Ilkka Jokela

Graafitietokannat esineiden internetissä

Tietotekniikan
pro gradu -tutkielma
1. marraskuuta 2023

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Kokkolan yliopistokeskus Chydenius

Tekijä: Ilkka Jokela
Yhteystiedot: ilkka.t.jokela@student.jyu.fi
Puhelinnumero: 044-553 33153
Ohjaaja: Lasse Harjumaa
Työn nimi: Graafitietokannat esineiden internetissä
Title in English: Graph databases and internet of things
Työ: Tietotekniikan pro gradu -tutkielma
Sivumäärä: 93
Tiivistelmä: Esineiden internetin (IoT) laitteet tuottavat valtavat määrät tietoa. Monesta eri laitteesta ja lähteestä kerätty tieto on usein jäsentämätöntä. Relaatiotietokannoista poiketen graafitietokantojen tietomalli soveltuu hyvin jäsentelemättömälle tiedolle. Tämän tutkielman tavoitteena on selvittää mitä tulee huomioida graafitietokantaa hyödyntävän IoT-järjestelmän väliohjelmisto- ja sovelluskerroksen suunnittelussa ja toteutuksessa. Lisäksi tutkielman tavoitteena on selvittää mitä graafitietokantavaihtoehtoja on saatavilla, soveltuuko graafitietokanta IoT-järjestelmän tietokantajärjestelmäksi ja miten dokumentoida tietokannan valinta ja muut arkkitehtuuripäätökset. Tutkielman tutkimus toteutetaan suunnittelututkimuksena. Suunnittelututkimuksessa suunnitellaan ja toteutetaan graafitietokantaa hyödyntävä IoT-järjestelmä, joka mittaa ja visualisoi olosuhteita.

Tutkimuksessa kävi ilmi, että graafitietokantaa hyödyntävän IoT-järjestelmän väliohjelmisto- ja sovelluskerroksen suunnittelussa ja toteutuksessa on otettava huomioon, että järjestelmän arkkitehtuuri ja tietomalli on helposti laajennettavissa ja muokattavissa. Tutkimus osoitti, että graafitietokannan tietomalli on joustava ja helposti laajennettavissa, minkä vuoksi graafitietokanta soveltuu hyvin IoT-järjestelmän tietokantajärjestelmäksi. Graafitietokannoista Neo4J on selkeästi suosituin, monipuolisin ja tuetuin. Tutkimuksessa selvisi, että arkkitehtuuripäätösten dokumentointiin ei ole yhtä kaikkiin projekteihin sopivaa mallia. Tämän vuoksi tutkimuksessa luotiin arkkitehtuuripäätösten dokumentaatiomalli, joka soveltuu IoT-järjestelmän väliohjelmisto- ja sovelluskerroksen suunnitteluun ja toteutukseen.
Avainsanat: Esineiden internet, graafitietokanta, Neo4J, NoSQL
Abstract: Devices of the internet of things (IoT) produce massive amounts of data. Data is often unstructured as it is gathered from multiple sources and by different devices. Unlike relational databases, the data model of graph databases is suitable for unstructured data. The objective of this study is to find out, what has to be considered when designing and implementing middleware and application layers for

IoT-system which uses graph database. In addition, the aim of this study is to determine what graph databases options are available. whether graph database is suitable for IoT-systems and how to document the choice of database and other architechtural decisions. The study is conducted as a design science research. IoT-system which uses graph database and measures and visualizes environment is designed and built in the research.

The research pointed out that possibility to easily expand and modify the architecture and data model of IoT-system has to be considered, when designing and implementing middleware and application layer for IoT-system using graph database. The research also showed that the graph database suits well for IoT-systems as it has flexible and easily extensible data model. Neo4J is by far the most popular, versatile and supported graph database option available. The research also demonstrated that there is no single documentation model for documenting architecture decision which suits all projects. For that reason, an architecture decision documentation model was created. The documentation model is suitable when designing and implementing middleware and application layers for IoT-systems.
Keywords: Internet of things, graph database, Neo4J, NoSQL

Sisällys

1 Johdanto

1.1 Tutkielman tausta

Nopeasti kasvavalle esineiden internetille (Internet of Things) on useita määritelmiä, mutta ei yhtään standardisoitua määritelmää [6]. Määritelmiä kuitenkin yhdistää se, että esineiden internet koostuu internetiin kytketyistä laitteista, jotka keräävät ja tuottavat valtavat määrät dataa [51]. Esineiden internetin keräämä ja tuottama tieto, valtavan määrän ohella, on usein aikasarjamuotoista ja monesta lähteestä kerättyä, minkä vuoksi IoT-järjestelmissä käytetään yhä useammin ei-relaatiotietokantoja, eli Not Only SQL -tietokantoja (NoSQL), perinteisten relaatiotietokantojen sijasta [11] [32] [49]. Toisin kuin relaatiotietokannoissa NoSQL-tietokannoissa ei ole linkkiä taulujen välillä [49]. NoSQL-tietokannassa tietomallia ei ole pakko ennal-ta määritellä ja tieto tallennetaan usein avain-arvo-muodossa [49]. Kerättyä tietoa käytetään päätöksenteossa ja sillä voidaan ohjata laitteita esimerkiksi älykodeissa. Älykotien lisäksi esineiden internetin sovelluksia kehitetään muun muassa terveydenhuollon, teollisuuden ja älykaupunkien tarpeisiin [11].

NoSQL-tietokantoihin lukeutuvalla graafitietokannalla on tehokasta kuvailla ja löytää entiteettien välisiä monimutkaisia riippuvuuksia ja suhteita, minkä vuoksi sitä on perinteisesti hyödynnetty sosiaalisten verkostojen kuvaamiseen [32]. Graafitietokannan tietomalli on helposti muokattavissa, mikä mahdollistaa entiteettien ja niiden välisten suhteiden lisäämisen, poistamisen ja muokkaamisen ilman järjestelmän pysäyttämistä tai migraatioiden tekemistä [32]. Joustavuus tekeekin graafitietokannan tietomallista houkuttelevan IoT-järjestelmien näkökulmasta, koska IoT-järjestelmissä laitteiden ja järjestelmän muiden osa-alueiden väliset suhteet muuttuvat, kun laitteita lisätään ja poistetaan, minkä lisäksi järjestelmän keräämä ja tuottama tieto voi muuttua [32].

Esineiden internet ja NoSQL-tietokannat ovat erittäin tutkittuja aiheita. Sen sijaan graafitietokannat IoT-järjestelmän tietokantajärjestelmänä on vähemmän tutkittu ja dokumentoitu aihe, vaikka graafitietokantojen suosio onkin kasvussa. Täs-tä johtuen tässä Pro Gradu -tutkielmassa suunnitellaan ja toteutetaan suunnittelutieteen yhden suunnittelusyklin aikana IoT-järjestelmä, joka hyödyntää graafitietokantaa tietokantajärjestelmänään. Tutkimuksella pyritään selvittämään, mitä tulee

huomioida graafitietokantaa tietokantajärjestelmänään käyttävän IoT-järjestelmän väliohjelmisto- ja sovelluskerroksen suunnittelussa ja toteutuksessa. Lisäksi selvitetään soveltuuko graafitietokanta esineiden internetin sovelluksen tietokantajärjestelmäksi ja mitä graafitietokantavaihtoehtoja on saatavilla.

Tietojärjestelmien lisääntynyt kompleksisuus on kasvattanut ohjelmistoarkkitehtuurin merkitystä [99]. IoT-järjestelmän tietokantajärjestelmän valinta on yksi IoT-järjestelmän arkkitehtuuripäätöksistä. Arkkitehtuuripäätöksiä ja päätöksiin johtaneita perusteluita ei ole perinteisesti nähty osana ohjelmistoarkkitehtuuria [96] [99]. Hyvin dokumentoidut arkkitehtuuripäätökset ovat kuitenkin olennainen osa ohjelmistoarkkitehtuuria, sillä ne auttavat sidosryhmiä ymmärtämään suunniteltua ja toteutettua ohjelmistoarkkitehtuuria [99] [96]. Arkkitehtuuripäätösten dokumentointiin ei ole yhtä yhteisesti hyväksyttyä mallia, joka sopisi jokaiseen ohjelmistoprojektiin [99]. Tässä tutkielmassa esitellään arkkitehtuuripäätösten dokumentaatiomalli, joka soveltuu IoT-järjestelmän ohjelmistoprojektiin.

1.2 Tutkimuksen tavoite ja tutkimuskysymys

IoT-järjestelmien keräämä ja tuottama tiedon määrä on valtaisa. Valtaisan tiedon määrän lisäksi IoT-järjestelmien data on usein monimuotoista ja järjestelemätön-tä, mikä asettaa haasteita tietokantajärjestelmälle. Tutkimuksen yhtenä tavoitteena onkin selvittää soveltuuko graafitietokanta IoT-järjestelmän tietokantajärjestelmäksi ja mitä tulee huomioida graafitietokantaa hyödyntävän IoT-järjestelmän väliohjelmisto- ja sovelluskerroksen suunnittelussa ja toteutuksessa. Graafitietokantajärjestelmiä on saatavilla useita erilaisia, joten tavoitteena on myös tarkastella saatavilla olevia vaihtoehtoja ja niiden ominaisuuksia. Tietokantajärjestelmän valinta on vain yksi lukuisista ohjelmistoprojektin arkkitehtuuripäätöksistä. Arkkitehtuuripäätösten dokumentointi on olennainen osa ohjelmistoarkkitehtuuria, mutta dokumentointiin ei ole yhtä mallia, joka sopisi kaikkiin ohjelmistoprojekteihin. Lisäksi ohjelmiston arkkitehtuuri elää projektin aikana tehtyjen päätösten myötä, joten on tärkeää, että tehdyistä muutoksista ja päätöksistä jää jälki perusteluineen dokumentoinnin myötä. Tutkimuksen tavoitteena onkin myös tuottaa arkkitehtuuripäätösten dokumentaatiomalli tukemaan IoT-järjestelmän suunnittelua ja toteutusta. Tutkielman tutkimuskysymykset ovat:

> Mitä tulee huomioida graafitietokantaa hyödyntävän IoT-järjestelmän väliohjelmisto- ja sovelluskerroksen suunnittelussa ja toteutuksessa?

- Mitä graafitietokantavaihtoehtoja on saatavilla?
- Soveltuuko graafitietokanta IoT-järjestelmän tietokantajärjestelmäksi?
- Miten dokumentoida tietokannan valinta ja muut arkkitehtuuripäätökset?

IoT-järjestelmien sovellusalueita ja käyttökohteita on lukuisia, joten tutkimus on rajattu koskemaan olosuhteita mittaavaa ja visualisoivaa IoT-järjestelmää, joka on tyypillinen IoT-järjestelmän käyttökohde. Tutkimus toteutetaan suunnittelemalla ja toteuttamalla edellä mainittu IoT-järjestelmä yhden suunnittelusyklin aikana. Tulokset auttavat tulevia projekteja IoT-järjestelmän tietokantajärjestelmän valinnassa ja IoT-järjestelmän suunnittelussa ja toteutuksessa.

1.3 Tutkimusmenetelmä

Tämän Pro Gradu -tutkielman tutkimus on suunnittelutieteen suunnittelututkimus. Suunnittelututkimus toteutettiin suunnittelemalla ja toteuttamalla graafitietokantaa hyödyntävä IoT-järjestelmä. Suunnittelutiede (design science) on artefaktien suunnittelua ja tutkimusta kontekstissa [103, sivut 3]. Konteksteja eli asiayhteyksiä voivat esimerkiksi olla ihmiset, organisaatiot, tavoitteet, normit, liiketoimintaprosessit tai arvot, kun taas artefaktit voivat esimerkiksi olla algoritmeja, metodeja, ohjelmistokomponentteja ja palveluita [42, sivu 1] [103, sivut 3]. Tutkittavat artefaktit on suunniteltu toimimaan vuorovaikutuksessa ongelman kontekstin kanssa, jotta ne voivat kehittää jotakin annetussa kontekstissa tai tuottaa uutta tietoa kontekstiin liittyen [42, sivu 3]. Suunnittelututkimuksella ratkaistavat ongelmat ovatkin kehitys- tai parannusongelmia, ja jokaisella ongelmalla on asiayhteys, jossa kehitystä pyritään tuottamaan [42, sivu 3] [103, sivu 4]. Wieringa [103, sivu 4] korostaa, että artefakti itsessään ei ratkaise mitään ongelmaa, vaan sen vuorovaikutus ongelman asiayhteyden kanssa auttaa ratkaisemaan ongelman. Tämä tarkoittaa sitä, että artefakti saattaa toimia toisin toisessa ongelman kontekstissa, jolloin se saattaa ratkoa, tai jopa aiheuttaa, eri ongelmia toisessa kontekstissa.

Suunnittelutieteen kaksi osaa, suunnittelu ja tutkimus, vastaavat eri kysymyksiin, suunnitteluongelmaan ja tietokysymyksiin, joita pyritään ratkaisemaan iteroimalla [103, sivu 4]. Suunnitteluongelmat odottavat muutosta oikeaan elämään ja vaativat analyysiä sidosryhmien todellisiin ja hypoteettisiin tavoitteisiin [42, sivu 7]

[103, sivut 4-5]. Suunnittelijasta riippuen ratkaisuja voi olla monenlaisia, joten ratkaisun hyödyllisyyttä tulee arvioida suhteessa sidosryhmien tavoitteisiin [103, sivut 4-5]. Wieringa [103, sivu 5] arvioikin, että suunnitteluongelmaan ei ole olemassa yh-tä ainoaa oikeaa ratkaisua. Tietokysymykset sen sijaan eivät odota muutosta, vaan kysyvät tietoa maailmasta sellaisena kuin se nyt on. Tietokysymyksen vastaus on ehdotus, jonka oletetaan olevan ainoa oikea vastaus esitettyyn kysymykseen. Tämä tarkoittaa sitä, että tutkijalla ei ole varmuutta siitä, onko vastaus täysin oikea vai ei. On kuitenkin selvää, että tietokysymyksen vastaus ei ole riippuvainen tutkijasta, ja vastausta tuleekin arvioida totuutta vasten, joka ei ole riippuvainen sidosryhmien tavoitteista [103, sivu 5]. Wieringa [103, sivu 5] toteaa suunnitteluongelman ja tietokysymyksen usein menevän tutkimusraporteissa sekaisin, koska suunnitteluongelma muotoillaan usein kysymyksenä. Ero on kuitenkin selkeä, kun huomaa, että suunnitteluongelmaan voi olla useita ratkaisuja, kun taas tietokysymykseen on vain yksi vastaus. Huomionarvoista on myös se, että tietokysymyksiin vastaaminen voi johtaa uusiin suunnitteluongelmiin [103, sivut 5-6].

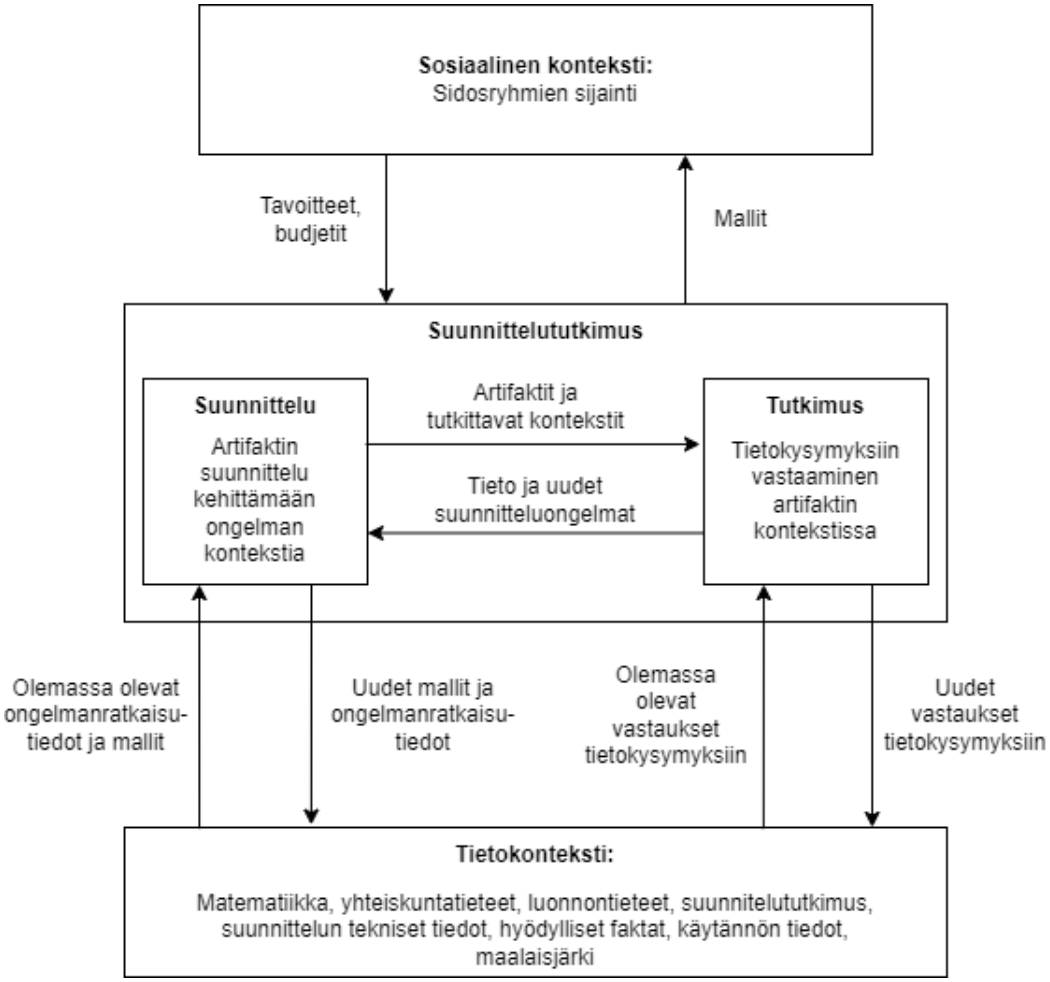
Suunnittelututkimuksessa tutkimuksen tavoite voi muodostua suunnittelu- ja tietotavoitteista, koska tutkimuksessa suunnitellaan ja tutkitaan iteroiden [103, sivu 13] [42, sivu 76]. Wieringan [103, sivu 13] mukaan jokaisen suunnittelututkimuksen tavoite on hauskuuteen ja uteliaisuuteen perustuvaa. Uusien artifaktien suunnittelu ja testaaminen on hauskaa, kun taas tietokysymyksiin vastaaminen on uteliaisuuden ohjaamaa. Tutkijalla voi olla myös yleishyödyllisiä tavoitteita, kuten yhteiskunnan parantaminen, jotka ovat usein samanlaisia kuin sidosryhmien tavoitteet [103, sivu 13] [42, sivu 7]. Suunnittelu- ja tietotavoitteiden lisäksi tutkijalla voi ol-la ennustetavoite, jolla pyritään ennustamaan mitä tapahtuu tulevaisuudessa. Ennustaminen vaatii kuitenkin pohjalle tietoa. Tietotavoitteet pyrkivät kuvailemaan ja selittämään jotakin ilmiötä [103, sivut 13-14]. Wieringan [103, sivu 14] mukaan tietotavoite voi esimerkiksi kuvailla ja selittää mitä tapahtuu, kun artefakti on vuorovaikutuksessa kontekstin kanssa. Tietokysymyksiin vastaaminen voi vaatia tutkimusinstrumenttien, esimerkiksi kyselytutkimuksen, suunnittelua. Tutkimusinstrumenteille voidaan asettaa omat suunnittelutavoitteet [103, sivu 14].

Suunnittelututkimuksella on usein ylemmän tason suunnittelutavoite, joka voi esimerkiksi olla artefaktin suorityskyvyn parannus tietyssä kontekstissa. Tällaista suunnittelutavoitetta kutsutaan artefaktin suunnittelutavoitteeksi tai tekniseksi tutkimustavoitteeksi. Artefaktin suunnittelun tavoitteena on ratkaista, lieventää tai parantaa jotakin ongelmaa projektin sosiaalisessa kontekstissa. Usein artefaktin suun-

nittelutavoite edistää myös yhden tai useamman ulkoisen sidosryhmän tavoitetta. Tutkimuksen tavoitteet määrittelevät ongelmat. Suunnitteluongelma on ongelma suunnitella artefakti, joka auttaa paremmin saavuttamaan jonkin tavoitteen. Suunnitteluongelmaa voidaan kutsua myös tekniseksi tutkimusongelmaksi. [103, sivut 14-17]

Roel J. Wieringan [103, sivu 7] kehys suunnittelututkimukselle on esitetty kuvassa 1.1. Suunnittelututkimuksen sosiaalinen konteksti sisältää sidosryhmät, jotka saattavat vaikuttaa tutkimusprojektiin tai saada vaikutteita siitä [42, sivu 5] [103, sivu 7]. Wieringan [103, sivut 7-8] mukaan sidosryhmään kuuluvat esimerkiksi suunniteltavan artefaktin käyttäjät, ohjeistajat ja ylläpitäjät [42, sivu 5]. Suunnitteluprojektin mahdolliset sponsorit, jotka lukeutuvat myös sidosryhmään, asettavat projektille budjetin ja tavoitteet. Projektin sponsori voi olla yritys, joka odottaa projektin tuottavan hyödyllisen mallin tai suunnitelman yritykselle. Tietokonteksti koostuu tieteen ja tekniikan luomista olemassa olevista teorioista, nykyisten suunnitelmien teknisistä tiedoista, saatavilla olevien tuotteiden hyödyllisistä faktoista, aiemmin suoritettujen suunnittelututkimusten tiedoista sekä maalaisjärjestä. Suunnittelututkimusprojekti hyödyntää tietokontekstia ja saattaa lisätä siihen uusia malleja eli suunnitelmia tai vastata tietokysymyksiin [103, sivut 6-8].

Tietoa, joka on saatavilla ennen suunnittelututkimusprojektia, kutsutaan aiemmaksi tiedoksi [103, sivu 8] [42, sivu 21]. Suunnittelututkimusprojektin tuottamaa tietoa kutsutaan puolestaan myöhemmäksi tiedoksi [103, sivu 8] [42, sivu 21]. Aiempi tieto voi koostua tieteellisistä julkaisuista, teknisestä kirjallisuudesta, ammattikirjallisuudesta sekä suullisesta tiedosta, joka on saatu esimerkiksi kollegoilta, konferensseista tai laboratorioista [103, sivu 8]. Mikäli suunnittelututkimusprojektin pitää tuottaa vastaus, joka perustuu tiukkaan tieteellisen tutkimuksen luomaan todistukseen jota ei ole saatavilla, tulee projektin tehdä tieteellinen tutkimus itse [103, sivu 8]. Wieringan [103, sivu 8] mukaan tämä kasvattaa merkittävästi tutkimuksen vaatimaa aikaa ja budjettia. Tämä johtaa käytännössä siihen, että päätös tutkimuksen tekemisestä tehdään yhdessä tukijoiden kanssa, koska he vastaavat tutkimuksen rahoituksesta. Johannesson ja Perjons [42, sivu 34] jakavat tiedon kuuteen eri tyyppiin riippuen tiedon tarkoituksesta. Määritelmällinen tieto on esimerkiksi konsepteja, terminologioita ja luokitteluja, kun taas kuvaileva tieto kuvailee olemassa olevaa tai mennyttä todellisuutta [42, sivu 34]. Selittävä tieto vastaa kysymyksiin miten ja miksi, samalla kun ennustava tieto tarjoaa ennustuksia [42, sivu 34]. Selittävä ja ennustava tieto tarjoaa sekä ennustuksia että selityksiä, kun taas prespriktiivinen tieto



Kuva 1.1: Kehys suunnitelututkimukseen [103, sivu 7]

tarjoaa malleja ja metodeja ongelman ratkaisuihin [42, sivu 34].

Tämän Pro Gradu -tutkielman sosiaalinen konteksti koostuu useista eri sidosryhmistä. Yksi sidosryhmä on IoT:sta ja graafitietokannoista kiinnostuneet tutkijat ja tutkimusyhteisö. Kokkolan yliopistokeskus ja Jyväskylän yliopisto asettavat omat tavoitteensa ja odotuksensa tietotekniikan Pro Gradu -tutkielmalle, mitkä on esitelty Jyväskylän yliopiston verkkosivuilla [43]. Tutkielman ohjaajalla ja arvioitsijoilla voi olla tavoitteita, jotka rakentuvat yliopiston asettamien tavoitteiden päälle. Sidosryhmiin lukeutuvat myös ne opiskelijat, jotka mahdollisesti hyödyntävät tutkielman suunnittelututkimusta omissa opinnoissaan. Lisäksi sidosryhmiin voidaan laskea he, jotka käyttävät Pro Gradu -tutkielman tuottamia malleja tai suunnitelmia omassa työssään. Tässä suunnittelututkimuksessa ei ole erillistä sponsoria, sillä tutkielmaa ei ole tehty toimeksiantona yritykselle tai yhteisölle. Tutkielman tietokonteksti perustuu tieteellisiin artikkeleihin, ammattikirjallisuuteen, olemassa olevien tuotteiden ja ratkaisujen määritelmiin ja ohjeiseen sekä aiemmin opittuun tietoon. Tässä Pro Gradu -tutkielmassa suunnitellaan ja toteutetaan olosuhteita mittaava ja visualisoiva IoT-järjestelmä, joka hyödyntää graafitietokantaa tietokantajärjestelmänään, yhden suunnittelusyklin puitteissa. Tutkielman tuottama uusi tieto kertoo, mitä tulee ottaa huomioon graafitietokantaa käyttävän IoT-järjestelmän suunnittelussa ja toteutuksessa. Myös uusi arkkitehtuuripäätösten dokumentaatiomalli on osa tutkielman tuottamaa uutta tietoa.

1.4 Tutkielman rakenne

Tutkielman rakenne koostuu johdannon lisäksi seitsemästä luvusta. Luvussa 2 esitellään esineiden internetin määritelmä ja arkkitehtuuri. Luvussa 3 keskitytään esineiden internetin erilaisiin tietokantajärjestelmiin ja tietokantajärjestelmän valintakriitereihin. Luvussa 4 syvennytään graafitietokantoihin. Arkkitehtuuripäätösten dokumentaatiomallit esitellään luvussa 5. Tutkimuksen toteutusta tarkastellaan luvussa 6. Tutkimuksen tulokset ja pohdinta käydään läpi luvussa 7. Tutkielman viimeisessä luvussa eli yhteenvedossa 8 esitetään kootusti tutkimuksen aikana tehdyt löydökset.

2 Esineiden internet

Tässä luvussa keskitytään esineiden internetin määritelmään ja arkkitehtuuriin. Alaluvussa 2.1 esitellään lyhyesti esineiden internetin erilaisia määritelmiä ja lukuisia sovellusalueita. Kuten esineiden internetin määritelmää, myöskään IoT-järjestelmien arkkitehtuuria ei ole standardisoitu [6]. Alaluvussa 2.2 onkin esitelty arkkitehtuureja, jotka ovat laajalle levinneitä ja tunnistettuja [6] [107]. Alaluku 2.3 keskittyy esineiden internetin havainnointikerrokseen. Alaluvuissa 2.4 ja 2.5 kuvaillaan verkkokerroksen ja sovelluskerroksen tehtäviä esineiden internetissä.

2.1 Esineiden internetin määritelmiä ja sovellusalueita

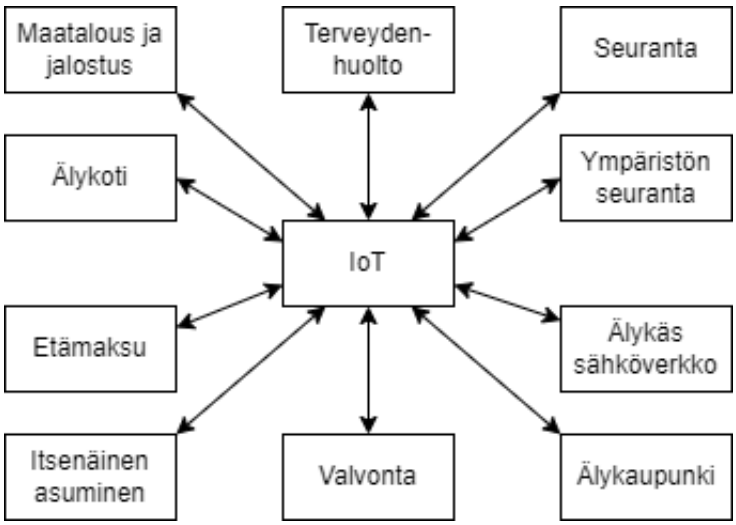
Nopeasti kasvava ja koko ajan teknisesti kypsempi esineiden internet (Internet of Things) voidaan määritellä monella tapaa, eikä sillä ole yhtä ainutta oikeaa tai standardisoitua määritelmää [28] [51] [6]. Madakam et al. [51] ovat esittäneet aikasarjan IoT-termin historiasta, josta voidaan huomata, että IoT terminä esiteltiin ensimmäisen kerran vuonna 1999 Kevin Ashtonin toimesta. Amghar et al. [11] määrittelevät IoT:n verkoksi liitettyjä laitteita, jotka luovat tietoa ja kommunikoivat keskenään verkkoprotokollilla ilman ihmisen osallistumista. D'Silva et al. [28] mukaan IoT on suuri määrä yhdistyneitä asioita, kuten laitteita, sijainteja ja ihmisiä. Madakam et al. [51] esittelevät useamman erilaisen määritelmän esineiden internetille. Niitä kaikkia kuitenkin yhdistää se, että esineiden internet muodostuu internetiin, langallisesti tai langattomasti, kytketyistä laitteista, jotka tuottavat valtavat määrät dataa.

