääkäyttäjän salasana.
  - authentication.readOnlyUser: Read only -käyttäjän tunnus.
  - authentication.readOnlyPass: Read only -käyttäjän salasana.
- JSON Serialization
  - spring.jackson.serialization-inclusion: Huomioidaanko NULL-arvoiset parametrit JSON-objektien serialisoinnissa.
- Logging
  - logging.file: Polku lokitustiedostoon.
  - logging.level: Lokitettavien viestien alimmaistaso, mahdolliset arvot: FATAL, ERROR, WARN, INFO, DEBUG, TRACE.

6.3.2 Rakenne ja toiminta

Ohjelman arkkitehtuuri rakentuu Spring Boot REST -projektille tyypillisellä tavalla, eli controller-, model-, service- ja repository-luokista. Jokaisella luokalla on oma tehtävänsä HTTP-kutsun käsittelyprosessissa. Controllerit ottavat kutsut vastaan, servicet sisältävät varsinaisen bisneslogiikan, joka määrittelee mitä millekin kutsulle tehdään, repositoryt antavat serviceille pääsyn käsiteltävien resurssien tietokantaan ja model-luokat edustavat kutsuissa käsiteltäviä resursseja.

Kun ohjelma käynnistetään, se konfiguroi itsensä osittain automaattisesti ja osittain käyttäen aiemmin mainittiua application.properties-tiedostoa. Se tyhjentää käyttäjätunnukset tietokannan opc_ua_user_account-kokoelmasta ja lisää sinne uudet admin- ja read-only-tunnukset. Näiden tunnuksien käyttäjänimi sekä salasana on määriteltävissä kyseisessä konfiguraatiotiedostossa. Salasanat salataan Bcryptillä, koska paljaan tekstimuotoisen salasanan tallentaminen tietokantaan ei ole järkevää tietoturvallisista syistä.

Controllerit edustavat rajapinnan päätason päätteitä, joita on kolme.

- /opcuaservers
  - OPCUAServer-objektien käsittely
- /opcuasubscriptions
  - OPCUASubscription-objektien käsittely
- /opcuavariables
  - OPCUAVariable-objektien käsittely

Controllerit kutsuvat vastaavan servicen metodeja parametreilla, jotka liittyvät käsiteltävään HTTP-kutsuun. Servicejä tässä projektissa on neljä. Servicet sisältävät metodeja, joilla esimerkiksi voidaan lukea, tallentaa, muuttaa tai poistaa tietokannasta sen tyyppisiä dokumentteja. Kaikkien näiden toimenpiteiden edessä on yleensä logiikka, joka varmistaa, ettei rajapintaa voida käyttää väärin.

- OPCUAServerService
  - Tietokantakommunikaatio OPCUAServerRepository-instanssin kautta
- OPCUASubscriptionService
  - Tietokantakommunikaatio OPCUASubscriptionRepository-instanssin kautta
- OPCUAVariableService
  - Tietokantakommunikaatio OPCUAVariableRepository-instanssin kautta
- UserAccountService
  - Tietokantakommunikaatio UserAccountRepository-instanssin kautta

Repositoryt eivät yleensä sisällä minkäänlaista bisneslogiikkaa pakollisten varmistusten lisäksi, vaan ne antavat käyttäjälleen suhteellisen suoran pääsyn tietokannan kokoelmassa olevaan dataan. Näitä on myös neljä, jokaista serviceä varten yksi. Ne ovat rajapintaluokkia, joiden metodeilla ei ole sisältöä, ja perivät MongoRepository-luokan. MongoRepository-luokka huolehtii siitä, että repositorystä luodaan automaattisesti toimiva instanssi ohjelman suorituksen aikana sen sisältämien metodien nimien, palautustyyppien ja argumenttien perusteella.

6.3.3 Turvallisuus

Rajapinta suojaa tietyt kutsut käyttöoikeuksien taakse Spring Boot Securityn WebSecurityConfigurerAdapter- sekä GlobalAuthenticationConfigurerAdapter-luokkien avulla. Oletuksena rajapinnan päätepisteet on suojattu siten, että kuka tahansa pystyy lukemaan tietoja käyttäjätunnuksella tai ilman, mutta ainoastaan ”ADMIN”-roolin omaavat käyttäjät pystyvät tekemään tietokannan dataa muokkaavia kutsuja.