Käyttipä määritelmänä yhtä edellä mainituista tai jotain muuta lukuisista tarjolla olevista, on IoT:n perusajatuksena kerätä ja havainnoida tietoa ympäristöstä tai vaikuttaa ympäristöön internetiin kytketyillä laitteilla. Kytketyt laitteet voivat olla sensoreita, jotka mittaavat ilman kosteutta ja lämpötilaa, tai ne voivat olla aktuaattoreita, jotka toimivat vastaanotetun tiedon perusteella [51]. Kerättyä tietoa voidaan käyttää päätöksenteossa ja sen perusteella voidaan ohjata laitteita esimerkiksi älykodeissa.

IoT-sovelluksia hyödynnetäänkin yhä useammalla tekniikan ja elämän eri osa-alueella. IoT-sovelluksia ja järjestelmiä kehitetään terveydenhuollon, teollisuuden,

kaupunkien ja kotien tarpeisiin [11]. Kuvassa 2.1 esitetään tärkeitä IoT-sovelluksia [48], jotka havainnollistavat IoT:ta käytettävän laajasti eri sovellusalueilla. Agarqal ja Misra [67] jakavat IoT-järjestelmien sovelluskohteet neljään kategoriaan: koti, yritys, julkiset palvelut ja matkapuhelinjärjestelmät. Kotikategoriaan kuuluville sovelluksille ominaista on se, että sensoreiden keräämää dataa käyttävät ainoastaan he, jotka omistavat tietoverkon. Yrityskategorialle tyypillinen sovelluskohde on rakennuksen tilan ja käytön monitorointi. Rakennuksen lämpötilaa ja ilmanvaihtoa voidaan ohjata sen mukaan, kuinka paljon siellä on henkilökuntaa paikalla. Vesi- ja viemäriverkon monitorointi on hyvä esimerkki sovelluskohteesta, joka asettuu julkisten palveluiden kategoriaan. Matkapuhelinjärjestelmien kategoriaan kuuluvat matkapuhelinten lisäksi muun muassa erilaiset navigointijärjestelmät [67].

Zhang et al. [105] jaottelevat IoT-sovellukset viiteen sovellusalueeseen: liikenne, terveydenhuolto, ympäristö, teollisuus ja sosiaalinen verkosto. Khanna et al. [44] puolestaan listaavat kahdeksan eri sovellusaluetta: teollisuuden prosessointi, maatalous ja jalostus, itsenäinen asuminen, älykäs liikkuvuus, älykäs sähköverkko, älykäs koti ja rakennus, julkisen turvallisuuden ja ympäristön monitorointi sekä lääketiede ja terveydenhuolto.



Kuva 2.1: IoT-sovellukset [48]

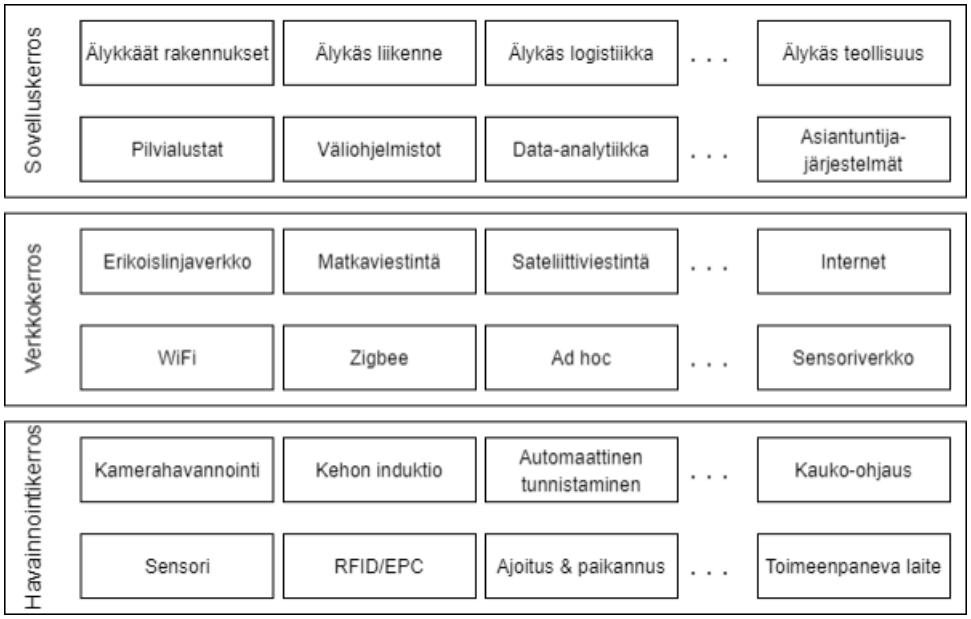
2.2 IoT-arkkitehtuuri

IoT-arkkitehtuurin tärkein tehtävä on varmistaa, että IoT-järjestelmän fyysiset laitteet ja virtuaaliset ohjelmistot ovat yhdistettyinä toisiinsa [33]. Arkkitehtuurin suunnittelussa tulee huomioida IoT-järjestelmien erityispiirteet kuten skaalautuvuus, laajennettavuus ja erilaisten laitteiden välinen dynaaminen toiminta [33]. Järjestelmään saatetaan lisätä tai poistaa laitteita käyttöönoton jälkeen. Laitteiden välinen yhteydenpito voi tapahtua milloin ja missä tahansa, joten siksi yhteydenpidon tulisi olla langatonta ja vain tarpeen mukaan tapahtuvaa [68]. Al-Qaseemi et al. [6] esittävät, että kolmi- ja viisikerroksiset arkkitehtuurit ovat yleisimpiä ja laajemmin käytettyjä lukuisista IoT-arkkitehtuureista.

Kolmikerrosarkkitehtuuria voidaan kutsua välttämättömäksi tai vähimmäiseksi IoT-arkkitehtuuriksi. Se koostuu nimensä mukaisesti kolmesta eri kerroksesta, jotka ovat havainnointi-, verkko- ja sovelluskerros [107] [6]. Kuvassa 2.2 esitetään Zhu et al. [107] mukaiset kolmikerrosarkkitehtuurin kerrokset listaten yleisiä tekniikoi-ta ja laitteita, joita eri kerrokset käyttävät. Havainnointikerrosta (Perception layer), joka on arkkitehtuurin alin kerros, kutsutaan myös fyysiseksi kerrokseksi. Sillä sijaitsee ympäristön havainnointia ja mittausta suorittavat laitteet kuten erilaiset sensorit, RFID tagit ja aktuaattorit [6] [107]. Alaluvussa 2.3 kuvaillaan tarkemmin havainnointikerroksen laitteita ja toteutuksia. Havainnointikerroksen laitteet lähettävät keräämänsä tiedot kolmikerrosarkkitehtuurin keskimmäiselle kerrokselle, verkkokerrokselle (Network layer), jonka vastuuna on yhdistää sovellus- ja havainnointikerros. Yhdistäminen toteutetaan joko langattoman tai langallisen tiedonsiirron avulla. Tiedonsiirtoon käytetään laitteista riippuen monenlaisia verkkoprotokollia ja tekniikoita, kuten Wifiä ja Zigbeetä [107].

Sovelluskerros (Application layer) on kolmikerrosarkkitehtuurin ylin kerros, ja sen vastuuna on käsitellä kerättyä dataa ja tarjota erilaisia palveluja käyttäjille ja toisille sovelluksille päätöksen teon avuksi [6] [107]. Esimerkkejä sovelluskerroksen tuottamista sovelluksista ovat muun muassa älykoteihin ja rakennuksiin liittyvät sovellukset. On huomioitava, että esineiden internetin älykkäät sovellukset tarvitsevat ja käyttävät sovelluskerroksella tietotekniikan muita tekniikoita kuten tietokantajärjestelmiä ja pilvipalveluita [107]. Sovelluskerroksesta voidaan lähettää tietoja myös takaisin havainnointikerrokselle verkkokerroksen kautta [6].

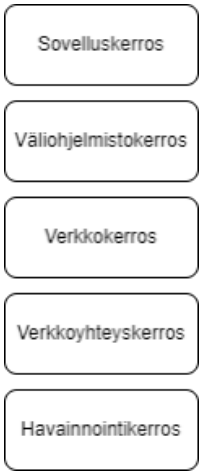
Zhu et al. [107] mukaan verkkokerros on esineiden internetin tärkein osa-alue, sillä se mahdollistaa tiedon havainnoinnin ja laitteiden älykkyyden. Tämän vuoksi on tärkeää, että verkkokerros jaetaan kahteen osaan, jotka ovat verkkoyhteysker-



Kuva 2.2: Esineiden internetin kolmikerrosarkkitehtuuri [107]

ros ja lähetyskerros. Jakamalla verkkokerros kahteen osaan, muodostuu IoT-arkkitehtuurista nelikerroksinen [107]. Verkkoyhteyskerros on vastuussa tiedonsiirrosta laitteiden ja verkon välillä asettuen havainnointikerroksen ja lähetyskerroksen väliin. Tiedonsiirto kerrosten ja laitteiden välillä hoidetaan usein jo mainituilla tekniikoilla kuten Wifillä ja Zigbeellä [107]. Al-Qaseemi et al. [6] kutsuvat verkkoyhteyskerrosta yhdyskäytäväkerrokseksi määrittelemässään esineiden internetin viisikerrosarkkitehtuurissa. Lähetyskerroksen tehtäväksi jää pidempien matkojen tiedonsiirto internetissä, joka toteutetaan hyödyntäen mobiilidataverkkoja tai esimerkiksi sateliittiverkkoja [107]. Verkkokerroksen käyttämiä lähetystekniikoita ja protokollia avataan yksityiskohtaisemmin alaluvussa 2.4.

IoT-arkkitehtuurin pilkkominen useampaan kerrokseen mahdollistaa kerrosten itsenäisen ja helpomman kehityksen ja ylläpidon [107]. Verkkokerroksen lisäksi pilkkomalla sovelluskerros kahteen osaan, muodostuu kolmikerrosarkkitehtuurista viisikerrosarkkitehtuuri. Sovelluskerros on luonnollista pilkkoa kahteen osaan siten, että irroitetaan havainnointikerroksen keräämän tiedon prosessointi, rikastuttaminen, säilöntä ja laskenta omaksi väliohjelmistokerroksekseen lähetyskerroksen ja sovelluskerroksen väliin [6] [107]. Väliohjelmistokerroksesta voidaan jakaa prosessoitua dataa useille eri sovelluksille [107]. Sovelluskerroksen tehtäväksi ylimpänä kerroksena viisikerrosarkkitehtuurissa jää rajapintana, visuaalisena käyttöliittymänä ja



Kuva 2.3: Esineiden internetin viisikerrosarkkitehtuuri [107] [68]

älykkäinä sovelluksina toimiminen käyttäjien ja IoT:n välillä [107]. Alaluvussa 2.5 on esitetty tarkemmin sovelluskerroksen ja väliohjelmistokerroksen toimintaa.

Kuvassa 2.3 on esitelty viisikerrosarkkitehtuurin eri kerrokset yhdellä tapaa. Kuvassa listattua väliohjelmistokerrosta kutsutaan monesti myös sovelluskerroksen tukikerrokseksi, sillä sen tehtävänä on käsitellä ja tallentaa tietoa ylimmän sovelluskerroksen tueksi [106]. Kuvasta 2.3 poiketen myös sovelluskerrosta saatetaan nimittää esityskerrokseksi. Esityskerroksen tehtävänä on visualisoida tukikerroksen käsittelemää dataa käyttäjille useilla eri tarkoituksia palvelevilla tavoilla ja sovelluksilla [106].

2.3 Havainnointikerros

Havainnointikerroksen tehtävänä on havainnoida ympäristöään ja kerätä reaaliaikaista dataa eri laitteiden avulla. Kerrosta kutsutaankin myös laitekerrokseksi [68]. Langattomat sensorit ja sensoriverkot ovat tyypillinen esimerkki havainnointikerroksen laitteista, jotka mittaavat ympäristöstä suureita kuten ilman kosteutta, lämpötilaa ja valoisuutta [104] [68]. Sensorit ovat vähävirtaisia laitteita, joilla on rajattu muisti, laskentateho ja akku tai paristo [48] [68]. Aktuaattoreilla voidaan esimerkiksi kontrolloida talon lämmitystä ja ilmanvaihtoa sensoreilla mitattujen arvojen kuten lämpötilan perusteella. Aktuaattorin ohjaus voi tapahtua sovelluskerroksesta käsin joko automaattisesti tai manuaalisesti [107].

Havainnointikerroksen laitteisiin lukeutuvat myös radiotaajuustunnistuslaitteet

(RFID), jotka keräävät tietonsa radiotaajuisen signaalin avulla [104]. RFID-laitteita käytetään esimerkiksi automaattiseen tunnistamiseen ja ne ovatkin ahkerassa käytössä erilaisissa rakennuksissa, jossa sisäänpääsyä halutaan kontrolloida perinteisten avainten sijaan tai ohella. RFID:tä hyödynnetään myös sähköautojen latauspisteissä tunnistamaan auton lataaja. RFID koostuu kahdesta osasta, tunnisteesta ja lukijasta. Tunniste sisältää tunnistetiedon, jonka lukija välittää internetiin [48]. Yhdistämällä RFID-järjestelmä langattomaan sensoriverkkoon pystytään paremmin seuraaman esineiden tai älykkäiden laitteiden tilaa ja sijaintia [67]. IoT-järjestelmissä havainnointikerrokseen kuuluvat lisäksi erilaiset älylaitteet, kuten matkapuhelimet [68].

Havainnointikerroksen havainnointiteknologia, laitteet ja ohjelmistot valitaan käyttötarkoituksen mukaan [107]. Valintaan vaikuttavat kerättävä tiedon tyyppi ja määrä, millä taajuudella tai kuinka usein tietoa kerätään, onko kerättävä tieto reaaliaikaista vai ei ja minkälaisessa ympäristössä tietoa kerätään [107]. Tietoliikenneprotokolla laitteiden välillä valitaan käyttötarkoituksen mukaan. Langattomat sensoriverkot hyödyntävät IEEE 802.15.4 standardia tai siihen perustuvaa Zigbeetä, jonka lisäksi käytössä voi olla Bluetooth [48]. Havainnointikerroksen laitteet voivat lähettää keräämänsä datan suoraan internetiin tai se voidaan lähettää tukiasemana toimivalle laitteelle, joka huolehtii tiedon välittämisestä eteenpäin verkkokerroksella yhdyskäytävää pitkin. Erillisen yhdyskäytävän käyttämisen etuina on, että havainnointilaitteiden ei tarvitse huolehtia muusta kuin tiedonkeruusta ja sen välittämisestä yhdyskäytävälle. Tämä lisää IoT-järjestelmän turvallisuutta, koska kaikki laitteet eivät ole suorassa yhteydessä internetiin [106]. Yhdyskäytävän tulee tukea laitteiden välisiä tiedonsiirtomenetelmiä, minkä lisäksi se on yhteydessä verkkokerroksessa langattoman tai langallisen internetyhteyden avulla.

Yang et al. [104] listaavat kuusi tärkeää teknologiaa ja ongelmaa, jotka IoT-järjestelmän havainnointikerroksen osalta tulisi ratkaista. Laitteiden etähallinta on tärkeää, jotta varmistetaan niiden luotettava tiedonkeruu. Järjestelmässä on usein paljon laitteita, joten niiden tulee olla edullisia, jotta kustannukset eivät kohoa liian korkeiksi. Laitteiden tulee olla vähävirtaisia tai niiden käyttämä virta on generoitava liikkeestä tai esimerkiksi auringosta. Niiden on oltava helposti käyttöön otettavia, jotta liiketoiminnassa voidaan hyödyntää ja hallita niiden keräämää dataa helpos-ti. Sensoreiden havainnointikykyä kehitetään mikrosysteemien, älykkäiden ja biologisten antureiden avulla. Sensoriverkkojen kehittäminen on tärkeää teollisuuden kehittämisessä. [104]

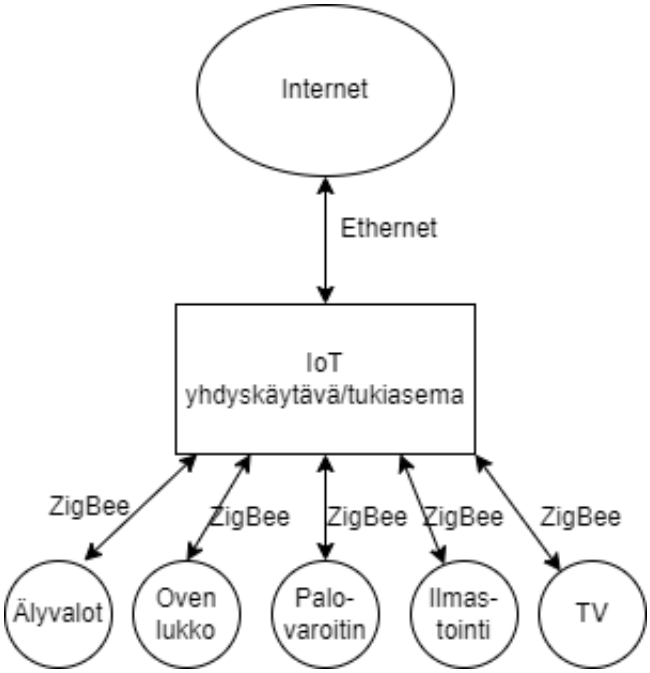
2.4 Verkkokerros

Verkkokerroksen tehtävänä on siirtää havainnointikerroksella kerätty data eteenpäin sovelluskerrokselle. Siirtämiseen voidaan käyttää joko langatonta tai langallista internet-yhteyttä. Viisikerrosarkkitehtuurissa verkkokerros voidaan jakaa verkkoyhteyskerrokseen ja verkkokerrokseen, kuten kuvassa 2.3 on esitetty. Yhteyskerrosta kutsutaan myös yhdyskäytäväkerrokseksi ja verkkokerrosta lähetyskerrokseksi. Yhteyskerros koostuu tukiasemasta ja yhdyskäytävästä ja sen pitää pystyä käsittelemään valtavia määriä havainnointikerroksen laitteiden keräämää dataa [106] [68]. Yhteyskerroksen lyhyen kantaman verkkoteknologiat, kuten ZigBee ja 6LoWPAN, perustuvat pääsääntöisesti IEEE 802.15.4 standardiin, mutta käytössä voi olla myös esimerkiksi WIFI tai Bluetooth [104] [68] [106]. Pidempien kantamien verkkoteknologioita tarjoavat Sigfox, NB-IoT ja LoRaWAN [37].

Tukiasemia ja yhdyskäytäviä voi olla useita riippuen havainnointikerroksen rakenteesta ja IoT-järjestelmästä. Havainnointikerroksen laitteiden kirjo voi olla moninainen, joten yhteyskerroksen tehtävänä on toimia kommunikaatiopisteenä kaikille kerroksen laitteille, vaikka ne kommunikoisivat monilla eri tavoilla [68] [106]. Tukiasemat keräävät havainnointikerroksen laitteiden havainnoiman datan ja lähettävät sen yhdyskäytävää pitkin lähetyskerrokselle. Lisäksi ne saattavat prosessoida dataa ennen tiedon lähettämistä eteenpäin. Tukiasema ja yhdyskäytävä saattavat olla samassa laitteessa.

Kuvassa 2.4 on yksinkertaistettu esimerkki yhdyskäytävän ja sensoreiden välisestä suhteesta, joka perustuu Zhong et al. [106] esittämään kuvaukseen hotelliketjun IoT-järjestelmän arkkitehtuurista. Kuvasta voimme havaita yhdyskäytävän, esimerkin tapauksessa myös tukiaseman, roolin linkkinä julkisen internetin ja havainnointikerroksen välillä, minkä lisäksi yhdyskäytävä voi olla yhteydessä myös muihin yhdyskäytäviin laajemmassa IoT-järjestelmässä. Yhdyskäytävä on yhteydessä internetiin joko langallisesti tai langattomasti [106]. Esimerkissä tieto yhdyskäytävän ja havainnointikerroksen kaikkien laitteiden välillä välitetään ZigBee-tietoliikenneprotokollalla, mutta yhtälailla laitteet voisivat toimia myös eri protokollilla, jos yhdyskäytävä olisi suunniteltu sitä tukemaan. Tiedonsiirto yhdyskäytävän ja laitteiden välillä tapahtuu ZigBee-protokollalla langattomasti. Tukiasema on myös vastuussa tiedon välittämisestä havainnointikerroksen laitteille, mikäli IoT-järjestelmässä on tarve välittää tietoa sovelluskerrokselta laitteille. Välitettävä tie-to voi olla ohjaustietoa, jolla aktuaattorit ohjaavat kiinteistön ilmastointia. Sovelluskerrokselta voidaan välittää laitteille myös ohjelmistopäivityksiä tukiaseman tai

yhdyskäytävän kautta [106].



Kuva 2.4: Yhdyskäytävä ja sensorit kommunikoivat keskenään ZigBeen avulla [106]

Verkkokerroksen toiminta voidaan jakaa kahteen osaan, lyhyen matkan ja pidemmän matkan kommunikaatioon [104]. Edellä kuvailtu yhdyskäytävän ja havainnointikerroksen laitteiden välinen toiminta on lyhyen kantaman kommunikaatiota, mikä usein hoidetaan IEEE 802.15.4 standardiin perustuvilla tietoliikenneprotokollilla. Havainnointikerrokselta kerätty data tulee välittää sovelluskerrokselle, mikä tapahtuu pidemmän matkan kommunikaatiolla [48]. Pidemmän matkan kommunikaatio tapahtuu joko langattomasti tai kaapeleita pitkin, IP-protokollaan perustuvalla internetillä, matkapuhelinverkoilla tai satelliittiverkoilla [104]. Kaapeleilla tapahtuva tiedonsiirto on usein luotettavampaa kuin langattomasti tapahtuva. Langaton tiedonsiirto kuitenkin mahdollistaa IoT-järjestelmien sijoittamisen sellaisiin paikkoihin, joihin ei ole vedetty tietoliikkennekaapelointia.

2.5 Sovelluskerros

Sovelluskerros on kolmikerrosarkkitehtuurin ylin kerros. Sen tärkeimpänä tehtävänä on analysoida ja prosessoida havainnointi- ja verkkokerrokselta tulevaa tietoa. Voidaan sanoa, että sovelluskerros on rajapinta käyttäjien, olivatpa ne sitten järjes-

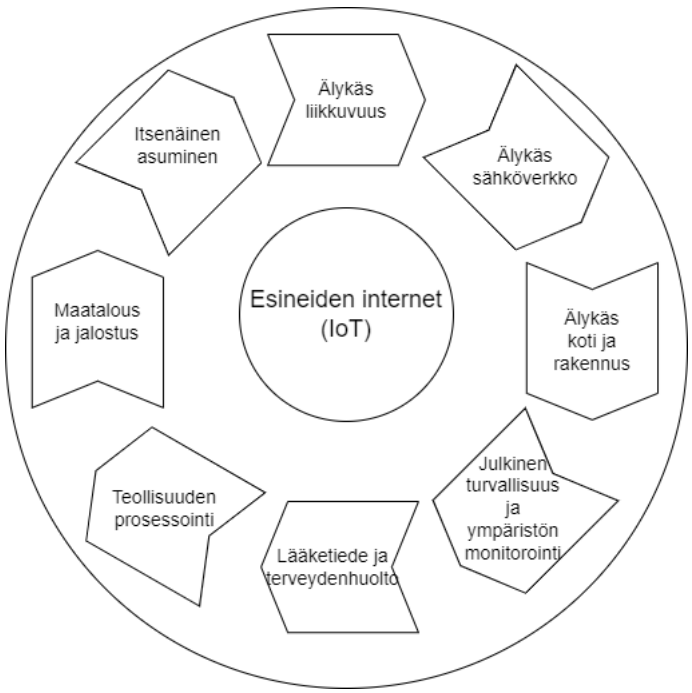
telmiä tai ihmisiä, ja IoT-järjestelmän välillä [106]. Kuvassa 2.2 on lueteltu erilai-sia palveluita ja sovelluksia, jotka kuuluvat sovellukerrokseen [107]. Zhong et al. [106] jakavat sovelluskerroksen kahteen osaan: sovelluksia tukevaan kerrokseen, väliohjelmistokerrokseen, ja sovelluksen esityskerrokseen, joka on vastuussa IoT-järjestelmän tuottaman tiedon visualisoinnista. Tiedon visualisoinnissa hyödynnetään alempien kerroksien ja sovelluskerroksen tuottamaa dataa [68]. Visualisointi voidaan toteuttaa esimerkiksi graafien, vuokaavioiden ja tuottotaulukoiden avulla. Visualisoinnin tehtävänä on auttaa päätöksentekoprosessissa, joka tapahtuu kerätyn tiedon perusteella [68].

Sovelluskerroksen tehtäviin lukeutuu IoT-laitteiden ja niiden luoman datan hallinta. Kushwah et al. [48] esittävät laitteiden hallinnan olevan erittäin tärkeää esineiden internetin kehitykselle. Laitteita pitää pystyä hallitsemaan etänä, jotta niitä voidaan palauttaa käyttöön, päivittää, sammuttaa ja laittaa päälle aina tarvittaessa. Koska laitteita voi olla monenlaisia IoT-järjestelmässä, tulee niiden monipuolisuus huomioida laitehallintasovellusta valittaessa tai suunnitellessa. [48]

Myös tiedonhallinta on IoT-järjestelmissä erittäin tärkeätä. Sensorit ja järjestelmän monet muut laitteet tuottavat valtavat määrät tietoa sovelluskerrokselle [105]. Tieto on rakenteeltaan moninaista johtuen siitä, että havainnoivia laitteita saattaa olla järjestelmässä usean tyyppisiä ja kerätyt suureet ovat erilaisia [68] [105]. Kerätyllä tiedolla on usein myös riippuvuus sijaintiin, jossa sensori on tiedon kerännyt [48] [105]. Sijainti voi tulla sensoreiden havainnoiman raakadatan mukana tai se voidaan siihen myöhemmin lisätä [105]. Tietoa pitää usein käsitellä ja analysoida, jotta sovelluskerroksen eri sovellukset pystyvät käyttämään tietoa, minkä lisäksi turha data pitää karsia pois [48] [25]. Tämä on väliohjelmistojen tehtävä, ja sen tärkeydes-tä kertoo se, että se on viisikerrosarkkitehtuurissa erotettu omaksi kerroksekseen, kuten alaluvussa 2.2 on esitetty. Väliohjelmistot ovat myös vastuussa tiedon analysoinnista ja tallentamisesta tietokantajärjestelmiin, jotka ovat osa sovelluskerrosta [48] [107] [106]. Tietokantajärjestelmät ovat olennainen osa IoT-järjestelmien tiedonhallintaa ja niihin on keskitytty luvussa 3.

Väliohjelmistokerroksen tehtävät, ohjelmistot ja siihen kohdistuvat vaatimukset riippuvat IoT-järjestelmän sovellusalueesta [105]. Sovellusalueita Khanna et al. [44] listaavat kahdeksan erilaista, jotka ovat nähtävissä kuvassa 2.5. Zhang et al. [105] luettelevat kuusi vaatimusta väliohjelmistoille: yhteentoimivuus, itsesopeutuvuus, reaaliaikaisuus, skaalautuvuus, luotettavuus sekä keveys. IoT-laitteiden laaja kirjo aiheuttaa sen, että väliohjelmistojen pitää taata yhteentoimivuus tukemalla erilaisia

tietorakenteita ja protokollia [105] [25]. Väliohjelmistojen tulisi sopeutua muuttuvaan tilanteeseen, kuten yllättävään kuormituksen kasvuun, IoT-järjestelmässä itsenäisesti, jotta se palvelee käyttäjän tarpeita [105] [25]. Aikariippuvaisten sovellusten on saatava reaaliaikaisesti lähetetty tieto ja sen perusteella tehdyt johtopäätökset [105] [25]. Väliohjelmistojen tulee olla skaalautuvia, koska IoT-järjestelmä, ja sen tuottama data, saattaa kasvaa alkuperäisen käyttöönoton jälkeen [105] [25]. Väliohjelmistojen pitää palvella käyttäjiä ja kehittäjiä luotettavasti myös silloin, kun järjestelmään väistämättä tulee virhetilanteita [105] [25]. Keveys on vaatimuksena hyvin riippuvainen siitä, että minkälainen IoT-järjestelmän käyttötarkoitus on. Mikäli väliohjelmiston tulee käsitellä massiivinen määrä tietoa, ei sen keveydellä ole suurta merkitystä, koska se on usein siirretty pilveen. Jos taas väliohjelmisto on sijoitettu laitteeseen, on sen syytä olla arkkitehtuuriltaan kevyt johtuen laitteiden rajoituksis-ta [105].



Kuva 2.5: IoT-sovellusalueet [44]

3 Esineiden internetin tietokantajärjestelmät

Tässä luvussa keskitytään esineiden internetin tietokantajärjestelmiin. IoT-järjestelmien käsittelemien suurten datamäärien takia tietokantajärjestelmät ovat olennainen osa IoT-järjestelmien tiedonhallintaa. IoT-arkkitehtuurissa tietokannat sijoittuvat kolmikerrosarkkitehtuurissa sovelluskerrokselle ja viisikerrosarkkitehtuurissa joko väliohjelmisto- tai sovelluskerrokselle. Alaluvussa 3.1 määritellään tietokannat perehtymällä lyhyesti niiden historiaan. IoT-järjestelmän tietokantajärjestelmän valintaan vaikuttavia tekijöitä listataan alaluvussa 3.2. Valintaan vaikuttavien tekijöiden perusteella muodostetaan IoT-järjestelmän tietokannan valintakriteerit alaluvussa 3.3. Tietokantatyypeistä relaatiotietokantoihin paneudutaan alaluvussa 3.4 ja NoSQL-tietokantoihin alaluvussa 3.5. Graafitietokantoihin keskitytään luvussa 4.

3.1 Tietokantojen historia

Tietokantojen historia ulottuu aikaan ennen tietokoneita, sillä ihmiset ovat säilyttäneet, indeksoineet ja hakeneet tietoja jo kauan ennen tietokoneiden keksimistä [16]. Tietokantajärjestelmän tehtävänä on luotettavasti säilyttää tietueita, muistis-sa tai kovalevyllä, ja tarjota tallennettuja tietoja käyttäjille [80, luku I]. Tietokanta ei ainoastaan sisällä siihen tallennettua tietoa, mutta myös tietoa tietokannan rakenteesta ja tallennetusta tiedosta [16]. Tietokanta terminä ajoittuu 1960-luvulle ja ensimmäiset tietokantajärjestelmät perustuivat navigointiin, jossa tallennetut tietueet sisälsivät viitteen toiseen tietueeseen, jonka perusteella tietue oli löydettävissä tietokannasta [16]. 1970-luvulla kehitetty ja Coddin [24] esittelemä relaatiotietokanta muutti tavan, jolla sovellukset hakivat tietueita tietokannasta [16] [18]. Aiemmin tiedonhaku perustui viitteisiin tai linkkeihin, jonka perusteella tietoa etsittiin kannasta, kun taas relaatiotietokannassa tieto haetaan itse tiedon perusteella [16] [24].

Chamberlin ja Boyce [19] kehittivät vuonna 1974 Structured English Query Languagen (SEQUEL), joka nyt tunnetaan laajemmin lyhenteellä SQL, jolla käyttäjä voi hakea, tallentaa ja muokata relaatiotietokannan tietuita [18]. SQL mahdollistaa englanninkielisten avainsanojen käytön matemaattisten merkintöjen sijasta, minkä vuoksi sen käyttö on helpompaa kuin aikaisempien tietokantakyselykielien käyttö

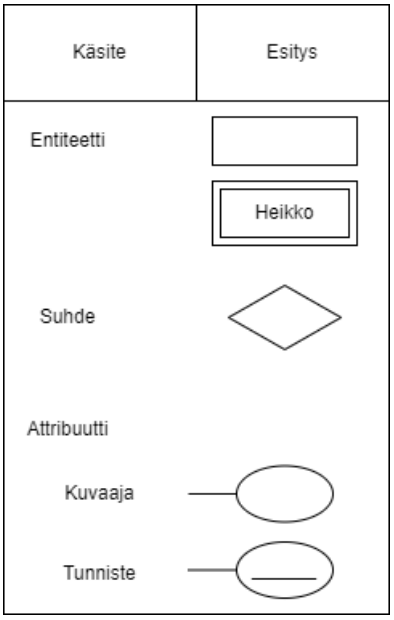
[19]. Chamberlin ja Boyce [19] ajattelivatkin, että SQL:n käyttö olisi niin helppoa, et-tä sitä käyttäisivät muutkin kuin tietotekniikkaan vihkiytyneet. Näin ei kuitenkaan käynyt, vaan SQL on suuresta suosiostaan huolimattaan edelleen tietotekniikan ammattilaisten työkalu [18].

Avainsanat SELECT, FROM ja WHERE luovat perustan SQL-kyselyille. SELECT-avainsanan yhteydessä käyttäjä määrittää sarakkeet, joista tietoa palautetaan. FROM-avainsanan jälkeen käyttäjän on määriteltävä tietokannan taulu, josta tietoa halutaan hakea. WHERE-avainsanan yhteydessä voidaan rajata palautettavan tiedon määrää halutuksi asettamalla ehto palautettavalle tiedolle. Ohessa on esitetty yksinkertainen SQL-kysely, jolla haetaan kaikki työntekijöiden nimet (NAME) ja palkat (SAL) työntekijätaulusta (EMP), joilla osasto (DEPT) on leluosasto (TOY) ja (AND) manageri (MGR) on nimeltään Anderson: [19]

```
SELECT   NAME, SAL
FROM     EMP
WHERE    DEPT = 'TOY'
AND      MGR = 'ANDERSON'
```

P. Chen [22] esitteli 1970-luvulla suositun Entity-Relationship (ER) mallin, jonka avulla voidaan abstrahoida ja visualisoida tallennetut tietoelementit entiteetteihin, attribuutteihin ja entiteettien välisiin suhteisiin [16] [93]. Entiteetit ovat esineitä, asioita tai tapahtumia eli kohteita, joista tieto halutaan tallentaa [22] [93]. Attribuutit ovat ominaisuuksia, jotka kuvailevat tai antavat tunnisteen entiteeteille [22] [93]. Suhteiden tarkoitus on kuvata entiteettien välisiä suhteita, jotka voivat olla kahden tai useamman entiteetin välisiä [22] [93]. Kuvassa 3.1 on esitetty edellä mainitut ER-mallin käsitteet, ja niiden visualisointiin ja suunnitteluun käytettävät kaaviot [93].

1980-luvulla kehitettiin oliopohjainen tietokanta, joka eroaa relaatiotietokannasta, ja on suunniteltu käytettäviksi olio-ohjelmointikielen kuten Javan ja C#:n kanssa [16]. Oliopohjaisen tietokannan tietomalli tukee suoraan olio-ohjelmointikielten olioita toisin kuin relaatiotietokannat [16] [45] [41]. Oliot kuvaavat oikean maailman entiteettejä eli asioita tai esineita [45]. Oliopohjaiset tietokannat eivät kuitenkaan saavuttaneet koskaan relaatiotietokantojen kaltaista suosiota [16]. Yhtenä syynä tähän oli object-relational mapping (ORM), joka kehitettiin 1990-luvulla, ja joka toi oliopohjaisten tietokantojen kaltaisen tuen olio-ohjelmoinnille myös relaatiotietokannoille [16]. Tudorica ja Bucur [95] luonnehtivat oliopohjaiset tietokannat kevyiksi NoSQL-tietokannoiksi, jotka jakavat joitain yhtäläisyyksiä niiden kanssa [41] [36].

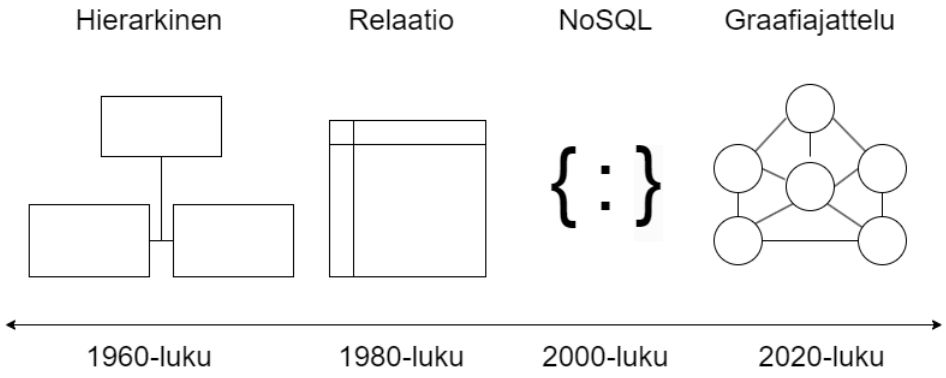


Kuva 3.1: ER-malli [93]

Hassan [36] mainitsee, että oliopohjaiset tietokannat, matalasta suosiostaan huolimatta, ovat edelleen käytössä tieteellisessä tutkimuksessa ja tietokoneavusteisessa suunnittelussa.

Carlo Strozz [16] käytti ensimmäisenä termiä NoSQL vuonna 1998 kuvailemaan avoimeen lähdekoodiin perustuvaa tietokantaansa, joka ei käyttänyt SQL-kyselykieltä. Termiä käytetäänkin nykyään kuvaamaan kaikkia tietokantoja, jotka eivät perustu relaatioon [94, luku 1]. NoSQL-tietokantojen suosio kasvoi 2000-luvulla, kun suuryhtiöt kuten Google ja Amazon kehittivät tietokantojaan massiivisen ja hajautetun tiedon käsittelyyn [94, luku 1] [16]. NoSQL-tietokannat jaetaan usein eri kategorioihin sen perusteella, miten tieto tallennetaan tietokantaan [16] [11]. NoSQL-tietokannat ovat kasvattaneet suosiota IoT-järjestelmien tietokantoina perinteisten relaatiotietokantojen rinnalla [82]. Goesnell ja Broecheler [34, luku 1] näkevät, että 2020-luku on graafiajattelun, ja NoSQL-tietokantoihin lukeutuvan graafitietokantojen, menestyksen aikaa, sillä graafien avulla on mahdollisuus nähdä tiedon välisiä suhteita ja ratkaista vaikeita ongelmia. Kuvassa 3.2 on esitetty Goesnellin ja Broechelerin [34, luku 1] näkemys tietokantateknologioiden historiasta ja tulevaisuudesta. Kuvassa 3.2 huomionarvoista on se, että Goesnell ja Broecheler [34, luku 1] sijoittavat teknologiat aikajanalla noin vuosikymmenen myöhemmäksi verrattuna niiden esittelyyn, koska teknologiat otettiin toden teolla käyttöön noin vuosikymmen nii-

den esittelyn jälkeen. Aikajanassa esitettyihin tietokantateknologioihin perehdytään NoSQL-tietokantojen osalta tarkemmin alaluvusssa 3.5 ja relaatiotietokantoihin alaluvussa 3.4. Graafitietokantoihin keskitytään luvussa 4.



Kuva 3.2: Aikajana tietokantojen teknologioista [34, luku 1]

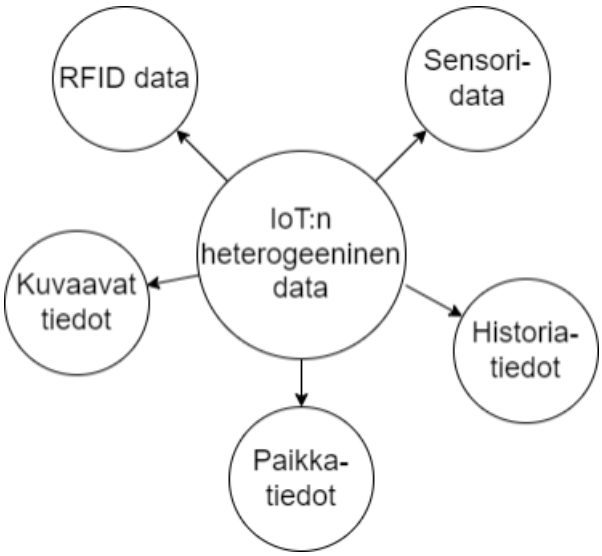
3.2 Tietokantajärjestelmän valintaan vaikuttavat tekijät

Azad et al. [15] mukaan tiedon tallentamisesta tulee ongelma, kun koko maailman esineet ovat yhdistettyjä IoT-järjestelmien kautta. Tietokantajärjestelmän tuleekin tarjota riittävästi laskentatehoa, levytilaa ja yhteyksiä [15]. Petrovin [80, luku I] mukaan tietokantajärjestelmän valinnalla voi olla pitkäkantoisia vaikutuksia. Ennen tietokantajärjestelmän valintaa tulee ymmärtää, että jokaisella järjestelmällä on heikkoutensa ja vahvuutensa [80, luku I]. Petrov [80, luku I] ehdottaakin, että ennen lopullista tietokantajärjestelmän valintaa, kannattaa simuloida tulevia kuormia erilaisia tietokantajärjestelmiä vasten. IoT-järjestelmissä havainnointikerroksen sensorit tuottavat valtavat määrät dataa [50] [49] [32]. Tiedon määrä on kuitenkin vain yksi tietokantajärjestelmän valintaan vaikuttava tekijä IoT-järjestelmässä. Azad et al. [15] kuvailevat IoT-järjestelmän tuottaman valtavan määrän tietoa olevan monimuotoista sekä tallennusmuodon että rakenteen osalta. Huomionarvioista on se, että tässä alaluvussa esitellyt IoT-järjestelmien tietokantajärjestelmän valintaan vaikuttavat tekijät ovat laajasti yhteneväisiä alaluvussa 2.5 mainittujen väliohjelmistokerroksen tehtävien ja vaatimusten kanssa. Tämä on luonnollista, koska tietokantajärjestelmät sijoittuvat viiden kerroksen IoT-arkkitehtuurissa väliohjelmistokerrok-

seen.

Amghar et al. [11] listaavat seitsemän IoT-järjestelmän tiedonhallintaan ja tietokantajärjestelmän valintaan vaikuttavaa tekijää. Tiedon monimuotoisuus on yksi valintaan vaikuttava tekijä, joka asettaa omat haasteensa tiedonhallintaan ja tallentamiseen [11] [27] [50]. IoT-järjestelmässä tietoa keräävät useat erilaiset laitteet, jotka tuottavat erilaista tietoa monessa muodossa, mahdollisesti eri tavalla mallinnetusti, joten valittavan tietokantajärjestelmän olisi hyvä tukea heterogeenisen ja mallintamattoman datan tallentamista [11] [49] [27] [50].

Diène et al. jaottelevat IoT-järjestelmän keräämät tiedot neljään kategoriaan: - RFID-data, sensoridata, kuvaavat tiedot, paikkatiedot ja historiatiedot, jotka ovat nähtävissä kuvassa 3.3. RFID-dataa muodostuu RFID-lukijan ja -tagin sisältävissä järjestelmissä, joita usein käytetään käyttäjän sisäänpääsyn antamiseen kiinteistöissä. Kuvaavat tiedot ovat metadataa havainnointikerroksella kerätystä tiedosta, jota esineiden internetin laitteet, prosessit ja järjestelmät tuottavat [27]. Metadata voi olla tietoa esimerkiksi laitteiden ominaisuuksista, käyttäjistä ja ympäristöstä [32]. Paikkatiedot ovat dataa mittauslaitteen sijainnista, joka voi olla GPS:ään tai muuhun teknologiaan perustuvaa [27].



Kuva 3.3: IoT:n heterogeeninen data [27]

Paikkatietojen merkitys korostuu järjestelmissä, joissa havainnointikerroksen laitteet ovat liikkuvia [11] [49]. Historiatiedoilla tarkoitetaan IoT-järjestelmän keräämää dataa, joista osa vanhenee ajan myötä, kun taas osa datasta pitää olla aktiivi-

sesti saatavissa [27]. Jotta voidaan tietää, mikä kerätystä tiedosta on vanhentunutta, tulee varsinaisen tiedon ohessa tallentaa aikaleima kuvaamaan tiedon keräyshetkeä [27]. Sensoridata on havainnointikerroksen keräämää tietoa, jota voidaan kerä-tä valtavasti ja reaaliaikaisesti, mutta jonka hakeminen ja käsittely voi olla hankalaa varsinkin reaaliaikaisesti [27].

Semanttinen yhteensopivuus on toinen tiedonhallintaan ja tietokantajärjestelmän valintaan vaikuttava tekijä [11]. Esineiden internetissä samaa tallennettua tietoa saattaa käyttää useampi sovellus, joten tietokannan on oltava ymmärrettävä monelle käyttäjälle [11] [49] [50]. Amghar et al. [11] väittävät, että yhteensopivuus voidaan saavuttaa yhdenmukaistamalla tietomuotoja, mallinnus- ja kyselykieliä. Li et al. [49] huomauttavat, että IoT-järjestelmässä tietoa voidaan varastoida kolmella eri tasolla: paikallisesti sensorilla, hajautetusti verkon noodeilla tai keskitetysti tietokannassa. Semanttinen yhteensopivuus saavutetaan parhaiten, kun IoT-järjestelmän luoma tieto on tallennettu keskitetysti [49]. Keskitetysti tiedon tallentamisella ei kuitenkaan tarkoiteta sitä, että tieto ei voisi olla hajautettuna tai kopioituna useampaan eri sijaintiin saatavuuden ja vikasietoisuuden parantamiseksi [49].

Kolmas tiedonhallintaan ja tietokantajärjestelmän valintaan vaikuttava tekijä on skaalautuvuus [11]. IoT tuottaa massiiviset määrät dataa, jota pitää pystyä käsittelemään nopeasti [11] [32] [49]. Tietokantojen tulee olla skaalautuvia, jotta kasvava tiedonmäärä saadaan tallennettua tehokkaasti [11] [49] [27]. Neljäs tietokantajärjestelmän valintaan vaikuttava tekijä on tiedon reaaliaikainen käsittely, koska osa IoT-sovelluksista käyttää reaaliaikaista tietoa [11] [27]. Tämä aiheuttaa sen, että tieto pitää tallentaa ja hakea tietokannasta nopeasti, minkä lisäksi tiedon käsittely pitää olla nopeaa [11] [49] [27]. Amghar et al. [11] mainitsevat turvallisuuden viidenneksi valintaan vaikuttavaksi tekijäksi. Koska IoT-laitteet tuottavat myös yksityistä tietoa, tulee sen käsittelyssä tarvittaessa huomioida tiedon salaus, ja autentikaation avulla se, kuka pääsee käsiksi tietoon [11] [27] [50].

Kuudes valintaan vaikuttava tekijä on paikkatietojen käsittely [11]. IoT-järjestelmien laitteista iso osa on liikkuvia, jonka myötä paikkatiedot ja niiden käsittely korostuu, koska laitteiden sijainnilla on merkitystä niiden keräämään dataan [11] [49] [27]. Tietojen yhdistäminen on seitsemäs IoT-järjestelmän tiedonhallintaan vaikuttava tekijä [11]. Tietojen yhdistämisellä tarkoitetaan tietojen yhdistämistä useasta eri lähteestä niin, että tarpeeton tieto karsitaan pois, jotta tallennetaan ja käytetään vain hyödyllistä dataa [11] [50]. Tietojen yhdistäminen on tärkeää myös semanttisen yhteensopivuuden kannalta, sillä käsittelemätön data on semanttisesti heikkoa [50].

Edellä liistattujen IoT-järjestelmän tietokantajärjestelmän valintaan vaikuttavien tekijöiden lisäksi on tekijöitä, jotka ovat yleisiä tietokantajärjestelmän valintaan vaikuttavia syitä. Hassan [36] korostaa kehitettävän sovelluksen käyttötarkoitusta tärkeänä valintaan vaikuttavana syynä, koska tietokannan on tuettava sovelluksen tarpeita. Vicknair et al. [102] vertailevat relaatiotietokantaa ja graafitietokantaa tietojen alkuperän näkökulmasta, ja nostavat yhdeksi subjektiiviseksi arviointikriteerik-si ohjelmoitavuuden helppouden. Vicknair et al. [102] esittävät myös, että teknolo-gian kypsyys ja tuki vaikuttavat siihen, kannattaako tietokantajärjestelmää valita. Ordonez et al. [79] nostavat myös vertailussaaan esille ohjelmoitavuuden helppouden ja teknologian kypsyyden käytännön valintakriteereinä tietokantajärjestelmää valittaessa.

IoT-järjestelmien tuottama tieto on usein vapaamuotoisempaa ja järjestelemättömämpää kuin relaatiotietokantoihin soveltuva data [32] [49] [15]. Perinteisten relaatiotietokantojen sijaan käytetäänkin yhä useammin paremmin skaalautuvia ja joustavampia NoSQL-tietokantoja IoT-sovelluksissa [11] [32] [15]. On kuitenkin tilanteita, joissa IoT-järjestelmän tietokannaksi soveltuu paremmin relaatiotietokanta, koska se suoriutuu nopeammin tietojen yhdistelystä kuin NoSQL-tietokanta [82] [14]. Azad et al. [15] toteavat tutkimuksessaan, että IoT-järjestelmien tietokannat jakautuvat kolmeen kategoriaan, jotka ovat relaatiotietokannat, NoSQL-tietokannat ja graafitietokannat. Tässä luvussa on esitelty näistä IoT-järjestelmissä käytetyistä tietokantajärjestelmistä relaatiotietokannat 3.4 ja NoSQL-tietokannat 3.5. Graafitietokanta on esitelty omassa luvussaan 4.

3.3 IoT-järjestelmän tietokannan valintakriteerit

Edellisessä alaluvussa 3.2 listattiin IoT-järjestelmän tietokannan valintaan vaikuttavia tekijöitä. Tässä alaluvussa muodostetaan niiden perusteella IoT-järjestelmän tietokannan valintakriteerit. Esiteltävien valintakriteerien keskenäinen painoarvo määräytyy tietokantaa valittaessa sen mukaan, että mikä on sitä käyttävän tai käyttävien sovellusten käyttötarkoitus, koska sovellusten käyttötarkoitus tulee aina huomioida tietokantaa valittaessa [36] [82]. Tietokannan lopullista valintaa tehtäessä ei kannata pelkästään luottaa ennalta määritettyihin kriteereihin, vaan tietokannan tulevaa kuormaa kannattaa simuloida ja testata etukäteen [80, luku I].

Amghar et al. [11] listaavat viisi valintakriteeriä IoT-järjestelmän tietokannalle vertaillessaan NoSQL-tietokantojen sopivuutta IoT-järjestelmän tietokannaksi. Va-

lintakriteerit ovat skaalautuvuus, tietojen reaaliaikainen käsittely, turvallisuus, tietojen yhdistely ja paikkatietojen hallinta [11]. Valintakriteerit perustuvat seitsemään IoT-järjestelmän tietokannan valintaan vaikuttaviin tekijöihin, jotka esiteltiin alaluvussa 3.2. Huomionarvoista on se, että Amghar et al. [11] jättävät tietojen heterogeenisyyden ja semanttisen yhteensopivuuden pois valintakriteereistä, vaikka listaavatkin ne tietokannan valintaan vaikuttaviksi tekijöiksi, koska kaikki NoSQL-tietokannat tukevat tietojen heterogeenisyyttä ja semanttinen yhteensopivuus voidaan saavuttaa ontologian avulla.

Kuvassa 3.4 on esitetty seitsemän IoT-järjestelmän tietokannan valintakriteeriä. Amghar et al. [11] kriteereistä poiketen mukana ovat valintakriteerit tietojen heterogeenisyys ja helppokäyttöisyys. Tietojen heterogeenisyys on valittu valintakriteereiksi, jotta IoT-järjestelmän tietokantavalinta voidaan tehdä laajemmasta joukos-ta tietokantajärjestelmiä relaatiotietokannat mukaan lukien. Tietojen heterogeenisyydellä tarkoitetaan tietokannan tukea mallinnetulle, mallintamattomalle ja monimuotoiselle datalle, jota erilaiset IoT-järjestelmän laitteet tuottavat [11] [49]. Semanttinen yhteensopivuus on sen sijaan jätetty pois valintakriteereistä, koska se voidaan saavuttaa ontologia avulla riippumatta siitä, että millä teknologialla tietokantajärjestelmä on toteutettu [11].

Kuva 3.4: IoT-järjestelmän tietokannan valintakriteerit [11] [82] [102]

Vicknair et al. [102] mainitsevat ohjelmoinnin helppouden, tietokantajärjestel-

män teknologisen kypsyyden ja saatavilla olevan tuen subjektiivisiksi tietokannan arviointikriteereiksi. Kuvan 3.4 valintakriteereissä ne on yhdistetty yhdeksi valintakriteeriksi helppokäyttöisyys, koska saatavilla oleva tuki ja dokumentaatio helpottavat tietokannan käyttöä ja ohjelmoitavuutta [26] [102]. Tietokannan helppokäyttöisyys on subjektiivinen valintakriteeri, johon vaikuttavat tietokannan tulevien käyttäjien aikaisempi kokemus ja saatavilla oleva tietokantajärjestelmän valmistajan ja yhteisön tuki [26] [102]. Tietokantajärjestelmän teknologinen kypsyys vaikuttaa myös sen turvallisuuteen, sillä kypsemmässä teknologiassa on ehditty huomata ja korjata mahdolliset tietoturvaongelmat [82] [15]. Tietokannan turvallisuutta, mikä on yksi tietokannan valintakriteereistä, arvioidessa tulee huomioida se, että miten autentikaatio, käyttäjien hallinta, tiedonsiirto, tietojen luottamuksellisuus, salaus ja eheys on varmistettu ja turvattu [82] [11] [102].

Valitakriteerillä skaalautuvuus tarkoitetaan sitä, että voiko tietokantaa skaalata joko vertikaalisesti tai horisontaalisesti, kun IoT-järjestelmän tuottaman tiedon tai käyttäjien määrä kasvaa [82] [11]. Vertikaalisessa skaalauksessa nostetaan tietokantainstanssin resursseja kuten muistin tai prosessoreiden määrää [82] [15]. Horisontaalisessa skaalauksessa tietokantainstanssien lukumäärää kasvatetaan [82] [15]. Uusille IoT-järjestelmille skaalautuvuus voi olla hyödyllistä, koska se mahdollistaa resurssien lisäämistä tarvittaessa myöhemmin, mutta alussa ei ole pakko investoida suuria määriä [15]. Monet IoT-järjestelmät ja niiden sovellukset käyttävät reaaliaikaista dataa [27] [11], joten tuki reaaliaikaisuudelle ja kyky prosessoida tietoa nopeasti on luonnollinen valintakriteeri IoT-järjestelmän tietokannalle.

Kuudes valintakriteeri on paikkatietojen käsittely. Paikkatietojen käsittelyllä tarkoitetaan sitä, että miten tietokanta käsittelee ja hallinnoi paikkatietoja [11]. Paikkatiedot ovat olennainen osa IoT-järjestelmien tuottamaa dataa, koska suuri osa IoT-laitteista ovat liikkuvia ja laitteiden sijainnilla on merkitystä niiden tuottamaan tietoon [11] [49] [27]. Seitsemäs valintakriteeri on tietojen yhdisteltävyys, jolla tarkoitetaan kykyä yhdistellä tietoja useasta eri lähteestä samalla karsien tarpeettoman tiedon määrää, joka on tärkeää johtuen IoT-laitteiden tuottaman massiivisen tiedon määrän vuoksi [11] [50] [15]. Azad et al. [15] väittävätkin, että tietojen yhdistelyl-lä voidaan mahdollisesti vastata valtavan tiedonmäärän luomaan haasteeseen IoT-järjestelmissä. Alla on seitsemän IoT-järjestelmän tietokannan valintakriteeriä kootusti listana:

- tietojen heterogeenisyys

- helppokäyttöisyys
- turvallisuus
- skaalautuvuus
- reaaliaikaisuus
- paikkatietojen käsittely
- tietojen yhdisteltävyys.

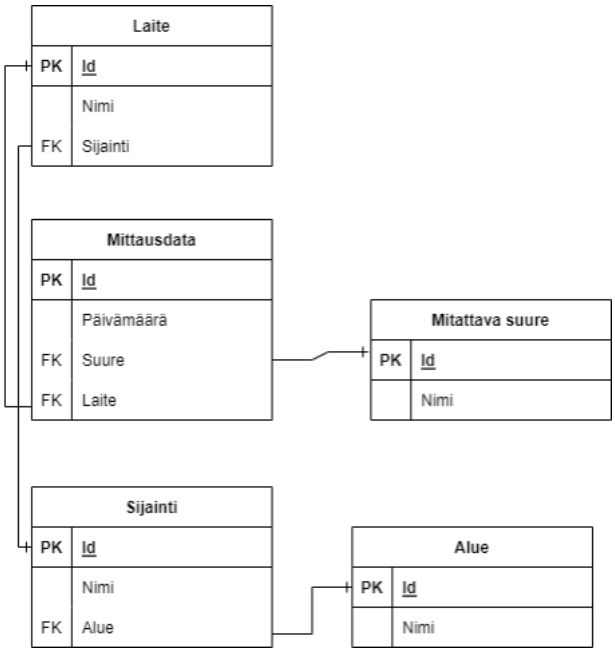
3.4 Relaatiotietokannat

Kuten alaluvussa 3.1 mainittiin, esitteli Codd [24] relaatiomalliin perustuvan tietojen tallentamisen jo 1970-luvulla. Relaatiotietokannat ovatkin yleisin ja tunnetuin tapa tallentaa tietoa tietokantaan johtuen niiden yksinkertaisuudesta [35] [15]. MySQL, PostgreSQL, Oracle ja SQL Server ovat esimerkkejä tämän hetken suosituista relaatiotietokannoista [82] [36]. Chamberlinin ja Boycen [19] esittelemä SQL on yleisin ja suosituin kyselykieli, jolla haetaan ja muokataan relaatiotietokannan tietueita [41] [35]. Se on niin suosittu ja tunnettu, että sitä käytetään jopa synonyyminä relaatiotietokannalle [85]. Ohjelmistokehittäjille on saatavissa laajasti opetusmateriaalia SQL-kielen käytön ja opiskelun tueksi [15]. Voidaankin todeta, että relaatiotietokannat täyttävät selkeästi IoT-järjestelmän tietokannan valintakriteereistä helppokäyttöisyyden. Relaatiotietokannat ovat teknologiana erittäin kypsiä, minkä johdosta ne täyttävät myös valintakriteerin turvallisuus [82].

Relaatiotietokannassa tieto tallennetaan tietokantatauluihin riveille ja sarakkeisiin taulukon muotoon [41] [24] [82]. Jokainen tietokantataulun rivi sisältää sarakkaiden määrittelemää uniikkia tietoa ja se usein tunnistetaan pääavaimella [41]. Taulut sisältävät varsinaisen tiedon lisäksi tiedon niiden välisestä suhteesta eli viiteavaimen, jolla viitataan toisen taulun pääavaimeen [24] [82]. Kuvassa 3.5 on esitetty Fosicin ja Solicin [32] luoma yksinkertaistettu esimerkki IoT-järjestelmän tietomallista relaatiotietokannan tietokantatauluina, joka kuvaa hyvin edellä esiteltyä pääavainten ja viiteavainten käyttöä kuvaten tietokantatauluihin tallennettujen tietojen välisiä suhteita.