Käyttäjän autentikointi tapahtuu Basic access authentication -menetelmän avulla HTTP-transaktioiden yhteydessä, eli yhden onnistuneen kirjautumisen jälkeen saa palvelimen generoiman tokenin, jonka voi laittaa seuraaviin kutsuihin header-osioon.

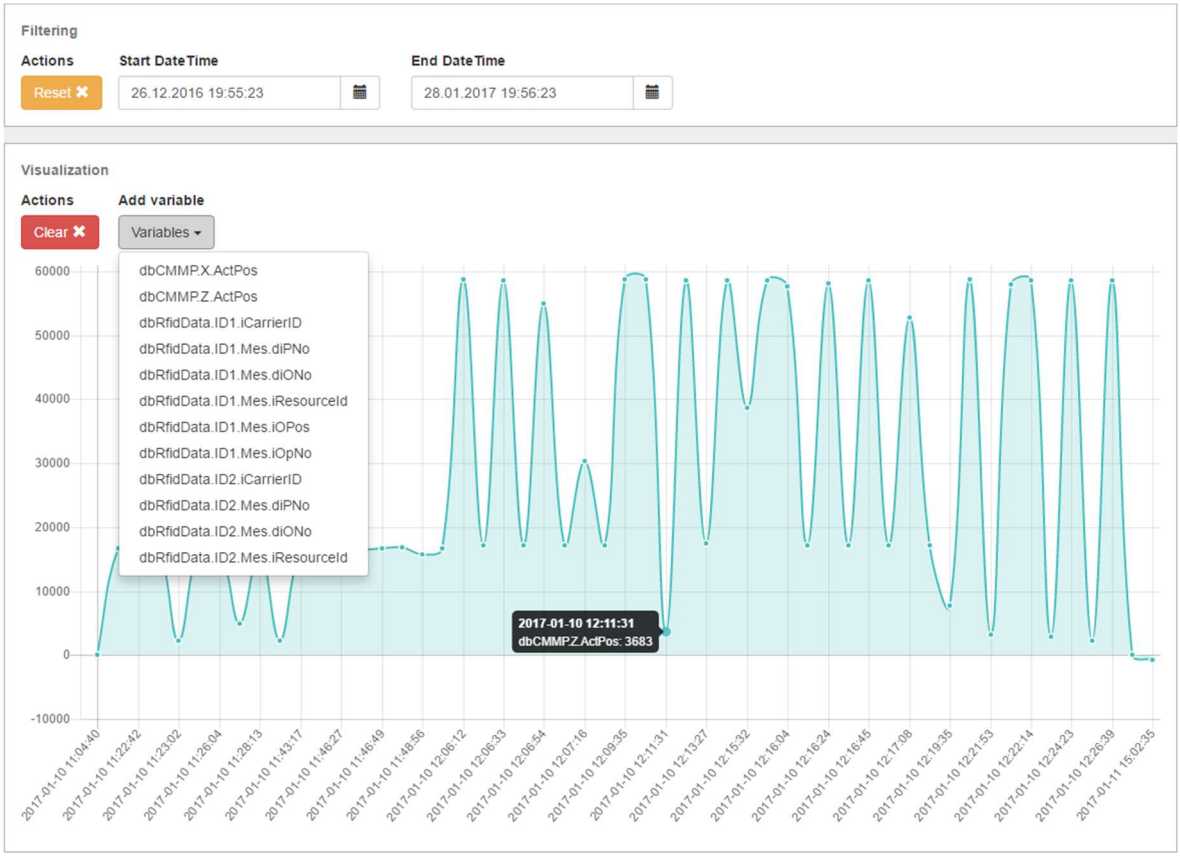
6.4 Websocket-palvelin

Websocket-palvelin antaa vaihtoehtoisen, reaaliaikaisen ratkaisun REST-rajapinnalle. Palvelun ideana on toimia yksinkertaisena datan toimittajana asiakasohjelmalle, joka pyytää palvelimelta haluttujen muuttujien arvonmuutostapahtumat. Tapahtuman tilaamisen jälkeen palvelin alkaa toimittamaan tapahtumadataa asiakasohjelmalle saman, jatkuvan socket-yhteyden kautta.

Palvelinohjelmisto on kirjoitettu Javascript-kielellä käyttäen socket.io websocket-kirjastoa. Sen täydellinen hyödyntäminen siis vaatii myös asiakasohjelman riippuvuudeksi socket.io-kirjaston.