Kuvan 3.5 laitetauluun on tallennettu laitteen yksilöivä pääavain ja nimi, jonka lisäksi viiteavain sijainti viittaa Sijaintitauluun, joka kertoo laitteen sijainnin. Sijaintitaulussa on yksilöivän pääavaimen ja nimen lisäksi viiteavain Aluetauluun, joka

kertoo alueen, jolla laite sijaitsee. Mittausdatataulussa on yksilöivän pääavaimen lisäksi päivämäärä, joka kertoo koska mittausdata on mitattu. Laitteeseen viittaava viiteavain kertoo mittauksen suorittaneen laitteen. Toinen viiteavain on Suuretauluun lähtevä, joka kertoo mitattavan suureen. [32]



Kuva 3.5: Esimerkki IoT-järjestelmän tietokantatauluista [32]

Mikäli kyseisestä tietomallista haluttaisiin hakea kaikki yhden laitteen (dev1) tekemät kosteusmittaukset (humidity) tietyllä alueella (east), näyttäisi kysely Fosicin ja Solicin [32] mukaan seuraavanlaiselta, kun taulujen nimet ovat device (laite), location (sijainti), region (alue), measurement_type (mitattava suure) ja measurement (mittausdata):

```
SELECT metrics.id, measurements.date, device.name,
dimensions_type.type, region.name

FROM device, location, region, measurements,
measurements_type

WHERE device.location = location.id
AND location.region = region.id
AND measurement.device = device.id
```

```
AND measurement.type = measurements_type.id
AND region.name = 'east'
AND measurements_type.type ='humidity'
AND device .name = 'dev1'
```

Fosic ja Solic [32] esittävät oheisen erittäin pienen taulukon 3.1 suhteellisen pitkän tietokantakyselyn tuloksena. Kyselyssä haettiin tietoa viidestä eri tietokantataulusta pääavaimia ja viiteavaimia hyödyntäen.

Taulukko 3.1: SQL-tietokantakyselyn tulokset [32]

id	**päivämäärä**	**laitteen nimi**	**mitattava suure**	**alueen nimi**
1	1.9.2018	dev1	kosteus	itä
2	2.9.2018	dev1	kosteus	itä

Edellä esitellystä IoT-järjestelmän tietomallista on esitetty alaluvussa 4.1 ja kuvassa 4.2 graafitietokantaversio. Alaluvussa 4.3 esitellään miten graafitietokannasta voidaan hakea Cypher-kyselykielellä samat tulokset.

Relaatiotietokannat tukevat ACID-periaatetta, joka koostuu sanoista Atomicity (suom. atomisuus), Consistency (suom. eheys), Isolation (suom. eristyneisyys) ja Durability (suom. pysyvyys), ja jolla varmistetaan tietokannan luotettavuus [41] [35]. Atomisuudella tarkoitetaan sitä, että jos tietokantatransaktiota ei voida kokonaisuudessaan suorittaa, niin sitä ei suoriteta ollenkaan [41] [82]. Eheys varmistaa tietokannan olevan vakaa jokaisen transaktion jälkeen [41] [82]. Eristeneisyys pitää huolen siitä, että samanaikaisesti suoritettavat transaktiot eivät vaikuta toisiinsa [41] [82]. Pysyvyydellä pidetään huolta siitä, että kun transaktio, esimerkiksi uuden tiedon tallennus tietokantaan, on suoritettu, niin tieto myös varmasti on tietokannassa, vaikka järjestelmässä tapahtuisi virheitä tai tietokanta menettäisi virran transaktion jälkeen [41] [82]. ACID-periaatteita noudattamalla varmistetaan tiedon oikeellisuus, mutta sillä on negatiivinen vaikutus tietokannan kirjoitusnopeuteeen, joka on yksi syy miksi relaatiotietokantojen kirjoitusnopeus on hitaampi kuin NoSQL-tietokantojen [41].

Hitaammalla kirjoitusnopeudella on vaikutusta IoT-järjestelmän tietokannan valintakriteereistä reaaliaikaisuuteen, sillä mitä nopeammin tietoa voidaan kirjoittaa tietokantaan, sitä nopeammin se on myös sieltä luettavissa. Relaatiotietokantoja voidaan skaalata vertikaalisesti [82] [15], mikä rajoittaa niiden skaalautuvuutta ja vai-

kuttaa sitä myöten valintakriteereistä skaalautuvuuteen. Skaalautuvuus on tärkeää, kun IoT-järjestelmän tuottaman tiedon määrä kasvaa koko ajan [15]. Asiminidis et al. [14] suorittama suorituskykyvertailu osoittaa, että relaatiotietokannat suoriutuvat hyvin IoT-järjestelmän tuottaman tietojen yhdistelystä, joka on yksi tietokannan valintakriteereistä.

3.5 NoSQL-tietokannat

NoSQL-termi esiteltiin ensimmäisen kerran 1990-luvun lopulla Carlo Strozzin toimesta. NoSQL-tietokantojen tärkein tehtävä on säilyttää ja hallita strukturoimatonta tietoa, joka on usen avain-arvo muodossa [49] [82]. NoSQL-tietokannoissa ei ole linkkiä taulujen välillä kuten perinteisissä relaatiotietokannoissa [32]. Ei-relaatiotietokannat soveltuvat hyvin käyttötarkoituksiin, joissa tallennetaan suuret määrät dataa georeplikoituna, ja joissa kyselyiden tulee olla nopeita, koska NoSQL-tietokannat ovat horisontaalisesti skaalautuvia [32] [11] [35]. NoSQL-tietokannat voidaan jakaa neljään eri kategoriaan: avain-arvo-tietokannat, sarakkeiset tietokannat, dokumenttitietokannat ja graafitietokannat [11] [35].

Poiketen perinteisistä relaatiotietokannoista NoSQL-tietokannat ovat skeemattomia, eikä tietoa tarvitse mallintaa ennen tallentamista [49] [82]. Tämä mahdollistaa heterogeenisen tiedon tallentamisen joustavasti [49]. Relaatiotietokannoista poiketen NoSQL-tietokannat eivät pääsääntöisesti noudata ACID-periaatetta pois lukien osa graafitietokannoista [35] [41] [102] [36]. Tudorica ja Bucur [95] kuitenkin esittävät, että väliohjelmistojen avulla on mahdollista toteuttaa ACID-periaatteet myös NoSQL-tietokannoille [41]. ACID-periaatteiden sijaan NoSQL-tietokannat noudattavat basic availability (suom. periaatteessa käytettävissä), soft state (suom. ei aina oikeellinen) ja eventual consistency (suom. lopulta eheä) (BASE) periaatteita korkeamman suorituskyvyn ja saatavuuden vuoksi [95] [82] [35]. BASE-periaatteilla tarkoitetaan käytännössä sitä, että luettavissa olevan tiedon oikeellisuutta ei taata joka hetki, mutta se on lopulta oikeellista [82] [35].

Rautmare ja Bhalerao [82] esittävät, että NoSQL-tietokantojen suosio IoT-järjestelmissä perustuu niiden skaalautuvuuteen, helppoon saatavuuteen ja hajautettuun arkkitehtuuriin. Azad et al. [15] listaavat korkean turvallisuuden, saatavuuden, tehokkuuden ja yksinkertaisten kyselyjen matalan vasteajan, hinnan ja virrankäytön NoSQL-tietokantojen vahvuuksiksi IoT-järjestelmän tietokantana. Heikkouksiksi Azad et al. [15] listaavat korkean kompleksiksuuden, turvallisuuden, kyselyjen

korkean vasteajan sekä heikon skaalautuvuuden osalla NoSQL-tietokannoista. Osittain ristiriitaiset vahvuudet ja heikkoudet osoittavat, että erilaiset NoSQL-tietokannat suoriutuvat eri tavalla riippuen niiden ominaisuuksista ja käyttötarkoituksesta [15].

Kuvassa 3.6 on esitetty Gupta et al. [35] vertailu edellä mainittujen kategorioiden ei-toiminnallisista ominaisuuksista. Huomionarvoista ominaisuuksissa on se, että ne ovat osittain yhteneväisiä alaluvussa 3.3 esiteltyjen IoT-järjestelmän tietokannan valintakriteerien kanssa. Kuvasta 3.6 voimme päätellä, että yksinkertaisille tiedoille, jotka voidaan tallentaa arvo-avaintietona, on arvo-avaintietokanta hyvä valinta, koska se tarjoaa joustavuutta, korkeaa suorituskykyä ja skaalautuvuutta [35]. Mikäli tieto voidaan tallentaa sarakkeisena ja se on jotakuinkin jäsenneltyä, on sarakkeinen tietokanta hyvä valinta, koska se tarjoaa korkeaa suorituskykyä ja skaalautuvuutta [35]. Gupta et al. [35] toteavat, että dokumenttitietokanta kannattaa valita silloin kun tieto pystytään tallentamaan JSON-formaatissa, koska myös dokumenttitietokanta tarjoaa skaalautuvuutta, usein korkeaa suorituskykyä ja joustavuutta. Jos tallennettavan tieto on vahvasti toisiinsa liittyvää tai se voidaan esittää graafina, kannattaa tieto tallentaa graafitietokantaa, koska se tarjoaa korkeaa vakautta ja vaihtelevaa suorituskykyä ja skaalautuvuutta [35]. Tässä alaluvussa esitellään avain-arvo-tietokannat 3.5.1, sarakkaiset tietokannat 3.5.2 ja dokumenttitietokannat 3.5.3. Graafitietokantoihin paneudutaan tarkemmin omassa luvussaan 4.

Tietomalli	Kyselyiden suorituskyky	Datan skaalautuvuus	Skeeman joustavuus	Tietokannan rakenne	Arvojen monimutkaisuus
Arvo-avain	Korkea	Korkea	Korkea	Pääavoin ja arvo	Ei ollenkaan
Sarakkeinen	Korkea	Korkea	Kohtalainen	Rivi sisältäen usean sarakkeen	Matala
Dokumentti	Korkea	Vaihteleva (Korkea)	Korkea	JSON puumuodossa	Matala
Graafi	Vaihteleva	Vaihteleva	Korkea	Graafi entiteettejä ja suhteita	Korkea

Kuva 3.6: NoSQL-tietokantojen ei-toiminnallisia ominaisuuksia [35]

3.5.1 Avain-arvotietokannat

Avain-arvotietokannoissa tieto tallennetaan joko muistiin tai levylle ja tallennettu tieto voidaan hajauttaa [94, luku 1]. Tieto tallennetaan avain-arvopareina ja avain osaa käytetään tiedon hakemiseen, koska se on arvoltaan yksilöllinen eli uniikki [94, luku 1] [11]. Arvo sisältää tallennettavan tiedon, joka voi olla jotain yleistä tietotyyppiä, objekti tai dokumentti [11]. Avain-arvotietokannat ovat yksinkertaisia, joten niistä hakeminen on nopeaa [11] [94, luku 1]. Nopeudesta johtuen avain-arvotietokantoja käytetäänkin usein välimuisteina [36]. Sekä Amghar et al. [11] et-tä Tiwari [94, luku 1] listaavat Rediksen, Berkeley DB:n ja Amazonin DynamoDB:n esimerkeiksi avain-arvotietokannoista. Amghar et al. [11] toteavat tekemässään vertailussa, että Redis soveltuu huonoiten IoT-järjestelmän tietokannaksi, koska se ei skaalaudu yhtä hyvin kuin vertailun muut NoSQL-tietokannat.

3.5.2 Sarakkeiset tietokannat

Sarakkeisissa tietokannoissa tieto tallennetaan sarakeorientoituneesti toisin kuin relaatiotietokannoissa, jossa tieto tallennetaan riviorientoituneesti [11] [94, luku 1] [36]. Sarakeorientoitunut tapa tallentaa tietoa mahdollistaa sen, että tyhjää tietoa ei tarvitse tallentaa niille sarakkeille, joilla ei ole mitään tietoa [94, luku 1]. Tämä tekee tiedon tallentamisesta tehokkaampaa [94, luku 1]. Sarakkeisiin tietokantoihin tieto tallennetaan arvo-avain-pareina ja jokaisella tallennetulla tiedolla on riviavain [94, luku 1] [11].

Sarakkeet voidaan ryhmitellä sarakeperheisiin, jotka toimivat avaimena tiedolle, jota sarakkeisiin on tallennettu [94, luku 1]. Sarakkeisen tietokannan lajiteltu rakenne mahdollistaa erittäin nopean tiedon haun riviavaimella [94, luku 1] [36]. Tiwarin [94, luku 1] mukaan sarakkeiset tietokannat hyödyntävät hajautettua tiedostojärjestelmää, joka mahdollistaa tiedon tallentamisen ja hajauttamisen useammalle levylle ja koneelle. Hassan [36] esittää, että sarakkeiset tietokannat soveltuvat tiedon louhinta ja analytiikka sovelluksiin. Avoimen lähdekoodin Hbase, Apachen Cassandra ja Googlen Bigtable ovat esimerkkejä suosituista sarakkeisista tietokannoista [94, luku 1] [11] [35] [36].

3.5.3 Dokumenttitietokannat

Dokumenttitietokantoihin tieto tallennetaan kokonaisina dokumentteina [11] [94, luku 1]. Usein käytetty formaatti dokumenttien serialisointiin on Javascript Object Notation (JSON) [11] [94, luku 1] [35]. Serialisoinnilla tarkoitetaan prosessia, jossa tietorakenne tai objekti muutetaan muotoon, jossa se voidaan siirtää kaapelia pitkin tai tallentaa myöhempää käyttöä varten [29]. Dokumenttien rakenteen ei tarvitse ol-la samanlainen, joten jokaiseen dokumenttiin voidaan tallentaa eri tietoja [11]. Dokumenttitietokannant mahdollistavat dokumenttien indeksoinnin sekä dokumentin päätunnisteen perusteella että ominaisuuksien perusteella [94, luku 1]. CouchDB ja MongoDB ovat esimerkkejä dokumenttitietokannoista [11] [94, luku 1] [35] [36].

Dokumenttitietokannat soveltuvat erinomaisesti IoT-järjestelmien tietokannoiksi [11]. Amghar et al. [11] osoittavat vertailussaan, että MongoDB:n vahvuuksiin lukeutuvat tietojen yhdistäminen ja paikkatietojen käsittely, jotka ovat tärkeitä tekijöitä IoT-järjestelmän tietokantaa valittaessa. Asiminidis et al. [14] väittävät edellisestä poiketen, että MongoDB ei sovellu suurien IoT-tietomäärien yhdistelyyn. Sekä Amghar et al. [11] että Asiminidis et al. [14] kuitenkin toteavat, että MongoDB on nopea käsittelemään lukukyselyjä, jotka kohdistuvat suureen määrään tietoja. Hassan [36] ja Gupta et al. [35] toteavat dokumenttitietokantojen soveltuvan huonosti tilanteeseen, jossa tietokannassa on paljon relaatioita tai normalisointia .

4 Graafitietokannat

Tässä luvussa keskitytään NoSQL-tietokantoihin luokiteltaviin verkko- eli graafitietokantoihin. Alaluvussa 4.1 määritellään graafitietokanta niiden käyttämän tietomallin mukaisesti. Alaluvussa 4.2 esitellään lyhyesti muutamia suosittuja graafitietokantajärjestelmiä. Alaluku 4.3 keskittyy graafitietokantojen kyselykieliin.

4.1 Graafitietokannan tietomalli

Graafi- tai verkkotietokannoilla on yleisesti haluttu käsitellä esimerkiksi sosiaalisia verkostoja, petosten havaitsemisia, reaaliaikaisia suosituksia, master datan hallintaa ja hierarkisia suhteita [47] [15]. Graafitietokantojen suosio IoT-järjestelmien tietokantana on kuitenkin kasvussa [32]. Robinson et al. [84, luku 1] korostavat, et-tä graafitietokantoja hyödyntääkseen ei tarvitse ymmärtää verkkoteoriaa, mutta on kuitenkin ymmärrettävä mitä graafi on. Robinson et al. [84, luku 1] määrittelevätkin graafin joukoksi solmuja ja kaaria. Solmut ovat graafien entiteettejä ja ne ovat yhteydessä toisiinsa kaarien avulla, jotka kuvaavat entiteettien välisiä suhteita [11] [98] [84, luku 1]. Graafitietokanta on tietokanta, joka tukee Create (suom. luonti), Read (suom. luku), Update (suom. päivitys) ja Delete (suom. poisto) (CRUD) toimintoja graafiseen tietomalliin perustuen [84, luku 1] [81]. Robinson et al. [84, luku 1] väittävät, että graafitietokannat ovat yleensä optimoituja transaktioiden suorituskykyä, eheyttä ja saatavuutta silmällä pitäen, joten niitä käytetäänkin usein transaktiojärjestelmien yhteydessä [81]. Graafitietokannat tukevat relaatiotietokantojen yhteydessä alaluvussa 3.4 esiteltyjä ACID-periaatteita [81] [35]. Osittain ACID-periaatteiden tukemisesta johtuen kirjoitusnopeudet ovat yleisesti hitaampia kuin osassa muita NoSQL-tietokantoja [81].

Graafitietomalleista suosituin ja käytetyin on labeled property graph model (suom. otsikoitu ominaisuussisältöinen graafitietomalli), joka mahdollistaa erittäin laajasti eri skenaarioiden mallintamisen [84, luku 3] [12]. Otsikoidun ominaisuussisältöisen graafitietomallin mukaisesti tieto tallennetaan graafitietokannassa solmuihin ominaisuuksina, jotka käytännössä toimivat avain-arvovarastoina [98] [84, luku 3] [13, luku 1]. Solmut ovat nimiöityjä otsikolla, joka auttaa kuvaamaan solmun

sisältämää tietoa ja roolia graafissa [98] [84, luku 3] [13, luku 1]. Anthanu [13, lu-ku 1] esittää, että solmulla voi olla useampi kuin yksi otsikko, jolloin solmulla on useampi rooli graafissa. Angles et al. [12] kutsuvat otsikoitua ominaisuussisältöistä graafitietomallia, jolla on useampi kuin yksi otsikko tai useampi arvo samassa ominaisuudessa, multi-valued property graph:ksi (suom. moniarvoinen ominaisuussisältöinen graafitietomalli). Solmuilla voi olla yksi tai useampi kaari tai suhde, jotka sisältävät tiedon solmujen välisestä suhteesta, minkä lisäksi niihin voidaan tallentaa muutakin solmuihin liittyvää tietoa [32] [11] [98] [13, luku 1]. Anthapu [13, luku 1] kuvailee graafeille ominaisia solmujen välisiä suhteita eli kaaria seuraavasti:

- Kaari liittää alku- ja loppusolmun kuvaten niiden välistä suhdetta.
- Kaaren suunta voi olla joko solmuun tuleva tai siitä lähtevä.
- Kaaren suunta kuvaa solmujen välisen suhteen suuntaa.
- Kaarella on tyyppi, joka kuvaa solmujen välistä yhteyttä.
- Kaarella voi olla arvo-avaintiedostoista koostuvia ominaisuuksia.
- Kaaren ominaisuudet kuvaavat solmujen välistä suhdetta.

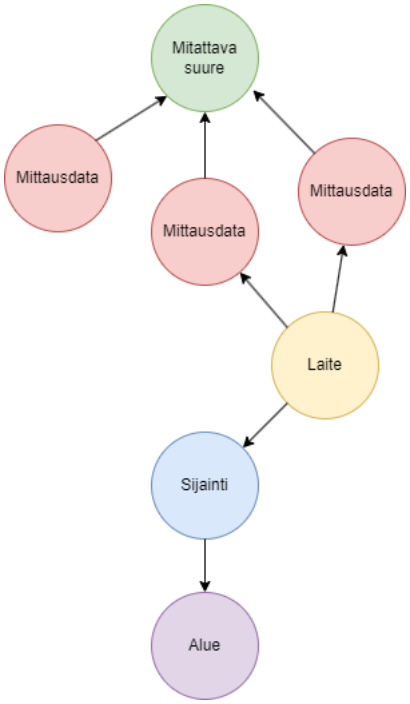
Kuvassa 4.1 on esitetty Silva et al. [28] yksinkertainen havainnollistava esimerkki solmuista ja niiden välisestä suhteesta. Kuvassa 4.1 A, Henkilö-solmu, omistaa Talo-solmun B. Kuvan 4.1 kaaren tyyppi on omistaa, joka yhdessä kaaren suunnan kanssa kertoo sen, että A-solmu omistaa B-solmun [13, luku 1] [28]. Kuten edellä mainittiin, voi sekä solmuilla että kaarilla olla ominaisuuksia, jotka kuvaavat paremmin solmua tai kaarta, ja johon voidaan tallentaa yksilöllistä tietoa. Kuvan 4.1 kaltaisessa esimerkissä Henkilö-solmulla voisi olla muun muassa seuraavia ominaisuuksia etunimi, sukunimi, syntymäaika, id [34, luku 1]. Väri, asuinpinta-ala, rakennusvuosi ja katuosoite voisivat olla Talo-solmun ominaisuuksia. Esimerkki kaaren ominaisuudesta voisi sen sijaan olla omistussuhteen alkamispäivämäärä.

Kuva 4.1: Henkilö-solmu ja omistaa yhteys Talo-solmuun [28]

Fosicin ja Solicin [32] mukaan graafitietokannoilla on tehokasta löytää ja määritellä monimutkaisia keskinäisiä riippuvuuksia ja suhteita solmujen välillä [84, luku 3]. Ne ovat myös helposti ja tehokkaasti mallinnettavia ja datanavigoitavia, minkä vuoksi niitä käytetään sosiaalisten verkostojen kuvaamiseen [32]. Graafitietokannat soveltuvat erinomaisesti IoT-järjestelmien tiedonhallintaan, koska niillä on yksinkertaista määritellä laitteiden, käyttäjien ja IoT-järjestelmän tietojen välisiä suhteita [98] [15]. Graafitietokannan hyöty kasvaa, kun solmujen välinen liitännäisyys kasvaa [15].

Kuvassa 4.2 onkin esitetty osa Fosicin ja Solicin [32] IoT-järjestelmän graafitietokannan tietomalli. Tietomalli on identtinen alaluvussa 3.4 ja kuvassa 3.5 esitettyyn tietomalliin, joka on rakennettu relaatiotietokantaan. Tietomallissa vihreällä taustalla oleva mitattava suure-solmu kertoo mitattavan suureen, kun taas punaisella olevat mittausdatasolmut sisältävät varsinaisen mittausdatan [32]. Mittausdatasta mitattavaan suureeseen lähtevä kaari osoittaa sen, että mittausdata sisältää mitattavan suureen. Keltaisella värillä oleva laitesolmu kuvaa mittauksen suorittavaa havainnointikerroksen sensoria [32]. Keltaisesta laitesolmusta mittausdataan lähtevä kaari eli suhde visualisoi laitteen olevan vastuussa mittausdatasta. Laitesolmusta lähtee myös kaari siniseen sijaintisolmuun, joka havainnollistaa laitteen sijaintia. Sijainti-ja aluesolmun yhdistävä kaari kuvailee sitä, että sijainti on tietyllä alueella. Fosic ja Solic [32] väittävät tietomallin visualisoivan hyvin IoT-järjestelmän tietoa, koska mallista on nopeasti nähtävissä, mikäli jokin laite ei ole kerännyt yhtään mittausdataa, sillä sille ei muodostuisi ollenkaan suhdetta mittausdata solmuun.

Graafitietokannan tietomalli on joustava, koska se mahdollistaa uusien solmujen ja suhteiden lisäämisen malliin ilman, että se vaikuttaisi jo tietomallissa olevaan tietoon [84, luku 3]. Tämän johdosta graafitietokantaan ei tarvitse tehdä migraatioita, kun siihen lisätään uusia solmuja, suhteita tai niiden ominaisuuksia [84, luku 3]. Uusille solmuille ja suhteille ei ole välttämätöntä lisätä datatyyppiä kuten relaatiotietokannnoissa [31]. IoT-järjestelmässä on tyypillistä, että laitteita eli solmuja lisätään ja poistetaan, minkä lisäksi eteen voi tulla tilanne, että halutaan lisätä uusi mitattava suure [32]. Voidaankin todeta, että graafitietokanta täyttää valintakriteereistä tiedon heterogeenisyyden tukemisen. Azad et al. [15] mainitsevat, että graafitietokannat eivät sovellu hyvin IoT-järjestelmän tietokannaksi, mikäli järjestelmässä on vain yksi dataa keräävä sensori, joka on yhdistettynä pilveen. Graafitietokannat tukevat monimutkaisia tietokantakyselyitä, jotka ovat suhteessa nopeampia kuin muissa tietokannoissa [15]. Graafitietokantojen kyselykieliin perehdytäänkin alalu-



Kuva 4.2: Esimerkki IoT-järjestelmän tietomallista graafitietokannassa [32]

vussa 4.3.

4.2 Graafitietokantajärjestelmiä

Tässä alaluvussa on esitelty lyhyesti muutamia suosittuja graafitietokantajärjestelmiä. Graafitietokantajärjestelmiä hyödynnetään erityisesti petosten havaitsemisessa, suositusjärjestelmissä ja sosiaalisten verkkojen kuvaamisessa [47] [15] [31]. Graafitietokantojen suosio on kuitenkin yleisesti nopeassa kasvussa, jonka myötä niitä hyödynnetään yhä useammalla teollisuuden alalla [31] [32].

4.2.1 AllegroGraph

AllegroGraph on graafitietokanta, joka perustuu ominaisuussisältöisen graafitietomallin sijasta Resource Description Frameworkiin (RDF) [31]. RDF on W3C:n suosittelema malli linkitetyn tiedon serialisointiin [84, liite A]. AllegroGraph tukee kyselykielistä SPARQL:ää ja Prologia [31]. AllegroGraph on helposti skaalattavissa, mutta sen käyttö on keskittynyt maantieteellisen tiedon käsittelyyn ja sosiaalisten verkkojen analyyseihin [31].

4.2.2 ArangoDB

ArangoDB tukee useita tietomalleja [31]. Graafitietomallin lisäksi se tukee avain-arvopareja ja dokumentteja [31]. ArangoDB Query Language (AQL) on ArangoDB:n ainoa tuettu kyselykieli, mutta sillä voidaan hakea yhdellä haulla tietoa käyttäen jokaista edellä mainittua tietomallia, mikä tekee tietojen yhdistelystä helpompaa [31]. ArangoDB:stä on saatavilla kaksi versiota, ilmainen Community-versio ja maksullinen Enterprise-versio kaupalliseen käyttöön [31]. ArangoDB:n tyypillinen sovellusalue on petosten havaitseminen [31].

4.2.3 InfiniteGraph

InfiniteGraph on nopeasti kasvava graafitietokanta, jota hyödynnetään muun muassa petosten havaitsemiseen, asiakkuushallintajärjestelmissä ja sosiaalisissa verkostoissa [31]. InfiniteGraphin tietomalli on moniarvoinen ominaisuussisältöinen graafitietomalli [31]. Tietokanta voidaan jakaa useaan osaan ja hajauttaa usealle levylle [31]. InfiniteGraph tukee REST-rajapinnan kautta tehtävien kyselyjen lisäksi omaa DO-kyselykieltä [39] [31]. InfiniteGraphista on ladattavissa maksuton version, jonka lisäksi saatavilla on kaupalliseen käyttöön erilaisia lisenssejä [39] [31].

4.2.4 OrientDB

OrientDB on avoimen lähdekoodin NoSQL-tietokanta, joka tukee graafitietomallin lisäksi dokumentteja, avain-arvopareja ja objekteja [31]. Poiketen muista graafitietokannoista OrientDB hyödyntää dokumentteja ja objekteja fyysisten solmujen tallentamiseen [31]. Tietokanta tukee Gremlin-kyselykielen lisäksi SQL:ää ja Javaa [31]. OrientDB:stä on saatavilla ilmainen, mutta hieman rajoitettu, Community-versio, jonka lisäksi on tarjolla maksullinen Enterprise-versio kaupalliseen käyttöön [31]. OrientDB:tä hyödynnetään muun muassa master datan hallinnassa, petosten havaitsemisessa ja suositusjärjestelmissä [31]. OrientDB on asennettavissa ainoastaan hajautetusti [31].

4.2.5 Neo4J

Neo4J on suosituin graafitietokannoista, ja se on helposti saavutettavissa, tehokas ja skaalautuva [28] [31]. Vuonna 2007 luotu Neo4J perustuu avoimeen lähdekoo-diin [11] [31]. Neo4J:llä on helppo hakea solmujen linkitettyä dataa ja niiden välisiä

suhteita ilman SQL-kielen JOIN-lausekkeita, minkä vuoksi se sopiikin erinomaisesti edellä mainittujen sosiaalisten verkostojen kuvaamiseen [28] [32]. Neo4J tukee kyselykielistä Cypheria ja Gremliniä [32]. Neo4J:n tyypillisiä sovelluskohteita ovat esimerkiksi reaaliaikaiset suositusjärjestelmät, identiteetti- ja pääsynhallinta, petosten havaitseminen, tekoäly ja IoT [31]. Neo4J:tä käytetäänkin maailmanlaajuisesti eri teollisuuuden aloilla kuten esimerkiksi terveydenhuollossa, auto- ja sotateollisuudessa [31]. Neo4J:sta on tarjolla myös sulautettu versio, jota voidaan käyttää suoraan sulautetulla laitteella [31].