6.5 Datan visualisoija

Kokonaisuuteen kuuluu myös web-sovelluksena luotu käyttöliittymä, joka mahdollistaa tietokannassa sijaitsevan muuttujadatan tarkastelun X/Y-graafilla aikaleimarajojen, sekä muuttujanimen perusteella suodatettuna. Sillä pystytään myös tarkastelemaan yhtä muuttujaa kerralla reaaliaikaisesti websocket-yhteyden ansiosta.



Kuva 11. OPC UA Data Visualizer -sovelluksen käyttöliittymä, SeAMK teollisen internetin laboratorion varastoaseman Z-koordinaatti piirrettynä kuvaajaan ajan suhteen.

Käyttöliittymä on kirjoitettu HTML-sivustona, joka hyödyntää AngularJS 1.5.8 Javascript -sovelluskehystä, sekä Bootstrap 3.3.7 CSS/Javascript -kirjastoa.

6.5.1 Konfigurointi ja kääntäminen

Projekti käyttää npm-pakettienhallintaa riippuvuuksien määrittelyyn ja automatisoitua käännöstoiminnallisuutta varten. Kaikki käännöstehtävät ja riippuvuudet on

määritelty juurikansiossa sijaitsevaan package.json-tiedostoon. Käännetty paketti luodaan ./dist -hakemistoon.

Ohjelmistossa on konfiguroitavissa ennen käännöstä vain REST- sekä Websocket-palveluiden osoitteet ja graafille piirrettävien arvojen maksimimäärä reaaliaikaisessa socket-yhteydessä. Näiden arvot löytyvät ./app/app.js-lähdekooditiedostosta rest_url-, serv_url- ja chart_max_values-muuttujien nimillä.

6.5.2 Rakenne ja toiminta

Datan visualisoija on hyvin yksinkertainen AngularJS-sovellus, tarkoituksena on vain demonstroida datan hakemista sekä visualisointimahdollisuuksia REST-palvelun tai Websocket-palvelimen kautta. Se on siis eräänlainen esimerkkisovellus.

Ohjelman rakenne on AngularJS 1.x.x -sovellukselle hyvin tyypillinen. Palvelut antavat rajapintametodit, joiden kautta dataa voidaan hakea kohdepalvelimilta, controllerit taas sisältävät sovelluksen käyttöliittymälogiikan. Controllereita ja näiden sisältämiä jäsenmuuttujia sekä funktioita voi upottaa HTML-tiedostoihin HTML-direktiivien sekaan, mikä mahdollistaa reaaliaikaisen datan muuttamisen sivun näkymässä ilman selaimessa auki olevan sivuston päivittämistä.

7 Jatkokehitysideat

Työn lopputuloksena on vakaasti toimiva ohjelmistokokonaisuus, joka täyttää toiminnallisuuksiltaan työlle asetetut alkuperäiset tavoitteet. Tästä huolimatta mahdollisia lisätoimintoja on silti useita. Tässä luvussa käsitellään niitä ja pohditaan, että mihin suuntaan näitä työssä syntyneitä ohjelmistoja voisi jatkokehittää. Nämä samat jatkokehitysideat löytyvät myös jokaisen ohjelman github-repositorystä issueina.

7.1 Lähdekoodit githubissa

Jokaisen työssä syntyneen sovelluksen lähdekoodit ovat saatavilla github-repositoryjen kautta.

Sovellusten käännetyt versiot löytyvät myös Github-projekteista ”releases”-alueelta. Nämä ovat julkaistuja ”vakaita” versioita ohjelmistoista ja niiden toiminnallisuus sekä rakenne voi poiketa varsinaisen versionhallinnan sisältämän uusimman version ominaisuuksista.

- OPC UA Data Gateway
  - https://github.com/Harha/OPC-UA-Data-Gateway
  - https://github.com/Harha/OPC-UA-Data-Gateway/releases
  - Visual Studio Project, C++14
  - .exe (x86)
- OPC UA Data REST
  - https://github.com/Harha/OPC-UA-Data-REST
  - https://github.com/Harha/OPC-UA-Data-REST/releases
  - Apache Maven Project, Java 8
  - .JAR (Java 8)
- OPC UA Data Server
  - https://github.com/Harha/OPC-UA-Data-Server
  - NPM Project, Javascript, Socket.io
- OPC UA Data Visualizer
  - https://github.com/Harha/OPC-UA-Data-Visualizer
  - https://github.com/Harha/OPC-UA-Data-Visualizer/releases