Neo4J tarjoaa graafitietokantaa PaaS-palveluna Neo4J AuraDB nimellä [72]. Tämä tarkoittaa sitä, että Neo4J vastaa graafitietokannan ylläpidosta ja päivityksistä. AuraDB:stä on saatavilla kolme eri tilausta, jotka ovat AuraDB Free, AuraDB Professional ja AuraDB Enterprise [75]. AuraDB Free on tarkoitettu pieniin projekteihin, oppimiseen ja kokeiluihin, kun taas AuraDB Professional ja Enterprise tilaukset ovat kaupalliseen ja tuotantokäyttöön suunnattuja [75]. AuraDB Free käyttö on ilmaista, mutta tilauksessa on rajoitettuna noodien lukumäärä kahteensataantuhanteen ja suhteiden lukumäärä neljäänsataantuhanteen [72]. AuraDB Free sisältää Neo4J:n kaikki perustoiminnallisuudet kuten esimerkiksi Neo4J-käyttöliittymän verkon kautta, ACID-yhteensopivuuden ja tuen Cypher-kyselykielelle [72]. Ilmaisessa tilauksessa ei voi luoda kuin yhden Neo4J-tietokannan, kun taas maksullisissa Professional ja Enterprise tilauksissa ei ole rajoitusta tietokantojen lukumäärässä [72].

Neo4J AuraDB palvelun lisäksi Neo4J:llä on Community Edition-versio tietokantapalvelimestaan, jonka käyttö on ilmaista ei-kaupallisessa tarkoituksessa nojaten GPLv3 lisenssiin [75] [17] [31]. Community Editionin lisäksi tarjolla on maksullinen Enterprise-versio ja valtioille suunnattu Government-versio [75][31]. Neo4J-tietokantapalvelimen Community Edition-version ylläpitäminen virtuaalikoneessa tai docker-kontissa, esimerkiksi Azuren tai AWS:n pilvipalvelussa, ovat vaihtoehtoja AuraDB:lle, mutta käytännössä Neo4J suosittelee verkkosivuillaan AuraDB:tä toteutettavan IoT-järjestelmän kaltaiseen kokeiluun ja protoiluun [72]. Neo4J:llä on kattavat dokumentaatiot sekä AuraDB:lle että tietokantapalvelimilleen.

4.3 Graafitietokannan kyselykielet

Holzschuher ja Peinl [38] esittävät, että graafitietokannat ovat NoSQL-tietokantojen joukossa erityisen kiinnostavia, koska poiketen useista avain-arvo- ja dokumentti-

tietokannoista, graafitietokannat tarjoavat kunnolliset kyselykielet. Graafitietokannat on suunniteltu tallentamaan tietoa, josta muodostuu toisiinsa liittyneitä verkko-ja ja suhteita, jonka vuoksi kyseisen tiedon hakeminen on sieltä nopeaa ja tehokasta [38] [32]. Relaatiotietokannoissa vastaavanlaisen, paljon suhteita ja liitännäisyyksiä sisältävän, mallintaminen vaatii paljon monen suhde moneen relaatioita [38]. Tämä johtaa siihen, että tiedon hakemiseen relaatiotietokannasta SQL-kyselykielellä vaaditaan monimutkaisia JOIN-lausekkeita [38].

Graafitietokannoille ei ole muodostunut SQL:n kaltaista kyselykieltä, joka olisi käytössä kaikissa graafitietokannoissa, vaikka monet tietokannat tukevat useampia kyselykieliä [84, luku 3]. Suosituimpia ja käytetyimpiä kyselykieliä ovat SPARQL, Cypher ja Gremlin, jotka kaikki eroavat toisistaan, vaikka perustuvatkin hahmonsovitukseen ja graafissa kulkemiseen eli navigaatioon [84, luku 3] [12]. Hahmonsovitus perustuu graafin muotoisen kyselyn luomiseen, jonka avulla pyritään löytämään graafitietokannasta vastaavan muotoinen graafi [12]. Hahmonsovitus mahdollistaa relaatioille tyypillisten operaatioiden kuten tulosten yhdistämisen, jolla voidaan rajata tulosten joukkoa halutunlaiseksi [12]. Navigaatiolla tarkoitetaan graafissa matkaamista, joka voidaan toteuttaa polkuihin perustuvilla kyselyillä [12].

Angles et al. [12] kuvailevat tyypillisen polkukyselyn hakevan sosiaalisessa verkostoa kuvaavasta graafista ystävän ystävää eli polkua, jossa solmut ovat ystävyyttä kuvaavilla suhteilla, eli kaarilla, kytkettyjä toisiinsa. SPARQL:ää käytetään Resource Description Framework (RDF) tietomallin mukaisen tiedon kyselyyn, kun taas Cypher on suunnitelty ominaisuussisältöisten graafitietomallien kyselyihin [12] [84, luku 3]. Gremlin on luonteeltaan imperatiivinen ja se perustuu enemmän navigaatioon kuin hahmonsovitukseen eikä se muistuta syntaksiltaan SQL:ää [12] [38] [32]. Robinson et al. [84, luku 3] ovat sitä mieltä, että Cypher on graafitietokantojen kyselykielistä helpoiten opittava ja hyvä pohja muiden kyselykielten oppimisille. Amghar et al. [11] väittävät, että Cypher on lyhyempää ja helpommin ymmärrettävää kuin SQL-kieli, jota se hieman muistuttaa syntaksiltaan.

Cypher koostuu ehdoista, jotka muodostetaan avainsanoilla kuten useissa muissakin kyselykielissä [84, luku 3] [13, luku 2]. Match-avainsanalla voidaan määritellä haettava asia graafitietokannasta [84, luku 3] [13, luku 2]. Match-avainsanan rinnal-la usein käytetään SQL:stäkin tuttua WHERE-avainsanaa, jolla voi rajata palautettavien solmujen määrää [84, luku 3] [13, luku 2]. RETURN-avainsanalla määritetään se, että mitä Cypher-kyselyllä halutaan palauttaa [13, luku 2]. Hyvin yksinkertainen Cypher-kysely, jolla palautetaan kaikki tietokannassa sijaitsevat elokuvat, joiden ni-

mi on "Minun elokuvani"näyttäisi tältä:

```
MATCH (e:Elokuva)
WHERE e.nimi = 'Minun elokuvani'
RETURN e
```

Edellä mainittujen ehtojen lisäksi Cypheristä löytyy ehtoja muun muassa solmujen ja suhteiden luomiselle (CREATE), poistamiselle (DELETE), solmujen suhteiden ja ominaisuuksien asettamiselle (SET) ja poistamiselle (REMOVE) [13, luku 2]. MERGE-avainsanalla luodaan solmu vain, jos sitä ei graafitietokannasta ennestään löydy [13, luku 2]. ORDER BY-avainsanalla voidaan järjestää palautettavat solmut tai ominaisuudet haluttuun järjestykseen ja LIMIT-avainsanalla voidaan rajata palautettavien lukumäärää [13, luku 2]. Graafitietokannat ovat erinomaisia kuvaamaan asioiden välisiä suhteita. Cypher-kyselykielellä solmujen välinen suhde voidaan muodostaa seuraavasti, kun halutaan luoda uusi Henkilösolmu ja määrittää sille ja edellisen esimerkin Elokuvasolmulle suhde On ohjannut [13, luku 2] [84, luku 3]:

```
MATCH (e:Elokuva {nimi: 'Minun elokuvani'})
MERGE (h:Henkilö {nimi: 'Olli'})
MERGE (h)-[:ON_OHJANNUT]->(e)
```

Koska kyselyssä käytetään MERGE-avainsanaa, ei uutta Henkilösolmua luoda, jos se jo tietokannasta löytyy. Myös solmujen välinen suhde voidaan luoda MERGE-avainsanalla, jolloin suhdetta ei muodosteta solmujen välille, jos se jo niiltä ennestään löytyy [13, luku 2]. Yksinkertaisesta esimerkistä huomaamme, että Cypher-kyselykielessä solmujen välisten suhteiden suuntaa kuvataan merkeillä > ja < [84, luku 3]. Suunnalla on merkitystä, sillä se kuvaa esimerkissämme sitä, että henkilö on ohjannut elokuvan eikä elokuva ole ohjannut henkilöä. Cypher tukee useita yleisiä tietotyyppejä kuten numeroita, merkkijonoja, booleaneja, päivämääriä solmujen ja suhteiden ominaisuuksina [13, luku 2].

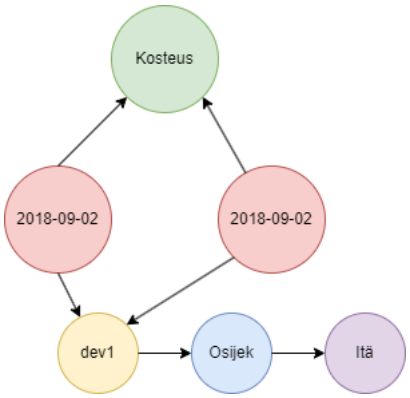
Alaluvuissa 3.4 ja 4.1 esiteltiin Fosicin ja Solicin [32] IoT-järjestelmän tietomalli relaatiotietokantatauluihin 3.5 ja graafiin 4.2 mallinnettuina. Alaluvussa 3.4 esitettiin edellä mainittuun tietomalliin tehty suhteellisen pitkä SQL-tietokantakysely, jolla haettiin yhden laitteen (dev1) kaikki kosteusmittaukset (humidity) tietyllä alueella (east). Fosic ja Solic [32] esittävät saman kyselyn myös Cypher-kyselykielellä:

```
MATCH
p = (d: Device) - [*] -> () - [] -> (t: Type),
w = (d: Device) - [: Below * 2]
WHERE
d.name = "dev1" AND t.name = "humidity"
RETURN p, w
```

Näemme heti Cypher-kyselystä, että se on paljon lyhyempi kuin vastaavan asian hakeminen SQL-kyselyllä, vaikka tietomalli on alkujaan suunniteltu relaatiotietokantaa varten. Yksi graafitietokannan vahvuuksista on se, että tietomallia on yksinkertaista muokata lisäämällä tai poistamalla solmuja, ominaisuuksia tai suhteita vaikuttamatta jo tallennettuihin tietoihin. Fosic ja Solic [32] osoittavatkin, että mikäli graafitietokannan tietomalliin lisää mittausdatalle ominaisuudeksi mitattavan suureen, tulee kyselystä vielä yksinkertaisempi:

```
MATCH
p = (d: Device) - [*] -> (m: Measurements)
WHERE
d.name = "dev1" AND m.type = "humidity" AND
d.Region = "east"
RETURN p
```

Käyttipä sitten lyhyempää tai pidempää Cypher-kyselyä, on tuloksena Fosicin ja Solicin [32] mukaan kuvan 4.3 graafi. Graafista voidaan nähdä mitatun suureen (kosteus), kaksi päivämäärää sisältävää mittaustulosta, mittauksen suorittaneen laitteen (dev1), kaupungin (Osijek) ja alueen (itä), jossa laite on havainnot tehnyt. Sama kyselyn tulos voidaan visualisoida myös taulukkona.



Kuva 4.3: Cypher-kyselyn tulos graafitietokannassa [32]

5 Arkkitehtuuripäätösten dokumentaatiomalli

Tässä luvussa käsitellään arkkitehtuuripäätösten dokumentointia. Alaluvussa 5.1 esitellään lyhyesti syyt, minkä takia arkkitehtuuripäätösten dokumentointi on tärkeää. Alaluvussa 5.2 esitellään olemassa olevia arkkitehtuuripäätösten dokumentaatiomalleja. Alaluvussa 5.3 esitellään uusi dokumentaatiomalli, joka yhdistelee olemassa olevia dokumentaatiomalleja.

5.1 Arkkitehtuuripäätösten dokumentoinnin motivaatio

Sovellusten ja tietojärjestelmien lisääntyneen kompleksisuuden vuoksi ohjelmistoarkkitehtuurin merkitys on kasvanut [99]. Ohjelmistoarkkitehtuurin on perinteises-ti mielletty koostuvan suunnitelluista malleista ja niiden toteutustavoista, kun taas malleihin johtaneet päätökset, ja päätösten perustelut, ovat jääneet arkkitehtuurin ulkopuolelle [99] [96]. Perinteiset arkkitehtuurimallit eivät dokumentoi hyvin muutosta, vaikutuksia, päätöksiä, päätöksiin johtaneita perusteluita, eivätkä ne myöskään tarjoa jäljitettävyyttä tavoitteiden ja arkkitehtuurin elementtien välillä [96]. Van Heesch et al. [99] korostavatkin, että ohjelmistoarkkitehtuurin suunnittelussa tehdyt päätökset, ja päätöksiin johtaneet pohdinnat, ovat olennainen osa ohjelmistoarkkitehtuuria [40]. Tyree ja Akerman [96] nostavat arkkitehtuuripäätökset ensiarvoisen tärkeiksi, sillä ne voivat auttaa sidosryhmiä, kuten esimerkiksi asiakkaita, kehittäjiä ja muita arkkitehtejä, ymmärtämään toteutettua arkkitehtuuria, mikäli päätökset on dokumentoitu ja jaettu hyvin. Hyvin dokumentoidut arkkitehtuuripäätökset auttavat myös tulevia projekteja, sillä osa tehdyistä päätöksistä voi olla kopioitavissa suoraan [99]. Vaikka päätökset eivät olisi kopioitavissa suoraan, voidaan niistä havaita vaihtoehtoja omiin päätöksiin tai kokonaan uusia päätöksiä, jotka eivät ole vielä tulleet vastaan [99]. Päätösten dokumentointiin ei ole kuitenkaan yhtä yleisesti hyväksyttyä mallia, joka soveltuisi kaikkiin projekteihin [99].

Kompleksinen ohjelmisto ja sen arkkitehtuuri voi koostua valtavasta määrästä päätöksiä [96]. Ohjelmistoarkkitehdin tulisi kuitenkin tehdä mahdollisimman vähän päätöksiä ja lykätä loput ohjelmiston myöhempään elinkaaren vaiheeseen, mikä mahdollistaa tasapainon teknisen organisaation ohjaamisessa ja rajoittamisessa

[96]. Arkkitehdin tulisi tehdä vain ne päätökset, jotka ovat ohjelmiston arkkitehtuurin osalta merkittäviä [96]. Tyree ja Akerman [96] ohjaavat arkkitehtiä kysymään itseltään, että vaikuttaako päätös yhteen tai useampaan järjestelmän ominaisuuteen kuten esimerkiksi suorituskykyyn tai turvallisuuteen? Mikäli vastaus on kyllä, tulee arkkitehdin tehdä päätös ja dokumentoida se kokonaisuudessaan. Päätösten kuvaaminen yksityiskohtaisesti on aikaa ja resursseja vievää, joten päätöksistä kannattaa tarkasti dokumentoida vain tärkeimmät päätökset ja muihin käyttää vähemmän aikaa [99] [96].

5.2 Nykyiset dokumentaatiomallit

Arkkitehtuuripäätösten dokumentointiin on ollut kolme erilaista lähestymistapaa päätössapluunat (decision templates), päätösmallit (decision models) ja muistiinpanot (annotations), jotka vastaavat tyydyttävästi osaan päätöksiin liittyvistä huolenaiheista, mutta eivät kuitenkaan kaikkiin [99]. Van Heesch et al. [99] esittelevätkin arkkitehtuuripäätösten dokumentointiin kehyksen, joka koostuu neljästä arkkitehtuuripäätöksen dokumentoivasta näkökulmasta päätöksen yksityiskohtainen kuvaus (a Decision Detail viewpoint), päätösten suhteiden kuvaus (a Decision Relationship viewpoint), päätösten kronologinen kuvaus (a Decision Chronology viewpoint) ja sidosryhmien päätökseen osallistumisen kuvaus (a Decision Stakeholder Involvement viewpoint). Van Heesch et al. [100] ovat täydentäneet dokumentaatiokehystään viidennellä näkökulmalla, joka kuvaa päätökseen vaikuttaneita voimia (a Decision Forces viewpoint). Jokainen näkokulma päätökseen dokumentoi tehtyä päätöstä omasta lähtökohdastaan, jonka vuoksi ne täydentävät toisiaan [99]. Dokumentaatiokehystä voi käyttää sellaisenaan tai sitä voi muokata lisäämällä omia näkökulmia tai muokata olemassa olevia kuvauksia [99].

Tässä tutkielmassa keskitytään päätösten yksityiskohtaiseen kuvaukseen, jolla voidaan dokumentoida päätösten perustelut yksityiskohtaisemmin kuin muilla kehyksen malleilla, jotka antavat vain yleiskuvan päätöksestä [99]. Päätöksen yksityiskohtainen näkökulma tai kuvaus muistuttaa Tyreen ja Akermanin [96] ja Jansenin ja Boschin [40] esittelemiä arkkitehtuuripäätöksen kuvaukseen tarkoitettuja dokumentaatiomalleja. Van Heesch et al. [99] huomauttavatkin, että ei ole olemassa yhtä yhteisesti hyväksyttyä mallia kuvaamaan arkkitehtuuripäätöksiä. Taulukossa 5.2 on esitetty Van Heesch et al. [99] malli päätöksen yksityiskohtaisesta kuvauksesta, joka muistuttaa Tyreen ja Akermanin [96] arkkitehtuuripäätösten dokumentaatiomallia.

Yhtäläisyyksiä on myös Jansenin ja Boschin [40] esittelemään päätösten dokumentointimalliin. Yksityiskohtaiseen kuvaukseen tulee päätöksen nimi. Nykyinen tila kertoo päätöksen tilan, joka voi olla esimerkiksi idea, kesken, hyväksytty ja hylätty [99] [96]. Päätösryhmä osoittaa, että kuuluuko kuvattava päätös johonkin tai useampaan ryhmään päätöksiä, jotka voidaan ryhmitellä esimerkiksi toiminnallisuuden tai arkkitehtuurin osa-alueen perusteella [99] [96]. Ongelmakenttään kuvataan sanallisesti päätöksessä ratkaistava ongelma [99] [96] [40]. Päätöskenttään kuvataan sanallisesti päätöksen lopputulos [99] [96]. Vaihtoehtoihin kirjataan vaihtoehtoiset ratkaisut ja totetustavat [99] [96]. Perusteluihin dokumentoidaan päätöksen oikeutus [99] [96]. Liittyviin päätöksiin listataan kaikki päätökset, jotka liittyvät käsiteltyyn päätökseen [99] [96]. Liittyvät vaatimukset kohtaan voidaan dokumentoida vaatimukset, jotka liittyvät käsiteltyyn päätökseen [99] [96]. Historiaan dokumentoidaan päätöksen historia sisältäen päätöksen siirtymät tilojen välillä [99].

Nimi	
Nykyinen tila	
Päätösryhmä	
Ongelma	
Päätös	
Vaihtoehdot	
Perustelut	
Liittyvät päätökset	
Liittyvät vaatimukset	
Historia	

Taulukko 5.1: Esimerkki yksityiskohtaisesta arkkitehtuuripäätöksen kuvauksen mallista [99].

Tyreen ja Akermanin [96] sekä Jansenin ja Boschin [40] dokumentaatiomallit sisältävät muutaman aiheen dokumentoinnin, joita Van Heeschin [99] päätöksen yksityiskohtainen kuvaus ei suoraan sisällä. Tyreen ja Akermanin mallissa oletuksiin kirjataan kaikki oletukset vallitsevasta päätöksentekoympäristöstä sekä oletukset, joilla on rajaavia vaikutuksia vaihtoehtoihin [96]. Tyreen ja Akermanin [96] sekä Jansenin ja Boschin [40] rajoitteet ovat nimensä mukaisesti rajoitteita ympäristöön, jotka mahdollinen päätös saattaa aiheuttaa. Arkkitehtuuripäätöksellä voi olla useita erilaisia seurauksia, ja ne voivat esimerkiksi johtaa tarpeeseen tehdä uusia päätöksiä, vaatimuksia tai muokata nykyisiä vaatimuksia, minkä vuoksi seuraukset do-

kumentoidaan sekä Tyreen ja Akermanin [96] että Jansenin ja Boschin [40] dokumentaatiomalleissa. Seurauksien lisäksi arkkitehtuuripäätös voi vaikuttaa liittyviin dokumentteihin, järjestelmiin ja suunnitelmiin [96]. Poiketen sekä Van Heesch et al. [99] että Tyreen ja Akermanin [96] malleista, Jansen ja Bosch [40] haluavat dokumentoida arkkitehtuuripäätökseen suunnittelun säännöt. Suunnittelun säännöillä kuvaillaan se, kuinka halutut ominaisuudet voidaan toteuttaa arkkitehtuuripäätöksen mukaisesti [40]. Van Heesch et al. [101] myöhemmin esittelemä malli, joka täydentää heidän alkuperäistä dokumentaatiomallia [99], arkkitehtuuripäätöksen dokumentoinnista ja arvioinnista demonstroi hyvin sen, että dokumentaation mallit ovat muokattavissa ja yhdisteltävissä, sillä malliin on yhdistelty sekä päätöksen yksityiskohtaista kuvausta että päätökseen vaikuttaneita voimia [100]. Päätökseen vaikuttaneista voimista on dokumentoitu päätöstä tukeneet seikat ja päätökseen negatiivisesti vaikuttavat seikat [101]. Lisäksi malliin on lisätty arviointi siitä, että oliko toteutettu päätös onnistunut [101]. Jansen ja Bosch [40] eivät dokumentoi mallissaan päätökseen vaikuttaneita voimia, mutta he sen sijaan haluavat dokumentoida päätöksen hyvät ja huonot puolet.

5.3 Uusi dokumentaatiomalli

Yksikään edellä mainittu ja esitelty dokumentaatiomalli ei suoraan sovellu jokaiseen toteuttavaan projektiin. Arkkitehtuuripäätösten yksityiskohtainen kuvaus kuvaa päätökset staattisesti ottamatta kantaa siihen, että kannattaako päätös tehdä ja oliko päätös lopulta onnistunut vai ei. Lisäksi osasta yksityiskohtaisista malleista puuttuu tieto siitä, että mitä hyötyjä ja haittoja päätöksellä on, ja miten edellä mainitut seikat vaikuttavat päätökseen. Sen sijaan arkkitehtuuripäätösten arvioinneista ja päätöksiin vaikuttaneiden voimien kuvauksista puuttuu yksityiskohtaisten kuvausten tuottama tarkempi tieto päätöksen taustoista.

Tästä johtuen on luotu arkkitehtuuripäätösten dokumentaatiomalli, joka yhdistää parhaiten soveltuvia osia arkkitehtuuripäätöksen yksityiskohtaisista kuvauksista [99] [96] [40], arkkitehtuuripäätökseen vaikuttaneiden voimien kuvauksesta [100] ja arkkitehtuuripäätöksen arvioinnista [101], kun tarkoituksena on toteuttaa IoT-järjestelmä graafitietokannalla. Dokumentaatiomallia on tarkoitus käyttää sekä päätösten tekemisen tukena että päätöksien dokumentoimiseen myöhempää käyttöä varten. Dokumentoiduilla päätöksillä on mahdollisuus auttaa tulevia graafitietokantaa IoT-järjestelmässään hyödyntäviä projekteja, mutta myös muita projekteja,

jotka tekevät arkkitehtuuripäätöksiä väliohjelmisto- ja sovelluskerrosten osalta.

Taulukossa 5.2 on esitetty malli yhden arkkitehtuuripäätöksen dokumentaation, joka koostuu kolmestatoista dokumentoitavasta aiheesta. Arkkitehtuuripäätöksen dokumentaatiota voi versioida, ja sitä voi muokata, kun päätökseen liittyvät asiat selkiytyvät tai muuttuvat. Dokumentaatiomallin nimeen dokumentoidaan arkkitehtuuripäätöstä kuvaava nimi. Tilakenttään kirjataan päätöksen tämän hetkinen tila, joka voi olla esimerkiksi uusi, käynnissä, hylätty tai valmis riippuen siitä, että onko päätös jo aloitettu tai valmis. Aihealuekenttään kirjataan päätöksen aihealue, jotta päätöksiä on mahdollista tarkastella aihealueittain. Aihealueet voivat olla vaatimustenmäärittelyssä tunnistettuja suurempia kokonaisuuksia tai ei-toiminnallisia vaatimuksia. Aihekenttään dokumentoidaan päätöksen kohteena oleva aihe lyhyes-ti ja kuvaavasti. Päätöskenttään dokumentoidaan tehty päätös ilman perusteluita. Jo pelkästään lukemalla edellä mainitut viisi kenttää, saadaan nopeasti käsitys siitä, että mitä arkkitehtuuripäätös koskee ja onko päätös jo tehty vai onko se vasta edes-sä. Dokumentaatiomallin alemmilla kentillä on tarkoitus avata päätöksen taustoja tarkemmin.

Nimi	
Tila	
Aihealue	
Aihe	
Päätös	
Perustelut	
Vaihtoehdot	
Seuraukset	
Liittyvät päätökset	
Päätöstä tukevat seikat	
Päätöstä vastustavat seikat	
Päätöksen lopputulos	
Lopputuloksen perustelut	

Taulukko 5.2: Dokumentaatiomalli arkkitehtuuripäätöksen kuvaukseen

Perusteiluihin kirjataan päätökseen johtaneet syyt, jotka avaavat tarkemmin sitä, että miksi kyseinen päätös on tehty tai hylätty. Päätöksen vaihtoehtoihin dokumentoidaan vaihtoehtoiset valinnat joita on päätöstä tehdessä harkittu ja huomioitu. Vaihtoehtojen dokumentointi on tärkeää, koska se osoittaa päätöksen lukijalle, että päätöstä tehdessä on harkittu muitakin vaihtoehtoja. Lisäksi vaihtoehdot saattavat auttaa myöhempiä projekteja löytämään vaihtoehtoisia toteutustapoja ja teknologiavalintoja. Dokumentaatiomallin seurauksiin kirjataan ne asiat, joihin tehty päätös vaikuttaa suoraan tai epäsuorasti [96] [40]. Erityisen tärkeää on kirjata seurauksiin päätöksestä mahdollisesti aiheutuvat rajoitteet, koska luodussa dokumentaatiomallissa ei ole erikseen kenttää rajoitteiden dokumentaatiolle. Päätöksen seurauksena voi muodostua myös uusia päätöksiä, joista kannattaa tehdä uusi oma päätöksen dokumentaatio [96]. Liittyviin päätöksiin dokumentoidaan päätökset, jotka liittyvät dokumentoituun päätöksen. Mikäli päätöksiä on lukuisia, voidaan luettavuutta parantaa taulukoimalla liittyvät päätökset [96].

Päätöstä tukeviin seikkoihin kirjataan luettelona ne asiat, jotka ohjaavat tai ovat ohjanneet tehtyä päätöstä kohti [101]. Päätöstä vastustaviin seikkoihin luetteloidaan vastaavasti ne asiat, jotka ovat päätöstä vastaan [101]. Päätöstä tukevat ja vastustavat seikat toimivat myös pohjana päätösten perusteluille. Jokainen sekä päätöstä tukevat että vastustavat seikat kannattaa pisteyttää, jotta voidaan arvioida yksinkertaisesti laskemalla, kannattaako päätös tehdä vai ei [100]. Pisteytys kannattaa pitää yksinkertaisena, jotta päätöksenteon ja seikkojen listaus ei kasvata dokumentoinnin työmäärää ja kompleksisuutta. Taulukossa 5.3 on esitetty yksinkertainen pisteytyksen malli, joka perustuu Van Heesch et al. [100] päätökseen vaikuttaneiden voimien arviointiin. Jokainen päätöstä tukeva seikka saa yhdestä kolmeen pluspistettä riippuen siitä, kuinka voimakkaasti se tukee päätöstä. Päätöstä vastustavat seikat saavat miinuspisteitä yhdestä kolmeen riippuen siitä, kuinka voimakkaasti ne ovat vastustavat päätöstä. Päätös kannattaa tehdä, jos lopputulos on positiivinen, kun plussat ja miinukset on laskettu yhteen. Dokumentaatiomallin 5.2 lopputuloskenttään kirjataan päätöksen lopputulos siinä vaiheessa, kun se on ensimmäisen kerran arvioitavissa. Lopputulos voidaan kirjata asteikolla epäonnistunut, kesken ja onnistunut. Osa päätöksistä voidaan arvioida hyvinkin nopeasti, kun taas joidenkin pää-

tösten lopputuloksen arvioiminen saattaa venyä pitkälle, jolloin lopputulos on pitkään kesken. Lopputuloksen yhteydessä tulee dokumentoida myös lopputuloksen perustelut, jotta sidosryhmillä on mahdollisuus nähdä, että minkä takia päätös on ollut onnistunut tai epäonnistunut [101].

+++	Tukee voimakkaasti päätöstä
++	Tukee kohtalaisesti päätöstä
+	Tukee hieman päätöstä
- - -	Vastustaa voimakkaasti päätöstä
- -	Vastustaa kohtalaisesti päätöstä
-	Vastustaa hieman päätöstä

Taulukko 5.3: Päätöstä tukevien ja vastustavien seikkojen pisteytys [100].

6 Tutkimuksen toteutus

Tämän Pro Gradu -tutkielman tutkimus toteutetaan luomalla konstruktio eli artefakti graafitietokantaa käyttävästä IoT-järjestelmästä, joka mittaa ja visualisoisi olosuhteita. Konstruktio suunnitellaan ja toteutetaan yhden suunnittelututkimuksen suunnittelusyklin aikana. Suunnittelusykli on esitelty alaluvussa 6.1. Toteutettavan IoT-järjestelmän kuvaus esitellään alaluvussa 6.2 ja arkkitehtuuri alaluvussa 6.3. Minkä lisäksi tässä luvussa esitellään IoT-järjestelmän havainnointikerros 6.4, verkkokerros 6.5, väliohjelmistokerros 6.6 ja sovelluskerros 6.7.