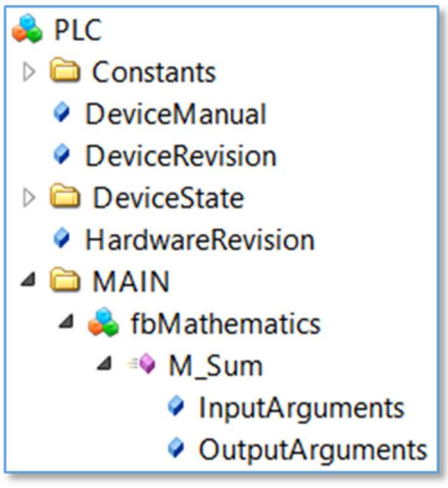
- NPM Project, Javascript, AngularJS 1.5.8
- www-data (HTML & Javascript)

7.2 Gateway, tuki PLC-metodikutsuille OPC UA:n kautta

OPC UA tukee PLC-ohjelman funktioiden kutsumista, näille OPC UA:n kautta tehtäville kutsuille voidaan antaa parametreja, ja ne voivat palauttaa dataa tai vain suorittaa jonkun rutiinin ilman palautusarvoa. Tämä toiminnallisuus on riippuvainen OPC UA -palvelimen sekä siihen kytketyn PLC:n valmistajan omasta OPC UA -toteutuksesta ja voi vaihdella eri valmistajien välillä.

7.2.1 Beckhoff TwinCAT 3 metodikutsu

TwinCAT 3 antaa mahdollisuuden suorittaa PLC-metodeja RPC-kutsuina OPC UA -yhteyden ylitse. Metodit näkyvät UA-nimiavaruudessa tavallisina OPC UA -metodeina, mutta ne suoritetaan silti reaaliaikaisesti kohde-PLC-laitteella IEC61131-standardin mukaisina metodeina. (Beckhoff [Viitattu 2.2.2017].)



Kuva 12. IEC61131-3-metodi (M_Sum) Beckhoffin OPC UA -palvelimella (Beckhoff [Viitattu 2.2.2017]).

IEC61131-3-standardin metodit konfiguroidaan aina PLC-ohjelmassa Function Blockin sisälle, ne voidaan siis kuvitella FBD:n jäsenfunktioina. Jotta metodia voitaisiin kutsua etäkutsuna OPC UA:n kautta, sille pitää antaa attribuutti *TcRpcEnable*, joka määrittelee tukeeko metodi RPC-kutsuja vai ei. (Beckhoff [Viitattu 2.2.2017].)

```
{attribute 'TcRpcEnable':='1'}
METHOD M_Sum : INT
VAR_INPUT
  a : INT;
  b : INT;
END_VAR
```

Kuva 13. Esimerkkimetodi FBD:n sisällä TwinCAT 3 -ohjelmassa. (Beckhoff [Viitattu 2.2.2017]).

7.3 Gateway, Lua-komentosarjakieli

Linkitetään Gatewayn kirjastoriippuvuudeksi Lua-komentosarjakielen tulkki dynaamisesti linkitettynä DLL-kirjastona. Tämä mahdollistaa lua-komentosarjojen suorituksen Gateway-ohjelman suorituksen ohella esimerkiksi OPC UA -tapahtumien aikana.

7.3.1 Lua

Lua on nopea ja kevyt muihin sovelluksiin upotettava komentosarjakieli. Lua tukee mm. proseduraalista, oliopohjaista, funktionaalista ja datapohjaista ohjelmointityyliä. Se on kirjoitettu C-kielellä hyvin kompaktissa paketissa. Käännetty paketti on kooltaan niin pieni, että se soveltuu jopa sulautettujen järjestelmien mikroprosessoreille suoritettavaksi. (Lua 31.1.2017a.)

Lua 5.0 ja sitä uudemmat versiot on lisensoitu MIT-lisenssin mukaisesti, kuten on kaikki tämän työn ohjelmistotkin, joten ongelmia lisensoinnin suhteen ei ole (Lua 31.1.2017b).

7.3.2 Lua-rajapinta

Lua-puolen implementaatiot luodaan Gatewayn C++-luokista ja niiden jäsenmetodeista. Tämä onnistuu käsittelemällä C++-olioita Lua-komentosarjojen puolella taulukoina.