6.1 Suunnittelusykli

Suunnittelutieteen tutkimusprojektissa suunnitellaan ja tutkitaan iteroiden [42, sivu 76] [103, sivu 27, 30]. Suunnitteluosuus voidaan jakaa kolmeen osa-alueeseen, jotka ovat ongelman tutkiminen, ongelman ratkaisun suunnittelu ja ratkaisun validointi [103, sivu 27]. Näitä kolmea osa-aluetta kutsutaan suunnittelun sykliksi, koska suunnittelututkimusprojektissa on tarkoitus iteroida näitä kolmea osa-aluetta useamman kerran [103, sivu 27]. Suunnittelusykli on osa suurempaa kehityssykliä, jossa validoitu ratkaisu viedään oikeaan maailmaan käytettäväksi ja arvioitavaksi [103, sivu 27]. Ongelman tutkimisvaiheessa pohditaan mitä ongelmaa halutaan ratkaista ja miksi [42, sivu 76] [103, sivu 27]. Ratkaisun suunnitteluvaiheessa suunnitellaan artefakti, joka saattaa ratkaista ongelman [42, sivu 76] [103, sivu 27]. Validointivaiheessa arvioidaan, ratkaiseeko suunnitelmat ongelman, kun taas ratkaisun toteutusvaiheessa ratkaistaan ongelma yhdellä suunnitelluista artefakteista [42, sivu 76] [103, sivu 27]. Toteutuksen arviointivaiheessa arvoidaan kuinka onnistunut tehty toteutus on [42, sivu 76] [103, sivut 27]. Arviointivaihe saattaa käynnistää uudeen syklin.

Wieringa [103, sivu 28] kutsuu ratkaisua hoidoksi (suom. treatment), koska suunniteltu ratkaisu ei välttämättä ratkaise ongelmaa täysin tai suunnitellut artefaktit voivat jopa luoda uusia ongelmia. Tietojärjestelmien ja ohjelmistotuotannon kontekstissa artefaktit voivat esimerkiksi olla algoritmeja, metodeja, merkintöjä, tekniikoita tai konsepteja, joita käytetään kun suunnitellaan, kehitetään, ylläpidetään ja

käytetään tietojärjestelmiä ja sovelluksia [42, sivu 1] [103, sivu 28]. Artefaktit suunnitellaan ja dokumentoidaan määrittelyihin, jotka ovat dokumentaatiota suunnittelupäätöksistä [42, sivu 127] [103, sivu 29]. Suunnittelututkimusprojektin toteutusvaihe poikkeaa kehityssyklin toteutusvaiheesta muun muassa siinä, että tutkimusprojektin toteutusta ei tuotteisteta [103, sivu 30]. Ratkaisun validoinnin tavoitteena on ennustaa, kuinka artefakti vuorovaikuttaa kontekstinsa kanssa ilman, että toteutettua artefaktia tarkkailtaisiin oikean maailman kontekstissa [103, sivu 31] . Validoinnis-ta poiketen ratkaisun evaluoinnin tavoitteena on tutkia, miten toteutetut artefaktit toimivat oikean maailman kontekstissa [42, sivu 137] [103, sivu 31].

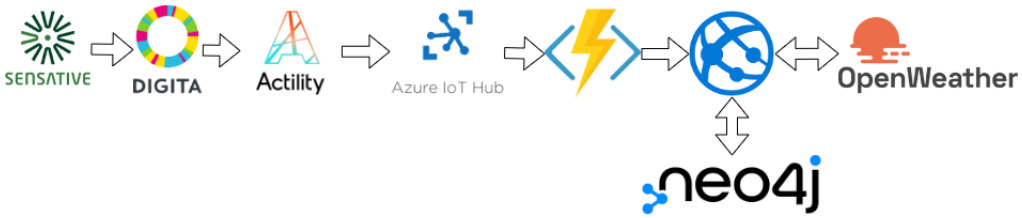
Ratkaisut tai hoidot on suunniteltu toimimaan oikeassa elämässä alkuperäisen ongelman kontekstissa. Kun ratkaisu on toteutettu, tarjoaa se tärkeää tietoa artefaktin ominaisuuksista ja itse ratkaisusta. Toteutuksen arvioinnissa tavoite on arvioida ratkaisun toteutusta sen jälkeen, kun sitä on sovellettu alkuperäisen ongelman kontekstissa [103, sivut 41] [42, sivu 137]. Tavoitteena on kuvailla, selittää ja arvioida toteutetun ratkaisun vaikutuksia, oli ne sitten positiivisia tai negatiivisia, suhteessa sidosryhmien tavoitteisiin [103, sivut 41-42] [42, sivut 137-138]. Wieringan [103, sivu 42] mukaan toteutuksen arvioinnin tieto-, ja sitä myötä, tutkimuskysymykset voidaan muotoilla kartoittavaksi, kuvailevaksi, arvioivaksi tai selittäväksi. Kartoittava kysymys pyrkii selvittämään, mitä vaikutuksia toteutetulla artefaktilla on. Kuvaileva kysymys taas haluaa tietää, että miten artefakti sai aikaan nämä vaikutukset ja millä mekanismeilla. Arvioiva kysymys kysyy, ovatko vaikutukset positiivisia ja/-tai negatiivisia suhteessa sidosryhmien tavoitteisiin. Selittävä kysymys pyrkii kertomaan, miksi vaikutukset ovat positiivisia tai negatiivisia [103, sivu 42].

6.2 IoT-Järjestelmän kuvaus

Tässä Pro Gradu -tutkielmassa toteuttiin IoT-järjestelmä yhden suunnittelusyklin aikana, jossa kolmella kuvan 6.4 Sensativen Strips MS +Comfort for LoRaWAN [91] sensorilla mitataan sisäilmasta suhteellista kosteutta, valoisuutta ja lämpötilaa. Sensorit on sijoitettu kolmeen eri asuntoon kolmelle eri paikkakunnalle: Helsinkiin, Tampereelle ja Kokkolaan. Sensoreiden keräämät mittausarvot välitetään Digitan LoRa-verkossa [30] verkossa Actilityn Thingpark [4] LoRa-sovellukseen, josta viestit välitetään Azuren IoT Hubiin [56] JavaScript Object Notation (JSON) muodossa. Viestin saapuminen IoT Hubiin laukaisee palvelimettoman Azure Functionin [55], joka karsii viestistä tarpeettomat tiedot ja lähettää kuvan 6.9 mukaisen

viestin .Net 7 [59] ohjelmistokehyksen päälle rakennetulle verkkosovellukselle, joka koostuu palvelin- ja käyttöliittymäsovelluksista. Verkkosovellus suoritetaan Azuren App Servicessä [54].

Palvelinsovellus muodostaa vastaanotetusta viestistä tietokantakutsut, joilla sensorin tiedot ja mittausarvot tallennetaan Neo4J AuraDB graafitietokantaan, joka on Neo4J:n tarjoama Software as a service (SaaS) graafitietokanta [71]. Samassa yhteydessä palvelinsovellus hakee OpenWeather-palvelusta [78] sen hetkiset säätiedot, joista tallennetaan tietokantaan ulkolämpötila, ilman kosteus ja paine. Käyttöliittymäsovelluksessa näytetään viivadiagrammeilla tietokantaan tallennettua tietoa. Tietokannasta haettavaa tietoa voi rajata muun muassa päivämäärillä ja mittauksen suureella. Kuvassa 6.1 on lueteltu edellä mainitut komponentit, joihin on perehdyt-ty tarkemmin tämän luvun alaluvuissa. Kuvan nuolet kuvaavat sitä, mihin suun-taan tieto kulkee IoT-järjestelmässä komponenttien välillä. Komponenteista Actility Thingpark, Azure IoT Hub, Azure Functions, verkkosovellus ja Neo4J AuraDb palvelut on toteutettu pilvipalveluina tai pilvipalveluihin.

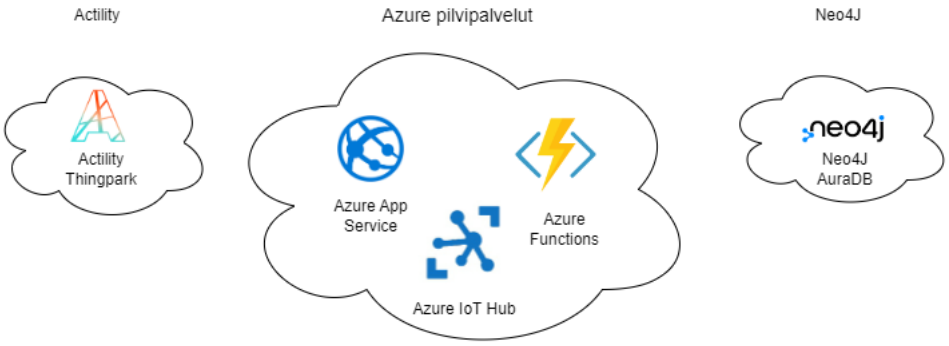


Kuva 6.1: IoT-järjestelmän korkean tason komponentit

Kuvassa 6.2 on esitetty käytetyt pilvipalvelut. Iso osa palveluista on Microsoft Azuren pilvipalveluita, koska niitä oli aiemmin käytetty [46] projektissa. Toteutetun IoT-järjestelmän Azuren palveluiden käyttö on käytännössä ilmaista opiskelijoille, koska Microsoft tarjoaa sadan dollarin edestä krediittejä Azureen, jonka lisäksi opiskelijoille on tarjolla paljon ilmaisia palveluita [57]. Opiskelijaetujen saaminen vaatii opiskelijatilin tekemistä käyttäen yliopiston sähköpostitunnuksia [57]. Opiskelijatili ei vaadi luottokortin käyttämistä toisin kuin Azuren normaalitili [57].

6.3 IoT-järjestelmän arkkitehtuuri

Edellä 6.2 kuvatun IoT-järjestelmän arkkitehtuuri, joka on esitetty kuvassa 6.3, perustuu luvussa 2.2 esiteltyihin IoT-järjestelmien yleisempiin kolmi- 2.2 ja viisiker-



Kuva 6.2: IoT-järjestelmän pilvipalvelut

rosarkkitehtuureihin 2.3. Sensativen Strips MS +Comfort for LoRaWAN sensorit muodostavat järjestelmän havannointikerroksen mitaten sisäilmasta suhteellista ilman kosteutta, lämpötilaa ja valoisuutta. Lisäksi sensori lähettää mittaamansa tiedon LoRa-verkossa lähimmälle LoRA-yhdyskäytävälle. Verkkokerrosta ei ole toteutetussa järjestelmässä jaettu erikseen verkkoyhteys- ja verkkokerrokseen, koska tiedonsiirto sensorilta väliohjelmistokerrokselle on hoidettu kokonaisuudessaan Digitan hallinnoimassa LoRa-verkossa. Verkkokerroksesta on kerrottu tarkemmin alaluvussa 6.5.

Toteutetussa järjestelmässä sovelluskerros on jaettu kahtia esitellyn viisikerrosarkkitehtuurin 2.3 mukaisesti väliohjelmistokerrokseen ja sovelluskerrokseen. Väliohjelmistokerros koostuu kolmesta eri palvelusta. Digita käyttää Actilityn Thingparkia LoRa-verkkonsa keskitettynä ratkaisuna, jolla hallitaan LoRa-verkkoon lisättäviä sensoreita [1]. Thingparkissa sensorin mittausdataa jalostetaan muun muassa lisäämällä siihen sensorin omistajan nimitieto, joka on palveluun erikseen määritelty sensorin yksilöllisen tunnisteen perusteella [4]. LoRa-yhdyskäytävän, joka ensimmäisenä vastaanottaa sensorin lähettämän viestin, sijainti liitetään sensorin tuottamaan tietoon. Thingparkista tieto välitetään Azuren IoT Hubiin, jolla voidaan hallinnoida sensoreita ja niiden välittämää dataa useista eri lähteistä [56]. Kolmas väliohjelmistokerroksen palvelu on Azure Functions, jonka tehtävänä on käsitellä IoT Hubiin tullutta sensoridataa ja välittää se eteenpäin verkkosovellukselle. Väliohjelmistokerroksen toteutusta on kuvailtu tarkemmin alaluvussa 6.6.

IoT-järjestelmän ylin kerros, sovelluskerros, koostuu verkkosovelluksesta ja graafitietokannasta. Verkkosovellus on kaksiosainen sisältäen sekä palvelinkoodin että käyttöliittymäkoodin. Verkkosovellus suoritetaan Azuren App Service palvelussa.

Kuva 6.3: Toteutetun IoT-järjestelmän nelikerrosarkkitehtuuri

Palvelinkoodin tehtävänä on vastaanottaa sensoridataa ja tallentaa se graafitietokantaan. Tämä osa palvelinkoodia voidaan myös luokitella väliohjelmistokerroksen tehtäviin, kuten alaluvussa 2.5 kuvailtiin. Lisäksi palvelinkoodi tekee käyttöliittymältä lähetettyjen pyyntöjen perusteella tietokantakutsuja graafitietokantaan. Graafitietokanta Neo4J AurDB on toteutetun IoT-järjestelmän ainoa tietokantajärjestelmä. IoT-järjestelmän tietokanta voidaan sijoittaa IoT-järjestelmän arkkitehtuurikuvauksessa joko väliohjelmistokerrokselle tai sovelluskerrokselle. Toteutetusta sovelluskerroksesta on kerrottu tarkemmin alaluvussa 6.7.

6.4 Havainnointikerros

Havainnointikerroksen tehtävänä IoT-järjestelmässä on havainnoida ympäristöä ja kerätä reaaliaikaista dataa eri laitteiden avulla. Toteutetussa IoT-järjestelmässä havainnointikerros muodostui kolmesta hyvin tyypillisestä havainnointikerroksen laitteesta eli vähävirtaisesta langattomasta sensorista, jotka mittaavat ympäristöstään ilman kosteutta, lämpötilaa ja valoisuutta. IoT-järjestelmän havainnointikerroksen sensoriksi valikoitui kuvan 6.4 Sensativen Strips MS +Comfort for LoraWAN, jolla voi mitata sisäilman lämpötilan lisäksi suhteellista ilman kosteutta ja valoisuutta [91]. Kyseinen sensori valittiin sen vuoksi, että se on ennestään tuttu ja saatavilla Kokkolan yliopistokeskuksen Ruoste-projektiin [46] osallistumisen myötä. Sensoreita on sijoitettuna kolmeen kaupunkiin, joka tekee IoT-järjestelmän tekemisestä monipuolisempaa ja mielekkäämpää. Eri paikkakunnille sijoitetut sensorit havainnollistavat IoT-järjestelmien paikkasidonnaisuutta ja hajautettua luonnetta. Lisäksi kuten alaluvussa 4.1 todettiin, graafitietokannat eivät sovellu hyvin IoT-järjestelmän tietokannaksi, mikäli järjestelmässä on vain yksi dataa keräävä sensori.

Kuva 6.4: Sensativen Strips MS +Comfort for LoRaWAN sensori

Sensorin käyttöönotto on erittäin yksinkertaista Sensativen [86] ohjeiden avul-

la. Sensativella on sensoreilleen sovellus [89], jolla voi konfiguroida sensorin lähettämän tietojen tiheyttä ja sisältöä [88]. Sovelluksen tuottamat asetukset lähetetään Lora-verkon palvelimelle Downlink-viestinä porttiin 11, joka on toteutetun IoT-järjestelmän tapauksessa Actilityn Thingpark [88]. Toteutetussa IoT-järjestelmässä sensorit konfiguroitiin lähettämään puolen tunnin välein ilman suhteellisen kosteuden, sisäilman lämpötilan keskiarvon ja valoisuuden mittausarvot.

Sensori lähettää mittausarvot heksadesimaalisena hyötykuormana, jonka purkamiseen Sensative tarjoaa avoimeen lähdekoodiin perustuvan sovelluksen [90], jol-la hyötykuorman voi purkaa. Toteutetussa IoT-järjestelmässä edellä mainitulle sovellukselle ei ollut tarvetta, koska Actilityn Thingpark muunsi hyötykuorman jo JSON-muotoon. Ruoste-projektissa [46] kyseisen sovelluksen avointa lähdekoodia hyödynnettiin onnistuneesti hyötykuorman purkamisessa. Sensative [91] lupailee sensorille jopa kymmenen vuoden toiminta-aikaa. Sensorin paristo on integroitu eikä sitä voi vaihtaa tai ladata. Sensative tarjoaa LoRa-verkon sensoreilleen laskurin [87], jolla voi arvioida sensorin pariston kestoa.

6.5 Verkkokerros

Verkkokerroksen tarkoituksena on siirtää havainnointikerroksella kerätty tieto eteenpäin väliohjelmisto- ja sovelluskerrokselle. Verkkokerros voidaan jakaa verkkoyhteys- ja verkkokerrokseen. Verkkoyhteyskerrosta kutsutaan myös yhdyskäytäväkerrokseksi ja verkkokerrosta lähetyskerrokseksi. Toteutetussa IoT-järjestelmässä koko verkkokerros koostuu Digitan operoimasta LoRaWAN-verkosta.

Toteutetun IoT-järjestelmän havainnointikerroksen laitteiksi valikoitui edellises-sä osiossa esitellyt Sensativen LoRaWAN-verkossa toimivat sensorit. Sensoreiden valinta lukitsi samalla myös käytettävän lähetystekniikan, sillä valitut sensorit toimivat ainoastaan LoRaWAN-verkossa. Suomessa LoRaWAN-verkkoa operoi Digi-ta, jonka verkko kattaa lähes koko Suomen. LoRaWAN soveltuu erinomaisesti IoT-järjestelmän tarpeisiin. LoRaWAN-verkko kykenee kahdensuuntaiseen langattomaan tiedonsiirtoon havainnointikerroksen laitteilta reitittimille ja reitittimiltä laitteille. Toteutetussa IoT-järjestelmässä tiedonsiirto oli käytännössä vain sensoreilta verkkoon päin, muutamia sensoreille lähetettyjä konfiguraatioviestejä lukuunottamatta. Verkko on suunniteltu vähäisten tietomäärien lähettämiseen, sillä tyypillinen tiedonsiirtonopeus verkossa on 0,3 - 0,5 Kb/s. [30]

6.6 Väliohjelmistokerros

Toteutetun IoT-järjestelmän väliohjelmisto kerros koostuu kolmesta eri palvelusta, jotka on esitelty tässä aliluvussa. Actilityn Thingpark 6.6.1 on vastuussa järjestelmän havainnointikerroksen laitteiden hallinnasta, jonka lisäksi se välittää ja rikastaa sensoreiden mittaamaa tietoa eteenpäin. Azure IoT Hubin 6.6.2 tehtävänä on vastaanottaa Thingparkista lähetetyt sensoreiden viestit keskitetysti, jotta niitä voidaan hyödyntää sovelluskerroksella. Azure Functions karsii IoT Hubiin saapunees-ta viestistä järjestelmän kannalta tarpeettoman tiedon pois ja välittää sen eteenpäin sovelluskerrokselle.

6.6.1 Actility Thingpark

Actility Thingpark on Digitan keskitetty ratkaisu LoRa-verkon laitteiden hallintaan, jonka johdosta se valittiin myös toteutettuun IoT-järjestelmään. IoT-järjestelmän arkkitehtuurissa havainnointikerroksen laitteiden hallinta sijoittuu väliohjelmistokerrokselle. Toteutetussa IoT-järjestelmässä Kokkolan yliopistokeskus vastasi sensoreiden lisäämisestä Thingpark sovellukseen. Sensoreiden lisääminen LoRa-verkkoon tehtiin jo Ruoste-projektin [46] yhteydessä. Lisäksi Thingparkiin lisättiin tunnistetiedot Azure IoT Hubista, jotta Thingpark osaa lähettää sensoreiden viestit eteenpäin oikeaan palveluun [2]. Thingpark vastaanottaa Strips MS +Comfort for LoRaWAN sensorin lähettämän mittausdatan heksadesimaalisena hyötykuormana. Thingpark jalostaa vastaanottaman viestin sensorin ja LoRa-verkon tiedoilla, jonka lisäksi se purkaa mittausdatan JSON-muotoon [5].

Kuvassa 6.5 on esitetty Thingparkin lähettämä sensoridata Azure IoT Hubiin yhden mittauksen osalta. Kuvasta on havaittavissa sensorin alkuperäinen hyötykuorma sekä purettuna ja ymmärrettävänä payload-objektina että alkuperäisenä heksadesimaalisena lähetyksenä payload_hex-objektina. Payload-objektista voimme havaita, että mitattavien suureiden osalta on kerrottu sekä mitattu arvo value-kentässä että yksikkö unit-kentässä. Sensoridatan Time-kenttä kertoo mittauksen ajankohdan, kun taas DevEUI on sensorin yksilöivä tunniste. CustomerId on asiakkaan yksilöllinen tunniste, joka tässä tapauksessa viittaa Kokkolan yliopistokeskukseen. CustomerData-objektissa on name-kenttä, joka kertoo kyseisen sensorin omistajan nimen. CustomerData-objektissa on tags- ja doms-kentät, joihin voi halutessaan määritellä lisätunnisteita Thingparkin kautta. LrrLAT ja LrrLON ovat yhdyskäytävän leveys- ja pituusastekoordinaatteja, jotka Thingpark on liittänyt viestiin.[5]

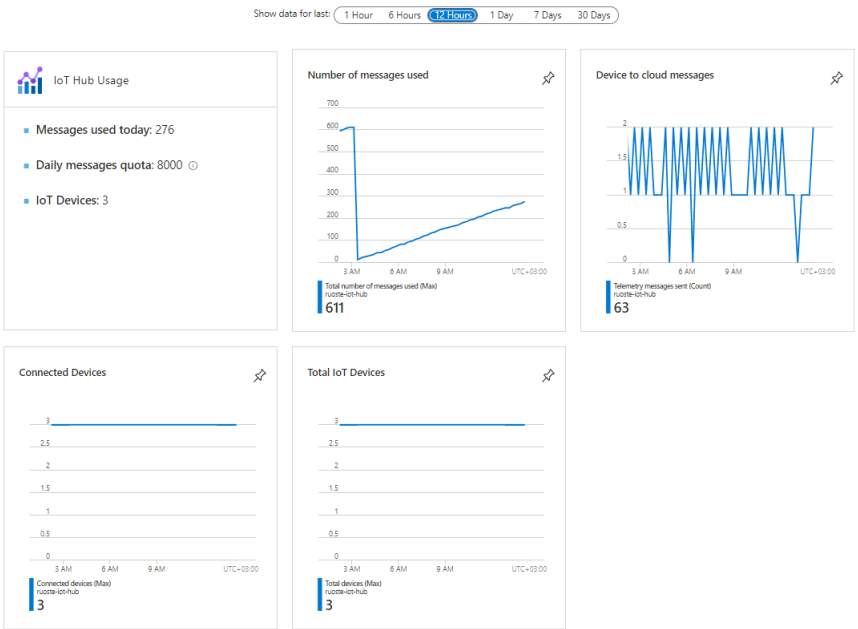
```
{
    "Time": "2023-04-15T10:17:32.746+00:00",
    "DevEUI": "70B3D52C0001962A",
    "FPort": "1",
    "FCntUp": "23340",
    "ADRbit": "1",
    "MType": "2",
    "FCntDn": "2243",
    "payload_hex": "ffff0400ee0629070066",
    "mic_hex": "350b771a",
    "Lrcid": "00000201",
    "LrrRSSI": "-95.000000",
    "LrrSNR": "-13.000000",
    "LrrESP": "-108.212387",
    "SpFact": "11",
    "SubBand": "G1",
    "Channel": "LC1",
    "Lrrid": "FF019079",
    "Late": "0",
    "LrrLAT": "60.183723",
    "LrrLON": "24.971588",
    "Lrrs": {
        "Lrr": "[Object]"
    },
    "DevLrrCnt": "1",
    "CustomerID": "100006325",
    "CustomerData": {
        "loc": null,
        "alr": {
            "pro": "SENS/STRIPS",
            "ver": "1"
        },
        "tags": [],
        "doms": [],
        "name": "Sensative Strip #30 Ilkka Jokela"
    },
    "ModelCfg": "*:sensorstrip",
    "DriverCfg": {
        "mod": {
            "pId": "sensativ",
            "mId": "strips",
            "ver": "1"
        },
        "app": {
            "pId": "sensativ",
            "mId": "strips",
            "ver": "1"
        },
        "id": "actility:sensative-strips:1"
    },
    "DevAddr": "E002FDD2",
    "TxPower": "16.000000",
    "NbTrans": "1",
    "Frequency": "868.1",
    "DynamicClass": "A",
    "payload": {
        "historyStart": 65535,
        "AverageTempReport": {
            "value": 23.8,
            "unit": "C"
        },
        "HumidityReport": {
            "value": 20.5,
            "unit": "%"
        },
        "LuxReport": {
            "value": 102,
            "unit": "Lux"
        }
    },
    "downlinkUrl": "https://digita.thingpark.com/iot-flow/downlinkMessages/3d596f09-3955-47cf-820e-61a9d5a1242b"
}
```


Kuva 6.5: Sensorin mittausdata Thingparkista IoT Hubiin JSON-muotoisena

6.6.2 Azure IoT Hub

Toteutetussa IoT-järjestelmässä väliohjelmistokerrokseen sijoittuvan Azure IoT Hubin tehtävänä on vastaanottaa Thingparkin lähettämä sensoridata 6.5, jotta sitä voidaan hyödyntää sovelluskerroksen sovellusten toimesta. Azure IoT Hub valikoitui toteutettuun IoT-järjestelmään muun muassa sen vuoksi, että siitä oli aiempaa kokemusta sensoriverkkoprojektin [46] myötä ja sensorit olivat valmiiksi siihen liitettyinä. Microsoftin kilpailija Amazon Web Servicellä (AWS) on IoT Hubia vastaava palvelu IoT Core [9]. Azure IoT Hubin luominen onnistuu helposti esimerkiksi Actilityn tarjoamien ohjeiden [2] avulla. Samat ohjeet neuvovat myös sen, että miten luotu IoT Hub liitetään Thingparkkiin. Actility tarjoaa vastaavanlaiset ohjeet myös AWS:n IoT Corelle [3]. Microsoftin [64] omissa ohjeissa on selkesti kuvattu, että miten sensori liitetään ja rekisteröidään IoT Hubiin.

IoT Hubista on helposti nähtävissä tietoa sensoreiden lähettämistä viesteistä sen jälkeen, kun ne on onnistuneesti rekisteröity IoT Hubiin. Kuvassa 6.6 on esimerkki IoT Hubin monitorointilastoista toteutetun IoT-järjestelmän osalta. Kuvasta on havaittavissa se, että viimeisen kahdentoista tunnin aikana IoT Hubiin on ollut yhteydessä yhteensä kolme eri laitetta. Monitoroinnista näkee myös sensoreiden lähettämien ja IoT Hubin vastaanottamien viestien lukumäärän.



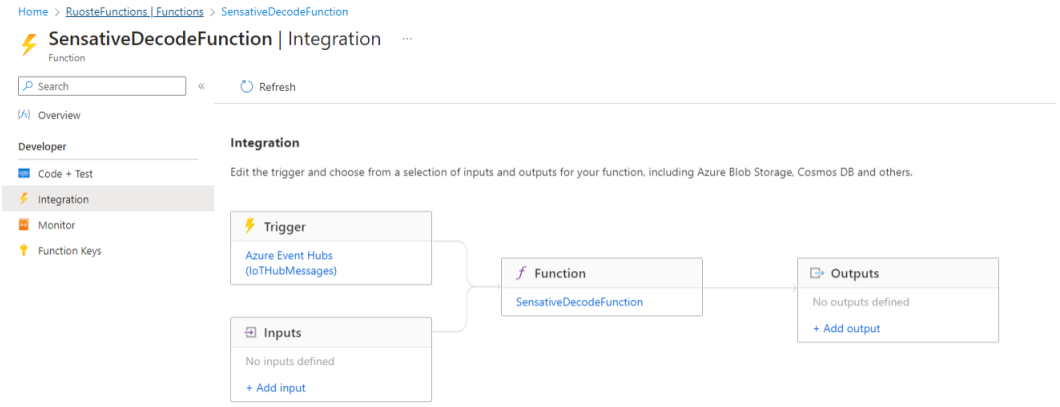
Kuva 6.6: Azure IoT Hubin monitorointitilastoja

6.6.3 Azure Functions

Azure Functions on Microsoft Azuren palvelu, jolla voidaan toteuttaa esimerkik-si palvelimettomia API-rajapintoja, verkkosovelluksia ja datan käsittely putkia [55]. Palvelimettomuus tarkoittaa käytännössä sitä, että Azure Functionsiin luodut ohjelmat eivät ole jatkuvassa ajossa, vaan ne suoritetaan tarvittaessa [55]. Azure Functionsin avulla luodut ohjelmat ovatkin tapahtumalähtöisiä ja Microsoft tarjoaakin erilaisia laukaisijoita, joilla ohjelmat voidaan käynnistää aina tarvittaessa [55]. Azure Functionsiin luodun ohjelman tarkoitus on toteutetussa järjestelmässä välittää IoT Hubiin saapunut sensorin viesti palvelimelle. Viestin välittämisen lisäksi ohjelma karsii kuvan 6.5 mukaisesta viestistä toteutetulle järjestelmälle tarpeettomat tiedot. Edellä mainitut tehtävät ovat hyvin tyyppillisiä väliohjelmistokerroksen tehtäviä.

Kuvassa 6.7 on kaaviokuva toteutetun järjestelmän Azure Functionsista, josta näkyy, että laukaisijana toimii IoT Hubiin saapuva viesti. Ohjelma on palvelimeton, joten se käynnistyy vain silloin, kun IoT Hubiin tulee uusi viesti Thingparkista. Palvelimettomille ohjelmille on tyypillistä, että niistä maksetaan ajokertojen mukaan. Azuren Functionseja saa ajaa miljoona kertaa kuukaudessa maksutta [58]. Azure Functions valikoitui IoT-järjestelmään, koska siinä on suoraan tuki Azuren IoT Hubille ohjelman laukaisijana. Lisäksi palveliton ratkaisu on viestin välittämiseen ja datan karsimiseen hyvä vaihtoehto, koska ohjelmaa ei tarvita kuin aina hetkellises-ti uuden viestin saapuessa. AWS:n palveliton AWS Lambda on vaihtoehto Azure Functionsille, jos käytössä on Azure IoT Hubin sijasta AWS:n IoT Core. IoT Coren viestiä voi samalla tavalla käyttää laukaisijana AWS Lambdalle [10].

Kuvassa 6.9 on esitetty JSON-muotoinen viesti, joka lähetetään järjestelmän palvelinsovellukselle. Viestissä palvelinsovellukselle menee mittausdatan lisäksi aikaleima, sensorin yksilöivä tunniste, Thingparkin asiakastunniste ja viestin lähettäneen yhdyskäytävän geokoordinaatit [5]. Karsittu viesti on huomattavasti lyhyempi kuin alkuperäinen Thingparkilta Azure IoT hubiin saapuva viesti. Karsittua viestiä on helpompi käsitellä palvelimen ohjelmakoodissa, koska käsiteltäviä ja varmennettavia muuttujia on vähemmän, joka vähentää mahdollisten virheiden määrää. Viestin karsimisen ja lähettämisen toteutettava JavaScript-koodi on esitetty kuvassa 6.8. JavaScriptin lisäksi Azure Functionseja voi ohjelmoida muun muassa C#, Pythonilla ja Javalla [65].



Kuva 6.7: Azure Functionsin laukaisijana toimii IoT Hubiin saapuva viesti

6.7 Sovelluskerros

Toteutetun IoT-järjestelmän sovelluskerros koostuu verkkosovelluksesta ja graafitietokannasta. Toteutettu verkkosovellus sisältää sekä palvelin- että käyttöliittymäkoodin. Azure App Servicessä ajettavasta verkkosovelluksesta on kerrottu tarkemmin osiossa 6.7.1. IoT-järjestelmän tietokantana toimii Neo4J AuraDB graafitietokanta, jota on kuvailtu tarkemmin osiossa 6.7.2. Verkkosovelluksen palvelinkoo-di ja graafitietokanta voidaan sijoittaa arkkitehtuurikuvauksessa väliohjelmistokerrokselle sovelluskerroksen sijaan. Graafitietokannan sijoittamista sovelluskerrokselle arkkitehtuurikuvauksessa kuitenkin tukee se, että tiedonhallinnan lisäksi graafitietokanta tarjoaa mahdollisuuden visualisoida dataa päätöksenteon tueksi. Verkkosovellus on esitelty alaosiossa 6.7.1 ja Neo4J AuraDB graafitietokanta alaosiossa 6.7.2.

6.7.1 Verkkosovellus

Toteutetun IoT-järjestelmän verkkosovelluksen tehtävä on kaksiosainen. Palvelinkoodin tehtävänä on vastaanottaa Azure Functionsista lähetetty sensoridata ja tallentaa se graafitietokantaan. Käyttöliittymäkoodin tehtävänä on visualisoida sensoridataa viivadiagrammien avulla. Käyttöliittymäkoodista lähetetään palvelinkoodille Http-protokollan mukainen GET-kysely, jonka perusteella palvelinkoodi tuottaa tietokantakyselyn ja palauttaa tuloksen käyttöliittymälle. Verkkosovelluksen koodi ajetaan Azuren App Servicessä, jonne koodi tuodaan Microsoftin [61] ohjei-

```
const options = {
    hostname: 'sensativewebapp.azurewebsites.net',
    path: '/api/sensativecomfort',
    method: 'POST',
    headers: {
    'Content-Type': 'application/json',
    'X-Api-Key': apikey
    }
};
context.log("JSON:", result.DevEUI_uplink);
const sensativeComfortRequest = {
    "Time": result.DevEUI_uplink.Time,
    "DevEUI": result.DevEUI_uplink.DevEUI,
    "LrrLAT": result.DevEUI_uplink.LrrLAT,
    "LrrLON": result.DevEUI_uplink.LrrLON,
    "CustomerID": result.DevEUI_uplink.CustomerID,
    "CustomerData": result.DevEUI_uplink.CustomerData,
    "payload": result.DevEUI_uplink.payload
}
context.log("Sensative comfort request", sensativeComfortRequest);
const postData = JSON.stringify(sensativeComfortRequest);
const request = https.request(options, response => {
    let data = '';
    response.on('data', chunk => {
    data += chunk;
    });
    response.on('end', () => {
    context.res = {
        status: response.statusCode,
        body: data
    };
    context.done();
    });
});

request.on('error', error => {
    context.log(error);
    context.res = {
    status: 500,
    body: error.message
    };
    context.log("Error sending data to app", error);
    context.done();
});

request.write(postData);
request.end();

context.done();
```

Kuva 6.8: Azure Functionsin koodi, joka karsii ja lähettää viestin palvelimelle

Kuva 6.9: Palvelinsovellukselle lähetettävä pyyntö JSON muodossa

den mukaisesti automaattisesti versionhallintatyökalusta Githubista koodin päivittyessä.

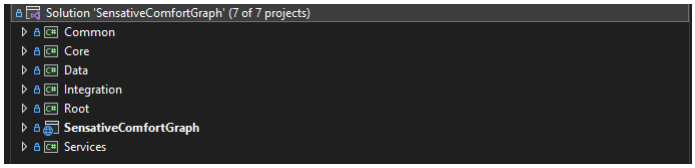
Azure App Service on Platform as a service-palvelu (PaaS), joka mahdollistaa koodin ajamisen ilman, että tarvitsee huolehtia palvelimista tai virtuaalikoneista [54]. AWS:llä on vastaava palvelu, joka on nimeltään AWS App Runner [8]. Toteutetussa IoT-järjestelmässä App Service valittiin lähdekoodin ajoalustaksi sen vuoksi, että se tukee .Net 7 ohjelmistokehyksellä toteutettuja ohjelmistoja. Opiskelijoil-le App Servicen perustaso on maksuton. App Service tukee .Netin lisäksi esimer-kiksi Node.js:ää, Javaa ja Pythonia [54]. Palvelinkoodi on ohjelmoitu käyttäen C#-ohjelmistokieltä, joka on F# ja Visual Basicin ohella ohjelmistokielet, joita .Net 7 ohjelmistokehys tukee [63].

Käyttöliittymäkoodi on ohjelmoitu JavaScriptillä, jonka ohella on käytössä ReactJs-ohjelmistokirjasto, joka helpottaa käyttöliittymän rakentamista komponentteja hyödyntäen [53]. Vaihtoehtoisia suosittuja ohjelmistokirjastoja ovat esimerkiksi Angular, Vue ja Svelte [7]. ASP.NET Core tarjoaa valmiin sapluunan, jossa on paketoituna .Net palvelinkoodin ja käyttöliittymäkoodin pohjat. Sapluuna on saatavilla esimerkiksi React, Angular ja Vue ohjelmistokirjastoille [66].

Verkkosovelluksen perustuu kerrosarkkitehtuuriin, joka on yksi yleisistä verkkosovellusarkkitehtuureista [60]. Kerrosarkkitehtuurin idea on jakaa ohjelmointilo-

giikka eri kerroksiin perustuen niiden tehtäviin eli vastuualueisiin [60] [97]. Tämä mahdollistaa sen, että yhteiseen käyttöön tarkoitettu koodi on käytettävissä sovelluksen eri osa-alueilla [60]. Jonkin kerroksen koodin muuttuessa, muutos koskettaa vain niitä kerroksia, jotka hyödyntävät muuttunutta kerrosta [60]. Teoriassa tämän pitäisi esimerkiksi mahdollistaa sovelluksen tietokantajärjestelmän vaihdon niin, että ainoastaan tietokantayhteydestä vastaava kerros muuttuu. Koodin jakaminen eri projekteihin kerroksittain mahdollistaa teoriassa niiden suorittamisen eri ympäristöissä [97].

Kuvassa 6.10 on esitetty toteutetun verkkosovelluksen eri kerrokset. Common-kerrokseen on sijoitettu kaikkialla ohjelmakoodissa käytettävissä olevat apumetodit, joilla voi esimerkiksi validoida muuttujien arvoja. Core-kerroksessa on luokkien mallit, jotka ovat myös hyödynnettävissä eri puolella sovellusta. Integration-kerros vastaanottaa Azure Functionsista tulevan sensoridatan, jonka jälkeen se välittää sen Services-kerrokselle, joka vastaa sovelluksen logiikasta eli luokkien muodostamisesta ja tiedon jäsentelystä. Tiedon jäsentely on tärkeää, jotta sen visualisointi on helpompaa käyttöliittymäkerroksella. Services-kerros kutsuu Data-kerrosta, joka on vastuussa tietokantayhteydestä ja kyselyistä. Services-kerroksen tehtäviin kuuluu säätietojen haku OpenWeatherMapin CurrentWeather API-rajapinnasta sensoritietojen mukana tulleita geokoordinaatteja hyödyntäen [77]. SensativeComfortGraph vastaa sovelluksen visualisoinnista eli loppukäyttäjälle näkyvästä osuudesta sisältäen esimerkiksi käyttöliittymäkoodin. Root-projekti on vastuussa projektin riippuvuuksien hallinnasta.



Kuva 6.10: Verkkosovelluksen projektit

Sovelluksen Data-kerroksella tietokantayhteyden ja Cypher-kyselyiden muodostamiseen käytetään Neo4J-yhteisön ylläpitämää avoimen lähdekoodin Neo4JClient-ohjelmistokomponenttia virallisen The Neo4j .NET driver [76] sijasta, koska se on latausmäärien perusteella huomattavasti suositumpi [21]. Neo4J tarjoaa ilmaisen kurssin viralliselle ohjelmistokomponentilleen ja sen käytölle .Net ohjelmistokehyksessä [69], jonka lisäksi sen sivuilla on kattavat ohjeet tietokantayhteyden muodos-

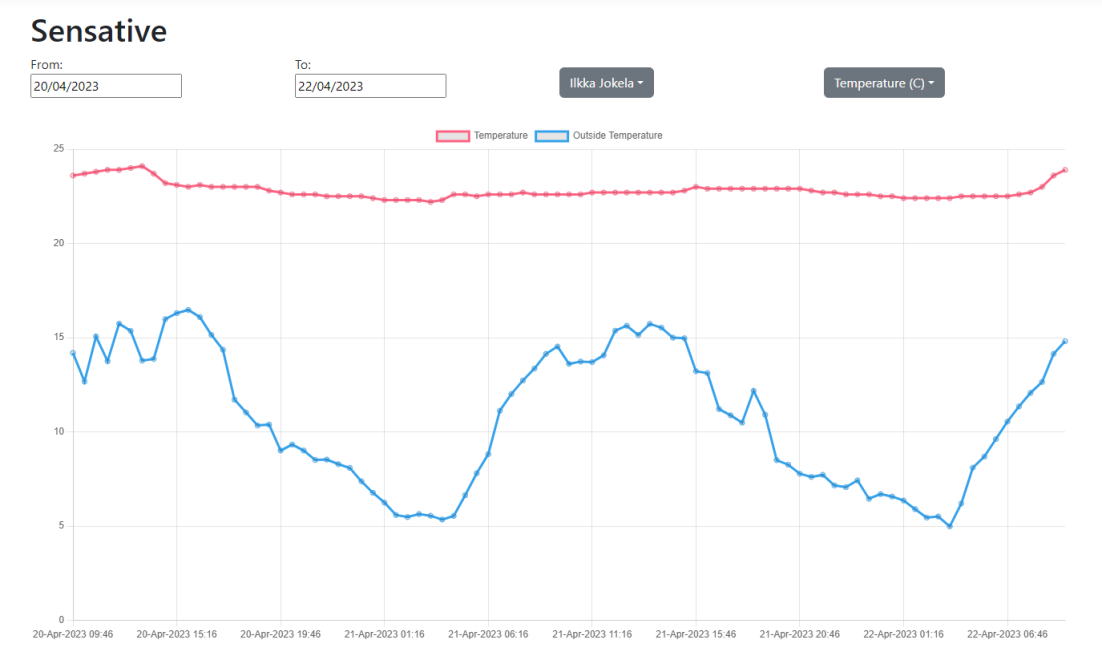
tamiseen [76]. Neo4JClientin käytölle ohjeistus on vähäisempää ja heidän oman dokumentaationsa mukaan vanhentunutta [21].

Kuvassa 6.11 on esitetty Neo4JClientia hyödyntävä toteutetun verkkosovelluksen koodi, jolla tallennetaan yksittäisen mittausarvon data Neo4J-tietokantaan. Kysely on rakennettu siten, että ensiksi tietokannasta etsitään laite annetulla nimellä, joka sijaitsee annetussa sijainnissa. Tämä tehdään yhdistämällä Match-avainsana Where-avainsanalla kyselyssä esitettyihin laitetta ja sijainteja koskeviin ehtoihin. Create-avainsana luo tallennettavan noodin annetuin parametrein mikäli edellä mainittu laitesijaintiyhdistelmä löytyy, jonka lisäksi mittauksen suorittaneen laitteen noodille d ja mittausarvon noodille m asetetaan suhde MEASURES. MEASURES-suhde kertoo, että annettu laite on mitannut esimerkin mittausarvon. Neo4JClientin tarjoaman WithParams-metodin avulla voidaan määritellä kyselyssä käytettävät parametrit. Esimerkissä käytettäviä paremetreja ovat mittausarvon nimi, aika, arvo ja yksikkö.

```
30          await _graphClient.Cypher
31                  .Match("(d:Device)-[:IS_LOCATED]->(l:Location)")
32                  .Where((Device d) => d.DevEUI == deviceName)
33                  .AndWhere((Location l) => l.Latitude == location.Latitude && l.Longitude == location.Longitude)
34                  .Create("(d)-[:MEASURES]->(m:Measurement {Name: $name, Time: datetime($time), Value: $value, Unit: $unit})")
35                  .WithParams(new { name = measurement.Name, time = measurement.Time.UtcDateTime, value = measurement.Value, unit = measurement.Unit })
36                  .ExecuteWithoutResultsAsync();
```

Kuva 6.11: Neo4JClientilla luotu Cypher-kysely, joka tallentaa mittausarvon tietokantaan

Sovelluksen esitys- eli visualisointikerroksella hyödynnetään react-chartjs-2 -komponenttikirjastoa sensorien mittausarvojen visualisointiin viivadiagrammein. React-chartjs-2 tarjoaa React-ohjelmistokirjastolle omat komponentit, jotka perustuvat
Chart.js-kirjastoon [83]. React:lle on tarjolla paljon eri komponenttikirjastoja graafien piirtämiseen, joista suosituin on Recharts ja toiseksi suosituin on React-chartjs-2 perustuen viikottaisiin latausmääriin [92]. React-chartjs-2 valikoitui Rechartsin sijaan toteutettuun sovellukseen, koska sillä oli parempi tuki useamman viivadiagrammin piirtämiseen samaan graafiin toteutetun IoT-järjestelmän tietomalli huomioon ottaen. Kuvassa 6.12 on käyttöliittymällä visualisoituna React-chartjs-2 avulla kahden päivän ajalta yhden sensorin puolen tunnin välein mittaamat sisälämpötilat ja samanaikaiset ulkolämpötilat, jotka on haettu tietokantaan OpenWeatherMap-palvelusta. Käyttöliittymällä voi muuttaa aikaväliä, sensorin omistajaa ja haettavaa suuretta.

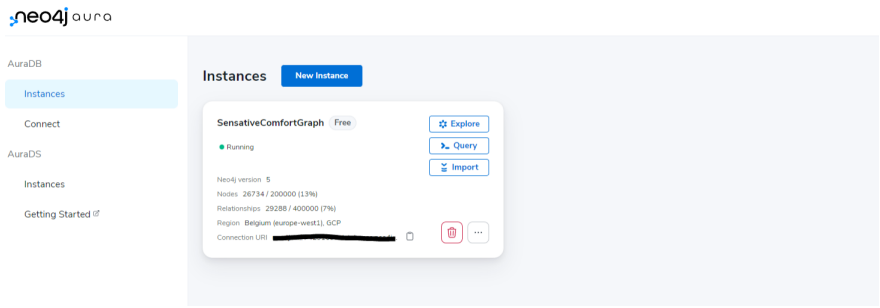


Kuva 6.12: Mittausdataa viivadiagrammein visualisoituna käyttöliittymällä

6.7.2 Neo4J AuraDB graafitietokanta

Toteutetun IoT-järjestelmän tietokantajärjestelmäksi valikoitui Neo4J AuraDB, joka on Neo4J graafitietokanta PaaS-palveluna [72]. Toteutetun IoT-järjestelmän käytössä oleva tilaus on AuraDB Free, jonka käyttö on ilmaista, mutta jossa on rajoitettuna noodien lukumäärä kahteensataantuhanteen ja suhteiden lukumäärä neljäänsataantuhanteen [72]. Graafitietokannan luominen AuraDB:hen on erittäin yksinkertaista Neo4J:n ohjeiden mukaan [70]. Kuvassa 6.13 on verkkosivu, jolle päädytään kun tietokanta on luotu, ja jonka kautta luotua tietokantaa voi hallita. Connect-välilehdeltä löytyvät ohjeet tietokantaan yhdistämiselle eri ohjelmointikielille kuten C#:lle, Pythonille, JavaScriptille ja Go:lle. Ohjeet sisältävät jo valmiiksi luodun tietokannan osoitetiedot, joten niiden käyttäminen on helppoa. Toteutetussa järjestelmässä yhdistäminen tehtiin käyttäen Neo4JClientia [21], jonka ohjeistus hieman poikkeaa virallisista ajureista, mutta on kuitenkin toteutukseltaan lähes vastaava.

Kuvasta 6.13 voidaan havaita, että Neo4J graafitietokannasta on käytössä versio 5 ja sitä suoritetaan Google Cloud Platformissa (GCP) Belgiassa. Lisäksi kuvasta näkyy käytettyjen noodien (Nodes) ja suhteiden (Relationships) lukumäärä suhteessa AuraDB:n ilmaisen tilauksen 200 000 ja 400 000 kiintiöihin. Hallintasivun



Kuva 6.13: AuraDB:n hallintanäkymä Neo4J graafitietokannalle

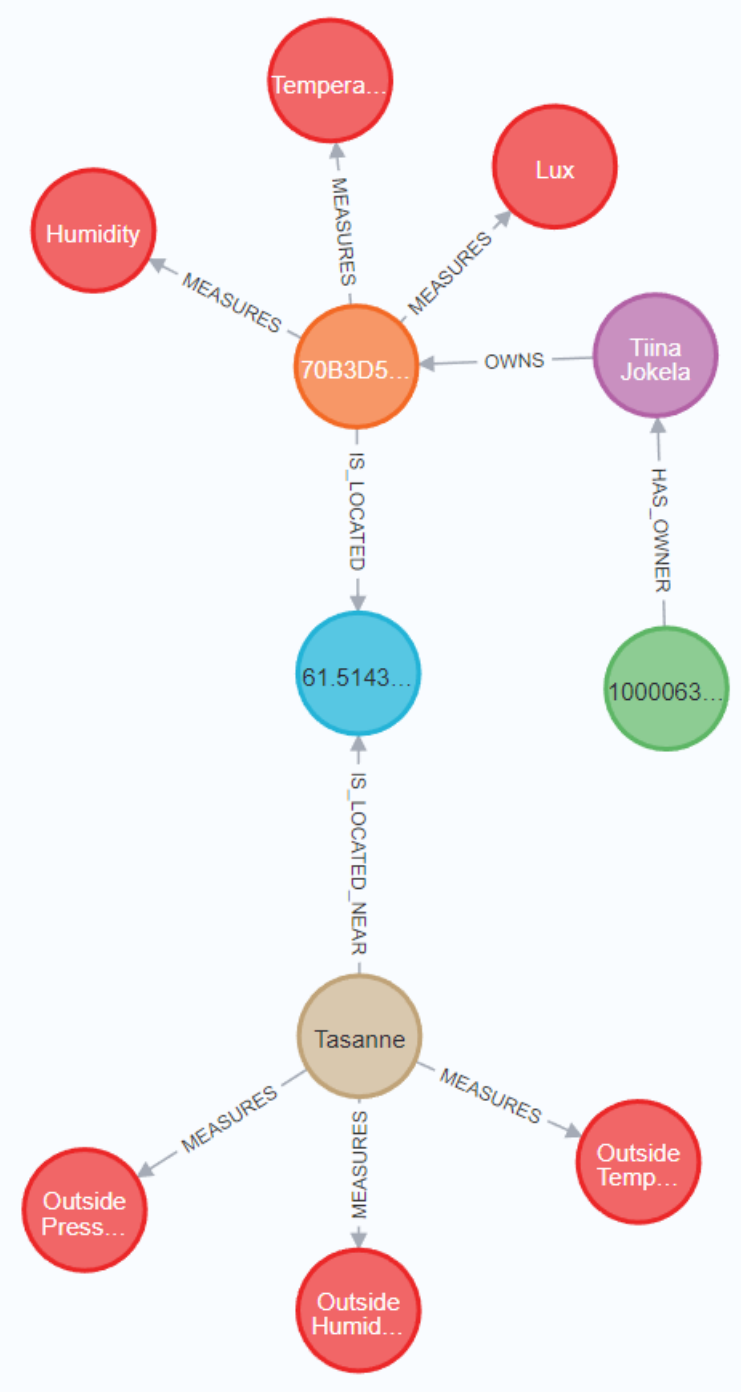
Query-painikkeesta pääsee Neo4J Browseriin [74], jonka kautta voi tehdä tietokantaan Cypher-kyselyitä. Hallintasivulta voi helposti tyhjentää käytössä olevan tietokannan, mikä on varsinkin projektin alkuvaiheessa kätevää, kun IoT-järjestelmän tietomalli saattaa muuttua paljon. Muita maininnan arvoisia toimintoja ovat mahdollisuus tuoda tietokanta muualta palveluun, tietokannan uudelleen nimeäminen ja poistaminen. Lisäksi Explore-painikkeesta pääse Neo4J Bloom-palveluun, jolla voi visualisoida graafitietokannan sisältöä [73].

Tallennettuja tietoja on mahdollista hakea IoT-järjestelmään toteutetun käyttöliittymän lisäksi esimerkiksi Neo4J:n Browserista Cypher-kyselyillä. Neo4J Browser tarjoaa kätevän tavan tarkastella tallennettuja arvoja joko visuaalisena graafina, taulukkona tai tekstinä. Toteutetussa IoT-järjestelmässä graafitietokantaan tallennetaan tietoja sensorista, sensorin omistajasta, sensorin mittausarvoja ja säähavaintoja. Edellisessä alaosiossa 6.7.1 esiteltiin sensorin mittausarvot tallentava koodi kuvassa 6.11. Kyseisen koodin luomassa Cypher-kyselyssä luodaan aina mittausarvolle solmu, jos mittauksen suorittanut sensorisolmu ja sen sijaintisolmu löytyvät jo tietokannasta. Cypher-kysely sensoreiden luomiselle poikkeaa mittausarvojen luomisesta, sillä sensorille halutaan luoda uusi solmu ainoastaan silloin, kun sille ei vielä löydy tietokannasta solmua. Käytännössä sensorin solmu luodaan, kun ensimmäiset mittausarvot saapuvat palvelimelle ja tietokantaan. Edellä mainittu toiminnallisuus saavutetaan käyttämällä Cypherista löytyvää avainsanaa MERGE. Alla on esitelty Cypher-kysely, jolla luodaan toteutetussa IoT-järjestelmässä sensori käyttäen MERGE-avainsanaa. Kyselyssä tarkastetaan ensiksi MATCH-avainsanalla, että tietokannasta löytyy sensorin omistaja (Owner) annetulla tunnisteella (SensativeId), jonka jälkeen MERGE-avainsanalla luodaan OWNS-suhde omistajan ja sensorin välille. Mikäli suhdetta sensorisolmuineen ei jo löydy, annetaan sensorisolmulle

DevEUI-tunniste luonnin yhteydessä ON CREATE-avainsanalla.

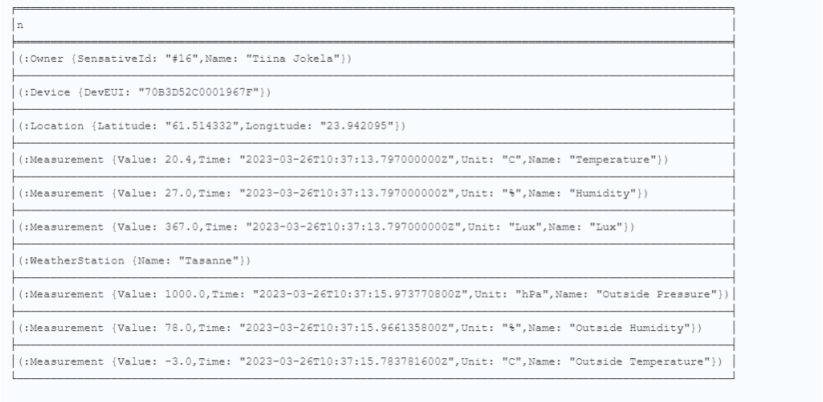
```
MATCH (o:Owner {SensativeId: $sensativeId})
MERGE (o)-[:OWNS]->(d:Device {DevEUI: $devEUI})
ON CREATE SET d.DevEUI = $devEUI
```

Kuvassa 6.14 on visualisoitu toteutetun IoT-järjestelmän otsikoitu ominaisuussisältöinen graafitietomalli yhden sensorin osalta Neo4J Browseria käyttäen. Tietomalli muodostui esitettyyn muotoon toteutuksen edetessä. Toteutuksen alussa graafitietokantaan tallennettiin ainoastaan sensorin omistaja, sensori ja mittaustulokset. Toteutuksen edetessä tietomalliin lisättiin sensorin sijainti, sääasema ja sääaseman mittaukset. Viimeisenä muutoksena tietomalliin lisättiin asiakassolmu. Myös solmujen väliset suhteet muuttuivat toteutuksen edetessä. Graafitietokannassa tietomallin muutokset ovat helppoja toteuttaa, koska relaatiotietokannasta tuttuja migraatioita ei tarvitse toteuttaa. Solmujen ja suhteiden lisääminen on yksinkertaista, koska niille ei tarvitse luoda tietokantatauluja tai viiteavaimia. Tietomallin graafinen visualisointi on yksi graafitietokannan vahvuuksista. Kuvasta 6.14 on esimerkiksi nopeasti havaittavissa, että Tiina Jokela omistaa oranssin sensorisolmun, joka mittaa sisätilan suhteellista kosteutta, lämpötilaa ja valoisuutta. Tiinan omistama sensori sijaitsee sijaintia kuvaavassa sinisessä solmussa. Lähellä samaa sijaintia on beige sääasema, joka mittaa ulkolämpötilaa, painetta ja ilman kosteutta. Vihreä solmu edustaa Kokkolan yliopistokeskuksen asiakkuutta Thingparkissa, joka kokoaa kaikki kolme toteutetun IoT-järjestelmän sensoria allensa. Kuvan tietomallista oli-si nopeasti nähtävissä, mikäli sensori ei mittaisikaan esimerkiksi lämpötilaa, koska kuvassa ei silloin näytettäisi kyseistä mittausarvosolmua. Tai vastavuoroisesti on helppo havaita se, mikäli tietokantaan tallennetaisiin vahingossa kahdesti samat arvot, koska mittausarvosolmuja olisi tuplasti.



Kuva 6.14: Toteutetun IoT-järjestelmän graafitietomalli

Kuvassa 6.15 on esitetty tekstinä sama tieto Neo4J Browserissa, kuin mikä edellä visualisoitiin graafina. Graafisesta visualisoinnista poiketen tekstitaulukosta näkee heti mittausarvot ja muut solmuihin tallennetut ominaisuudet. Graafisessa toteutuksessa ominaisuuksien näkeminen vaatii solmun klikkaamisen. Sen sijaan tekstitaulukosta ei näe suoraan solmujen välisiä suhteita, vaikkakin solmujen järjestys taulukossa antaa osviittaa suhteisiin. Kuvasta 6.15 näemme, että Measurement eli mittausarvosolmuun on tallennettu mitattu arvo (Value), aika (Time), yksikkö (Unit) ja nimi (Name). Sääasemasolmussa on sääaseman nimi, kun taas sensorisolmuun on tallennettu sensorin yksilöivä DevEUI-tunniste. Sensorin omistajasta tallennetaan nimi ja id.