Määritellään myös ohjelman alustusvaiheessa kerran suoritettava Lua-komentosarja. Se antaa käyttäjälle vapaat kädet ohjelman konfigurointiin liittyen. Tämä komentosarjatiedosto ja kyseisen tiedoston polku voisivat olla muunneltavissa ohjelman settings.json-konfiguraatiotiedostossa ua_lua_config JSON-objektin alla.

7.3.3 Lua-tapahtumat ja callbackit

Hyödyllinen lisäominaisuus voisi myös olla tuki tapahtumille Lua-puolella. Alustusvaiheessa suoritettavassa Lua-komentosarjassa voitaisiin esimerkiksi määritellä tietyille tapahtumatyypeille omat Lua-komentosarjat, jotka käsittelevät näitä tapahtumia.

C++-ohjelman puolella tulisi vain implementoida tietynlaisten tapahtumien ilmoitus Lua-tulkille. Hyödyllisiä tapahtumia voisivat esimerkiksi olla OPC UA -muuttujan arvon vaihtuminen ja tietyn PLC-metodin kutsuminen. Tämä mahdollistaisi näiden tapahtumien lisäkäsittelyn käyttäjän oman Lua-komentosarjan toimesta, palvelu voisi esimerkiksi puskea muuttujien arvoja jopa toisen REST-palvelun rajapinnalle Lua-komentosarjan kautta.

8 Yhteenveto ja tulokset

Työn suunnitteluvaihe koostui suurimmaksi osaksi sovellusten välisen tiedonsiirtoprotokollien valitsemisesta, toteutukseen käytettävien ohjelmointikielten sekä teknologioiden valinnasta, sekä kokonaisuuden rakenteen päättämisestä. Toteutusosuus sujui suhteellisen vaivattomasti, eikä suurempia muutoksia alkuperäiseen suunnitelmaan tarvinnut tehdä.

Opinnäytetyön lopputuloksena on kolmiosainen sovelluskokonaisuus, joka mahdollistaa OPC UA -palvelimen sisältämän informaation reaaliaikaisen päivityksen pilvipalvelun tietokantaan. Järjestelmä sisältää myös visualisointisovelluksen web-pohjaisella käyttöliittymällä. Työn päätavoitteena oli saada tämä informaation tallennusprosessi toimimaan Seinäjoen Ammattikorkeakoulun teollisen internetin laboratorion laitteiston kanssa, ja tavoite saavutettiin. Ohjelmistot toteutettiin siten, että ne ovat yleiskäyttöisiä, eivätkä pelkästään teollisen internetin laboratorion laitteistolle ja arkkitehtuurille räätälöityjä.

Mutta kuten jo ”Jatkokehitysideat”-luvusta voidaan päätellä, on työn tuloksia silti mahdollista kehittää eteenpäin vielä runsaasti. Tätä varten ohjelmistojen lähdekoodit löytyvät Github-palvelusta git-versionhallinnan alaisina kansioina. Näin kuka tahansa voi kopioida ohjelmistojen lähdekoodit tietokoneelleen, tehdä niihin muutoksia ja lähettää sitten muutokset Github-projektin ylläpitäjien hyväksyttäviksi. Projektin ylläpitäjä puolestaan vetää muilta käyttäjiltä saadut muutokset mukaan Github-projektin master-haaraan tai hylkää ne toistaiseksi antaen samalla palautetta hylkäämisen syystä.

LÄHTEET

AngularJS. Ei päiväystä. Developer Guide. [www-dokumentti]. Google, Inc. [Viitattu 15.3.2017]. Saatavissa: https://docs.angularjs.org/guide

AngularJS. Ei päiväystä. What is AngularJS?. [www-dokumentti]. Google, Inc. [Viitattu 18.2.2017]. Saatavissa: https://docs.angularjs.org/guide/introduction

Beckhoff. Ei päiväystä. Method Call. [www-dokumentti]. Beckhoff Automation GmbH & Co. KG. [Viitattu 2.2.2017]. Saatavissa: https://infosys.beckhoff.com/english.php?content=../content/1033/tf6100_tc3_opcua/1447358475.html&id=

Fielding, R. & Reschke, J. 2014. RFC-7231: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. Adobe. Greenbytes.

Fielding, R., Irvine, UC., Gettys, J., Mogul, J., Frystyk, H. & Berners-Lee, T. 2014. RFC-2068: Hypertext Transfer Protocol -- HTTP/1.1. MIT. DEC.

Fielding, R., Irvine, UC., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. & Berners-Lee, T. 1999. RFC-2616: Hypertext Transfer Protocol -- HTTP/1.1. MIT. W3C. Compaq.

Hovi, A. 2004. SQL Pikaopas. 12. painos. Jyväskylä: Docendo Oy.

Kepware. 25.9.2014. How OPC UA Protects Your Data. [www-dokumentti]. Kepware Technologies. [Viitattu 7.12.2016]. Saatavissa: https://info.kepware.com/blog/how-opc-ua-protects-your-data

Lua. 31.1.2017a. About. [www-dokumentti]. Lua.org. [Viitattu 2.2.2017]. Saatavissa: https://www.lua.org/about.html

Lua. 31.1.2017b. License. [www-dokumentti]. Lua.org. [Viitattu 2.2.2017]. Saatavissa: https://www.lua.org/license.html

MongoDB. Ei päiväystä. Document Databases. [www-dokumentti]. MongoDB Incorporation. [Viitattu 18.12.2016]. Saatavissa: https://www.mongodb.com/document-databases

Mäkelä, P. & Ristimäki, N. 2016. Seinäjoen ammattikorkeakoulun teollisen internetin laboratorio. Teoksessa: P. Junell, A. Heikkilä, S. Päällysaho & S. Saarikoski (toim.). Hyvinvointia ja innovaatioita monialaisesti ja raja-aitoja madaltaen. Katsaus Seinäjoen ammattikorkeakoulun toimintaan. Seinäjoen ammattikorkeakoulun julkaisusarja A. Tutkimuksia 25. 466-475.

Novotek. Ei päiväystä. OPC ja OPC UA. [www-dokumentti]. Novotek Oy. [Viitattu 7.12.2016]. Saatavissa: https://www.novotek.com/fi/ratkaisut/kepware-kommunikointialusta/opc-ja-opc-ua/

OPC Foundation. Ei päiväystä. Unified Architecture. [www-dokumentti]. OPC Foundation. [Viitattu 7.12.2016]. Saatavissa: https://opcfoundation.org/about/opc-technologies/opc-ua/

Open62541. 26.12.2016. open62541 Documentation. [www-dokumentti]. Open62541. [Viitattu 29.12.2016]. Saatavissa: http://open62541.org/doc/open62541-current.pdf

Oracle. Ei päiväystä. A Relational Database Overview. [www-dokumentti]. Oracle Corporation. [Viitattu 18.12.2016]. Saatavissa: https://docs.oracle.com/javase/tutorial/jdbc/overview/database.html

Prosys OPC. Ei päiväystä. What is OPC and OPC UA? [www-dokumentti]. Prosys PMS Ltd. [Viitattu 17.12.2016]. Saatavissa: https://www.prosysopc.com/opc/

Quva. Ei päiväystä. Yritysjohdon opas IoT:n ja teollisen internetin hyödyntämiseen. [www-dokumentti]. Quva Oy. [Viitattu 25.3.2017]. Saatavissa: http://quva.fi/site/attachments/yritysjohdon_opas_IoT_ja_teollisen_internetin_hyodyntamiseen.pdf

SeAMK. 9.3.2016. Teollisen internetin laboratorio Seinäjoelle. [www-dokumentti]. SeAMK Tekniikka. [Viitattu 21.3.2017]. Saatavissa: http://verkkolehti.seamk.fi/arkisto/maaliskuu-2016/teollisen-internetin-laboratorio-seinajoelle/

Spring. 2017. Spring Boot Reference Guide. [www-dokumentti]. Pivotal Software. [Viitattu 11.2.2017]. Saatavissa: http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/

Teknologiateollisuus. Ei päiväystä. Teollisesta internetistä uutta kasvua. [www-dokumentti]. Teknologiateollisuus ry. [Viitattu 3.4.2017]. Saatavissa: http://teknologiateollisuus.fi/fi/elinkeinopolitiikka/digitalisaatio/teollisesta-internetista-uutta-kasvua

Tutorials Point. Ei päiväystä. SQL - Overview. [www-dokumentti]. Tutorials Point Pvt. Ltd. [Viitattu 11.4.2017]. Saatavissa: https://www.tutorialspoint.com/sql/sql-overview.htm

Wapice. Ei päiväystä. OPC-järjestelmäratkaisut. [www-dokumentti]. Wapice Oy. [Viitattu 13.3.2017]. Saatavissa: https://www.wapice.com/fi/teknologiat/opc