Kuva 6.15: Graafitietokantaan tallennettuja tietoja tekstitaulukkona

Toteutetussa IoT-järjestelmässä graafitietokantaan tallennettua tietoa visualisoidaan käyttöliittymällä. Käyttöliittymällä käyttäjä valitsee haluamansa aikavälin, sensorin omistajan ja sensorin mittaamista suureista joko lämpötilan tai kosteuden. Käyttäjän valitsemien parametrien perusteella verkkosovelluksen palvelinkoodissa muodostetaan Cypher-kysely, jolla haetaan Neo4J graafitietokannasta pyydetty tieto edellä esitettyyn tietomalliin perustuen. Käytännössä Cypher-kysely näyttää seuraavanlaiselta, kun käyttöliittymältä on valittuna aikaväliksi maaliskuun viimeiset seitsemän päivää, lämpötila mitatuksi suureeksi ja sensorin omistajaksi Ilkka Jokela, joka omistaa sensorin DevEUI-tunnisteella "70B3D52C0001962A":

```
MATCH
(d:Device {DevEUI: '70B3D52C0001962A'})
-[:IS_LOCATED]->(l:Location)<-[:IS_LOCATED_NEAR]
-(ws:WeatherStation)-[:MEASURES]->
(om:Measurement {Unit: 'C'})
WHERE
om.Time >= datetime('2023-03-24')
AND om.Time <= datetime('2023-03-31')
RETURN
om.Name AS name, om.Unit AS unit,
om.Time AS time, om.Value AS value
UNION
MATCH
(d:Device {DevEUI: '70B3D52C0001962A'})
-[:MEASURES]->(m:Measurement {Unit: 'C'})
WHERE
m.Time >= datetime('2023-03-24')
AND m.Time <= datetime('2023-03-31')
RETURN
m.Name AS name, m.Unit AS unit,
m.Time AS time, m.Value AS value
```

Kyselyssä huomionarvoista on se, että siinä liitetään UNION-avainsanalla kahden eri kyselyn tulokset yhteen. Ensimmäisessä kyselyssä haetaan lähellä halutun sensorin sijaintia sijaitsevan sääaseman mittaamia lämpötiloja maaliskuun viimeiseltä seitsemältä päivältä. Toisessa kyselyssä haetaan halutun sensorin mittaamia lämpötiloja samalta ajanjaksolta. Sensorin sijainniksi on tallennettu LoRa-yhdyskäytävän sijainti, joka vastaanottaa sensorin lähettämän tiedon, koska sensorin omaa tarkkaa sijaintia ei ole saatavilla. Näin ollen tallennettu sijainti ei ole aina sama, vaikka sensori pysyisi paikallaan. Valittu tapa mallintaa sijainnin, sensorin ja sääaseman välinen suhde mahdollistaa sen, että sensoria voidaan liikutella ilman, että tietomalliin tarvitsee tehdä muutoksia. Kysely vaikuttaa huomattavasti vaikeammalta kuin se on, koska graafitietokanta on nopea etsimään asioiden välisiä suhteita ja se suoriutuu kyselystä rivakasti.

7 Tulokset

Tutkimuksen toteutuksen tuloksena syntyi olosuhteita mittaava ja niitä visualisoiva IoT-järjestelmä, jonka tietokantana toimii Neo4J AuraDB graafitietokanta. Graafitietokannan valintaan johtaneet syyt ja valinnan tulos ovat esitetty alaluvussa 7.1. Alaluvussa 7.2 esitetään IoT-järjestelmän väliohjelmisto- ja sovelluskerroksen toteutuksessa huomioitavia seikkoja, jotka perustuvat tutkimuksen toteutuksessa tehtyihin havaintoihin. Alaluvussa 7.3 esitetään tutkimuksen tuloksena syntynyt arkkitehtuuripäätösten dokumentaatiomalli graafitietokantaa hyödyntävän IoT-järjestelmän väliohjelmisto- ja sovelluskerroksen suunnittelun ja toteutuksen tueksi. Samassa alaluvussa 7.3 demonstroidaan se, miten mallia hyödynnettiin tutkimuksen toteutuksessa. Alaluvussa 7.4 pohditaan miten tutkimuksen toteutus onnistui, ja mitä mahdollisia jatkotutkimusideoita nousi esiin.

7.1 Graafitietokannan valinta

Tutkimusta suunnitellessa selvisi nopeasti, että Neo4J on selvästi suosituin ja tuetuin graafitietokanta, joten se oli luonnollinen valita IoT-järjestelmän tietokannak-si. Sen sijaan pohdintaa aiheutti se, missä Neo4J-tietokantaa kannattaa suorittaa ja ylläpitää. Esimerkiksi Microsoft Azure, jolla on tuotteina useita eri tietokantoja palveluina, ei tarjoa Neo4J-tietokantaa kuin kalliina Enterprise-versiona. Neo4J:lla on tietokantapalvelimestaan ilmainen Community edition ei kaupalliseen käyttöön, jota voi halutessaan suorittaa valitsemassaan pilvipalvelussa virtuaalikoneessa tai Docker-kontissa. Toteutuksen alkuvaiheessa tehtiin onnistuneita kokeiluja tietokantapalvelimen pyörittämisestä Docker-kontissa paikallisessa ympäristössä. Kuitenkin Neo4J:n ohjeistukseen ja dokumentaatioon perehtyessä kävi ilmi, että heillä on Neo4J AuraDB palvelu.

Neo4J suosittelee AuraDB:n ilmaista Free-tilausta protoiluun ja pieniin projekteihin. Neo4J AuraDB Free osoittautuikin loistavaksi palveluksi, sillä siihen oli helppo muodostaa yhteys ja se toimi yllättävän nopeasti rajatusta kapasiteetista huolimatta. Palvelussa oli mahdollisuus tyhjentää käytössä oleva tietokanta yhdellä klikkauksella, joka on kätevää kokeiluvaiheessa. Lisäksi AuraDB:ssä oli erinomaiset ohjeet

tietokantayhdeyden muodostamiseen koodista. Tutkimuksen kannalta oli kuitenkin tärkeää, että Neo4J AuraDB on normaali Neo4J-graafitietokanta, jotta tutkimuksen tavoitteista ei tarvitse tinkiä.

7.2 Huomioitavat seikat

Tämän alaluvun lopuksi esitellään lista IoT-järjestelmän väliohjelmisto- ja sovelluskerroksen suunnittelussa ja toteutuksessa huomioitavia seikkoja, jotka perustuvat tutkimuksen toteutuksen yhteydessä tehtyihin havaintoihin ja kokemuksiin. Havainnot ja kokemukset esitellään ennen listaa. IoT-järjestelmän väliohjelmisto- ja sovelluskerroksen komponentit toteuttiin pilvipalveluihin tai pilvipalveluina, joista suurin osa sijoittui Microsoft Azureen. Graafitietokannaksi valikoitui Neo4J AuraDB, joka on Neo4J tarjoama graafitietokanta palveluna.

Neo4J:lla on virallisia ohjelmistokomponentteja eri ohjelmointikielille, joilla voi muodostaa tietokantayhteyden ja Cypher-kyselyitä koodista. Dokumentaatio ja ohjeistus on kattavaa virallisille komponenteille. C#:n osalta yhteisön kehittämä Neo4JClient on kuitenkin huomattavasti suositumpi kuin virallinen Neo4J.Driver komponentti, vaikka Neo4JClient:n dokumentaatio on selvästi heikompi. Yksi syy suosioon saattaa olla se, että Neo4JClientissa Cypher-kyselyiden muodostaminen on huomattavasti helpompaa kuin virallisessa Neo4J.Driver:ssa, johtuen yksinkertaisemmasta syntaksista. Toteutuksen aikana selvisi, että Neo4JClientin tuki AuraDB:lle ei ole paras mahdollinen. Neo4JClientissa on bugi [52], joka aiheuttaa sen, että yhteyttä palvelinkoodista graafitietokantaan ei muodosteta optimaalisella tavalla. Bugista ei kuitenkaan aiheutunut haittaa, joka olisi vaikuttanut toteutetun IoT-järjestelmän toimintaan. Neo4JClient on latausmäärien perusteella niin suosittu, että todennäköisesti siitä korjataan kyseinen bugi, mutta päivitysten saatavuus on aina epävarmaa yhteisöön nojaavissa ohjelmistokomponenteissa ja kirjastoissa.

Neo4JClientilla solmujen luontikyselyjä tehdessä ilmeni, että tallennettavien solmujen ominaisuuksien kirjoitusasulla on merkitystä isojen ja pienien kirjainten osalta. Kannattaakin kirjoittaa ensiksi kysely Cypherilla tietokannan hallintatyökaluun, esimerkiksi Neo4J Browseriin, ja varmistaa, että kyselyy toimii, ja vasta sitten muuntaa se C#-koodiksi. Näin toimimalla välttää sen, että aikaa ei tuhraannu mahdollisten kirjoitusvirheiden takia. Sama huomio oli kirjattu Neo4JClientin dokumentaatioon [20]. Havainto oli merkittävä, sillä relaatiotietokantojen kanssa yleensä käytössä on ollut jokin ORM-työkalu, joka automaattisesti huolehtii luokkien ominai-

suuksien oikeasta kirjoitusasusta. Mittausarvojen luontikyselyissä meni hetki, ennen kuin oikea kysely muodostui, koska aluksi osasta sensorimittauksista tuli duplikaattisolmuja. Tässäkin auttoi se, että ensin loi Cypher-kyselyn ja testasi sitä suoraan tietokantaa vasten, ja vasta sitten muunti sen C#-koodiksi. Duplikaattisolmujen havaitseminen oli kuitenkin helppoa, koska ne olivat nopeasti nähtävissä Neo4J:n graafisesta esitystavasta.

Cypher-kyselyiden osalta haasteita aiheutti Neo4JClient tapa käsitellä päivämääriä. Ajan kanssa tehtävät vertailut tuntuivat vaikeilta, sillä vastaukset palauttivat aika-arvoja, jotka olivat haetun aikavälin ulkopuolella tai eivät kääntyneet C#:n tuntemiksi päivämääriksi. Kyselyt kuitenkin toimivat suoraan Neo4J Browserin kautta suoritettaessa odotetunlaisesti. Ongelman pystyi kiertämään Neo4JClientia käytettäessä sillä, että aina aikamääreitä sisältävissä Cypher-kyselyissä muodosti datetime-objektin, vaikka Neo4JClientin olisi pitänyt osata serialisoida C#:n DateTime- ja DateTimeOffset-objektit oikeaan muotoon. Lopulta koodin outo toiminta osoittautuikin bugiksi, josta oli tehty raportti Neo4JClientille [23]. Bugi koskee Neo4JClientia kirjoitushetkellä vain silloin, kun käytössä on Neo4J:sta versio 5. Kyseinen bugi aiheutti ylimääräistä työtä jonkin verran, mutta toisaalta se lisäsi ymmärrystä siitä, että miten aikamääreitä käsitellään Neo4J-tietokannassa.

Tietomallin toteutuksessa pohdintaa aiheutti se, miten sensorin mittausarvojen tallennuksen yhteydessä haettavien säätietojen mittausarvot, sääasema ja niiden välinen suhde sensoriin ja sen mittausarvoihin mallinnetaan, jotta mittausarvojen hakeminen on nopeaa. Toteutettu ratkaisu, jossa sensoria lähellä olevan LoRa-yhdyskäytävän sijainti on sensoria ja sääasemaa yhdistävä tekijä, on kohtuullisen nopea hakea. Lisäksi ratkaisu vaikuttaa siltä, että kyseisellä mallilla IoT-järjestelmä olisi laajennettavissa, vaikka yhden omistajan tai koko sovelluksen sensorimäärä kasvaisi. Graafitietokannan tietomallin joustavuus suhteessa relaatiotietokantaan tuli toteutuksen aikana hyvin esiin. Esimerkkinä mainittakoon asiakassolmu, joka yhdistää saman Thingpark-asiakkuuden alla olevat sensorit ja niiden omistajat. Asiakassolmu lisättiin vasta IoT-järjestelmän toteutuksen lopussa, mikä tuo hienosti esiin graafitietokannan yhden vahvuuden joustavan tietomallin. Asiakassolmun lisääminen vaati palvelinkoodissa vain Asiakas-luokan luonnin, jolla on Thingparkista saatu id. Graafitietokantaan asiakasta tallentaessa asiakassolmulle luodaan Has_Owner suhde omistajasolmuun, jos sellaista ei jo ole ennestään. Relaatiotietokannassa vastaavan suhteen lisääminen vaatisi sen, että ensiksi lisätään Asiakas-taulu tietokantaan, jonka jälkeen pitää luoda Owner-tauluun uusi vierasavain, joka viittaa Asiakas-

taulun pääavaimeen. Relaatiotietokannan operaatio vaatii tietokantamigraation, kun graafitietokannassa operaatio on vain solmun ja suhteen lisääminen.

Toteutuksen aikana kirkastui väliohjelmistokerroksen tärkeys ylimääräisen tiedon karsimisessa, kun osa mittausarvoista jäi tallentamatta tietokantaan. Toteutuksen alkuvaiheessa sensoreilta palvelinkoodille saapuva viesti sisälsi paljon sellaista tietoa, jota ei hyödynnetä järjestelmässä, mutta jota yritettiin varmentaa palvelinkoodissa oikeelliseksi. Karsimalla palvelimelle saapuvaa viestiä Azure Functionsissa saavutettiin se, että varmennettavien muuttujien määrä verkkosovelluksen palvelinkoodissa on huomattavasti vähäisempi. Tämä johti siihen, että palvelinkoodissa ei tule saapuvan viestin varmennuksessa virheitä, jotka estävät mittausarvojen tallentamisen tietokantaan. Turhien muuttujien karsiminen johti myös siihen, että koodin luettavuus parani huomattavasti, jonka myötä mahdolliset koodin virheet on helpompi havaita. Toisaalta mikäli karsittuja tietoja haluaisikin tulevaisuudessa käyttää, joutuu väliohjelmiston koodia muokkaamaan, mikä aiheuttaa lisätyötä.

Yksi pilvipalveluiden Microsoft Azureen keskittämisen hyödyistä oli salaisuuksien, kuten salasanojen, hallinnoinnin helppous Azuren Key Vault [62] palvelussa. Saman toimijan palvelussa on helppo antaa oikeuksia toisille sovelluksille hakea tietoja vaarantamatta tietoturvaa. Microsoft Azuren palveluiden käyttäminen tarkoitti sitä, että graafitietokantaa ja siihen liittyviä palveluita lukuunottamatta toteutuksessa ei jouduttu opiskelemaan uusia pilvipalveluita. Toisaalta Ruoste-projektista tuttujen komponenttien käyttö mahdollisti sen, että toteutuksesta tuli laaja sisältäen verkkosovelluksen käyttöliittymineen. Toteutetun järjestelmän käyttöliittymäkoodi on ohjelmoitu ReactJS-ohjelmistokirjastoa hyödyntäen. Ohjelmistokirjastolle on tarjolla paljon eri komponenttikirjastoja, joilla voi visualisoida aikasarjadataa diagrammein. Lopulta valinta kohdistui react-chartsjs2-kirjastoon, koska sillä oli suosituista kirjastoista kätevintä visualisoida kahta eri tulosjoukkoa samassa diagrammeissa. Ohjelmistokomponentteja on helppo kokeilla ja vaihdella, joten niiden testaaminen oli mielekästä, vaikka eri kirjastojen dokumentaation laajuus vaihtelee huomattavasti. Tulosten visualisointi vaati erillisen kirjaston käytöstä huolimatta tiedon ryhmittelyä palvelinkoodissa.

Toteutuksen aikana verkkosovelluksen arkkitehtuuri muutettiin kerrosarkkitehtuuriin, joka mahdollistaa sen, että sovelluksen eri osa-alueet ja riippuvuudet on helppo erottaa toisistaan. Lisäksi kerrosarkkitehtuuri mahdollistaa teoriassa sen, että taustalla olevan tietokannan vaihtaminen vaatii vain tietokantayhteyden sisältävän kerroksen muuttamiseen. Toteutetun IoT-järjestelmän osalta arkkitehtuurin

muuttamisen vaatima työ oli kohtalaisen laaja. .Net 7 ohjelmistokehyksen perussovelluspohja olisi varmasti ollut arkkitehtuuriltaan riittävä. Toisaalta nyt toteutettu ratkaisu mahdollistaa sovelluksen laajennoksen helpommin. Huomionarvoista on se, että .Net 7 sovelluspohjasta on saatavissa eri versioita, jotka perustuvat Reac-tin create-react-app:in sijasta muihin ohjelmistokirjastoihin kuten VueJS:ään. Lisäk-si saatavissa on versio, jossa on valmiiksi tuki Typescriptille.

Edellä esiteltyihin havaintoihin ja kokemuksiin perustuen ohessa on lista IoT-järjestelmän väliohjelmisto- ja sovelluskerroksella suunnittelussa ja toteutuksessa huomioitavista seikoista.

- Tarkista yhteisön tuottamat ohjelmistokomponentit virallisten lisäksi, huomioi kuitenkin niiden päivitystiheys ja yhteensopivuus eri tilanteissa.
- Kirjoita ja testaa tarvitsemasi tietokantakyselyt ensiksi tietokannan tukemal-la kyselykielellä suoraan tietokantaa vastaan, jonka jälkeen voit toteuttaa sen haluamallasi ohjelmointikielellä.
- Huomioi ja perehdy siihen, miten valitsemasi tietokantajärjestelmä ja ohjelmistokehys käsittelee päivämääriä ja aikaleimoja.
- Suunnittele järjestelmän tietomalli siten, että se on laajennettavissa ja tukee järjestelmän käyttötarkoitusta.
- Suunnittele ja toteuta sovelluskerroksen ohjelmistoarkkitehtuuri helposti laajennettavaksi ja muokattavaksi.
- Tallenna ja käsittele väliohjelmistokerroksella vain tarvittavat tieto ja karsi ylimääräinen tieto pois.
- Hyödynnä pilvipalveluiden tarjoamia salaisuuksien ja salasanojen hallintajärjestelmiä.
- Käytä jo ennestään tuttuja pilvipalveluita, ohjelmistokomponentteja ja kirjastoja säästääksesi aikaa, mutta tutki ja uskalla kokeilla uusia paremmin soveltuvia vaihtoehtoja.

7.3 Arkkitehtuuripäätöksen dokumentaatiomalli

Tutkimuksen tuloksena syntyi arkkitehtuuripäätösten dokumentaatiomalli IoT-järjestelmän väliohjelmisto- ja sovelluskerroksen suunnittelun ja toteutuksen tuek-

si. Dokumentaatiomalliin on yhdistetty ominaisuuksia arkkitehtuuripäätösten yksityiskohtaisista kuvauksista, arkkitehtuuripäätöksiin vaikuttavien voimien kuvauksista ja arkkitehtuuripäätösten arvioinnin dokumentoinnista. Malli on tarkoitettu arkkitehtuuripäätösten tueksi, dokumentointiin ja arviointiin. Käyttämällä mallia voidaan tehdä paremmin dokumentoituja ja harkittuja päätöksiä. Dokumentaatiomallilla on tarkoitus myös auttaa tulevia graafitietokantaa IoT-järjestelmässään hyödyntäviä projekteja, mutta myös muita projekteja, jotka tekevät arkkitehtuuripäätöksiä väliohjelmisto- ja sovelluskerroksen osalta. Malli on käytettävissä joko sellaisenaan tai muokattavissa projektin tarpeiden mukaisesti. Tehtyjen arkkitehtuuripäätösten dokumentointi on tärkeää, jotta sidosryhmillä kuten asiakkailla, kehittäjillä ja muilla arkkitehdeillä on ymmärrys toteutetusta arkkitehtuurista. Mallia kannattaa hyödyntää osana IoT-järjestelmän iteratiivista kehitysprosessia, jossa arkkitehtuuripäätökset muodostuvat, muuttuvat ja mahdollisesti poistuvat suunnittelun ja toteutuksen edetessä. Projektin aikana tulee tehdä valtava määrä eri kokoisia päätöksiä. Jokaista päätöstä ei kannata dokumentoida, sillä se kasvattaa merkittävästi kirjattavan ja ylläpidettävän dokumentaation määrää. Dokumentaatiomalliin kannattaakin kirjata kirjaushetkellä merkitykselliseksi koetut ja oletetut asia.

Taulukossa 7.1 on dokumentointuna yksittäinen toteutetun IoT-projektin arkkitehtuuripäätös mallin mukaisesti. Dokumentoitu päätös havainnollistaa ennen kaikkea sitä, miten päätösten dokumentaatiomallia tulisi käyttää, mutta myös sitä, minkälaisia päätöksiä kannattaa dokumentoida. Dokumentoidussa päätöksessä on pohdittu Neo4J AuraDB Freen valitsemista graafitietokannan ajoympäristöksi. Ennen varsinaista päätöksen tekemistä on listattu ja pisteytetty päätöstä tukevat ja päätös-tä vastustavat seikat, jotta voidaan arvioida päätöksen järkevyyttä. Päätöstä tukevat seikat on pisteytetty positiivisesti yhdestä kolmeen, kun taas vastustavat seikat negatiivisesti yhdestä kolmeen. Laskemalla pisteet yhteen havaitaan, että päätös olisi selkeästi positiivisen puolella (pisteet vastaan 7 ja pisteet puolesta 12). Pisteytys-tä ei voi kuitenkaan pitää ainoana päätöksen ajurina, koska pisteytys koskee vain niitä seikkoja, jotka on ymmärretty kirjauksen hetkellä. Arkkitehtuuripäätösten dokumentaatiota tuleekin päivittää, kun päätöksiin liittyvä tieto lisääntyy. Esimerkin päätöksessä on listattu vaihtoehtoisia ajoympäristöjä graafitietokannalle. Vaihtoehtoihin kannattaa listata sellaisia vaihtoehtoja, jotka ovat oikeasti vaihtoehtoja valitulle päätökselle ja joita on oikeasti pohdittu. Päätös "Toteutetaan järjestelmän graafitietokanta Neo4J AuraDB palvelun ilmaisella Free-versiolla"on dokumentoitu lyhyesti, mutta perusteluihin on kirjattu tärkeimmät päätökseen vaikuttaneet tekijät.

Perusteluista voidaan havaita, että ne ovat hyvin pitkälti päätöstä tukevia seikkoja. Mikäli päätös olisi ollut vastakkainen, tulisi perustelujen pohjautua päätöstä vastustaviin seikkoihin. Dokumentoidun päätöksen seurauksiin on kirjattu tarkkaan tehdystä päätöksestä aiheutuvat seuraukset ja rajoitteet. Seurauksissa on myös selkeäs-ti yksi uusi päätös dokumentoitavaksi, sillä jossain vaiheessa on päätettävä, muutetaanko tili maksulliseksi vai vaihdetaanko ajoympäristöä. Liittyviin päätöksiin on listattu päätökset, joihin dokumentoitu päätös liittyy läheisesti. Dokumentaatiomallin lopussa on vielä arviointi päätöksen onnistumisesta ja perustelut arvioinnille. Dokumentoidun päätöksen lopputulos on arvioitu onnistuneeksi lyhyiden perusteluiden kera.

Nimi	Neo4J AuraDB Free
Tila	Valmis
Aihealue	Teknologiavalinnat
Aihe	Graafitietokantajärjestelmän ajoympäristön valinta
Päätös	Toteutetaan järjestelmän graafitietokanta Neo4J AuraDB palvelun ilmaisella Free-versiolla
Perustelut	Neo4J AuraDB Free tarjoaa maksuttomasti täysiverisen graafitietokannan, jonka rajoitteet mahdollistavat IoT-järjestelmän toteuttamisen. Palvelussa on hyvä dokumentaatio, jonka perusteella käyttöönotto on helppoa. Aikaa säästyy paljon, kun ei tarvitse omaa palvelinta pystyttää.
Vaihtoehdot	Neo4J Community Edition omalla palvelimella, Neo4J Community Edition pilvipalvelussa, Neo4J AuraDB Professional
Seuraukset	Ohjelmointikieltä ja palvelinohjelmointiympäristöä valittaessa on huomioitava saatavilla olevat ajurit ja kirjastot tietokantaan liittymistä varten. Valitussa palvelussa on rajoitettu tallennettavien solmujen ja suhteiden lukumäärää, joten jossain vaiheessa on päivitettävä tili maksulliseksi tai vaihdettava tietokantajärjestelmän ajoympäristöä mikäli rajat täyttyvät. Neo4J AuraDB Free on hostattu GCP:ssä Belgiassa, joten mahdolliset latenssit on huomioitava.
Liittyvät päätökset	Ohjelmointikielet, palvelinohjelmointi ja jatkokehitys

Päätöstä tukevat seikat	• Maksuttomuus ++ • Ei tarvetta omalle palvelimelle ++ • Automaattiset päivitykset + • Täysiverinen graafitietokanta +++ • Käyttöönoton helppous ja nopeus ++ • Hyvä dokumentaatio ++
Päätöstä vastustavat seikat	• Ei mahdollista valita ajoympäristön sijaintia ja pilvipalvelua - • Ei säätömahdollisuuksia - • Vain yksi tietokantainstanssi ilmaisessa versiossa -- • Solmujen ja suhteiden lukumäärä rajoitettu --
Päätöksen lopputulos	Onnistunut
Lopputuloksen perustelut	Tietokantajärjestelmä toimi odotetunlaisesti ja sen käyttöönotto oli helppo. Ilmaisversion rajoitteet solmuille ja suhteille eivät täyttyneet projektin aikana.

Taulukko 7.1: Esimerkki toteutetun arkkitehtuuripäätöksen dokumentoinnista

7.4 Pohdinta

Tutkimuksen ensisijaisena tavoitteena oli selvittää, mitä tulee huomioida graafitietokantaa hyödyntävän IoT-järjestelmän väliohjelmisto- ja sovelluskerroksen suunnittelussa ja toteutuksessa. Lisäksi tavoitteena oli selvittää mitä graafitietokantavaih-

toehtoja on saatavilla, ja soveltuuko graafitietokanta IoT-järjestelmän tietokantajärjestelmäksi. Näiden lisäksi tutkimuksen tavoitteena oli selvittää, miten tietokannan valinta ja muut arkkitehtuuripäätökset kannattaa dokumentoida. Tutkimus osoit-ti, että graafitietokantaa hyödyntävän IoT-järjestelmän väliohjelmisto- ja sovelluskerroksen suunnittelussa ja toteutuksessa tulee huomioida lukuisia seikkoja. Huomioitavissa seikoissa korostuu se, että järjestelmän arkkitehtuuri ja tietomalli tulee olla helposti laajennettavissa ja muokattavissa. Lisäksi väliohjelmistokerroksella tulee karsia järjestelmälle tarpeeton ylimääräinen tieto pois. Tulokset noudattelevat Kuswah et al. [48] ja Da Cruz et al. [25] tekemiä havaintoja IoT-järjestelmän väliohjelmisto- ja sovelluskerroksen tehtävistä.

Tutkimuksessa selvisi, että graafitietokanta soveltuu hyvin IoT-järjestelmän tietokantajärjestelmäksi. Graafitietokannan tietomalli tukee IoT-järjestelmien tuottamaa moninaista dataa ja tietomalli on tarvittaessa helposti laajennettavissa. Tietomallin graafinen esitystapa on kätevä, sillä siitä havaitsee tehokkaasti entiteettien väliset suhteet ja riippuvuudet. Samaan lopputulemaan olivat myös päätyneet Fosic ja Solic [32] aiemmassa tutkimuksessaan. Tutkimuksessa selvisi myös se, että Neo4J on graafitietokannoista selkeästi suosituin, monipuolisin ja tuetuin, ja näin ollen paras vaihtoehto graafitietokannaksi IoT-järjestelmään. Tutkimus osoitti, että saatavilla ei ole yhtä yleisesti hyväksyttyä arkkitehtuuripäätösten dokumentaatiomallia, minkä vuoksi sellainen luotiin tutkimuksen aikana.

Tutkimuksen tuloksia voidaan pitää suuntaa-antavina, sillä tutkimuksessa toteutettiin vain yksittäinen IoT-järjestelmän konstruktio. Lisäksi tutkimus rajattiin koskemaan ainoastaan olosuhteita mittaavaa ja visualisoivaa IoT-järjestelmää. Tulosten luotettavuutta laskee se, että konstruktio tehtiin yhden suunnittelusyklin aikana eikä konstruktiosta kerätty ulkopuolista palautetta ja kehitysideoita. Tutkimus on toisaalta toistettavissa, sillä se on dokumentoitu hyvin ja lähdekoodit ovat saatavissa pyydettäessä.

Tutkimuksen tarjoama tieto auttaa graafitietokannoista ja IoT:sta kiinnostuneita tutkijoita ja tutkimusyhteisöä hyödyntämään graafitietokantaa IoT-järjestelmissä. Tutkimuksen tarjoaman tiedon merkitystä korostaa se, että aiheesta ei ole tehty paljon tutkimusta. Tässä tutkimuksessa toteutettu IoT-järjestelmä oli tyypillinen olosuhteita mittaava ja visualisoiva sovellus, jossa tietoa rikastettiin myös säätiedoilla. Tulevaisuudessa olisi mielenkiintoista nähdä, miten graafitietokanta soveltuu kompleksisempaan IoT-järjestelmään, jossa havainnointikerroksen laitteiden ja mitattavien suureiden lukumäärä on suurempi, laitteita on useammalta valmistajalta

ja mittausdataa käytetään ohjaamaan älykodin laitteita. Lisäksi olisi mielenkiintoista tutkia ja vertailla, miten graafitietokannan tietomallin joustavuus vertautuu suhteessa dokumentti- ja relaatiotietokantoihin, kun IoT-järjestelmän tietomalliin tuodaan uusia laitteita ja relaatioita. Tällä hetkellä graafitietokannoista selkeästi suosituin ja tuetuin on Neo4j, joten muiden saatavilla olevien graafitietokantojen tutkiminen olisi myös suotavaa. Graafitietokantojen tutkiminen IoT:n yhteydessä ei tarvitse rajoittua väliohjelmisto- ja sovelluskerrokseen, sillä osasta graafitietokannoista on saatavilla versioita, jotka on suunniteltu sulautetuille laitteille.

Graafitietokantoja on perinteisesti hyödynnetty sosiaalisten verkostojen kuvaamisessa ja petosten havaitsemissa. Esineiden internetin suosion yhä voimakkaas-ti kasvaessa on todennäköistä, että graafitietokantojen käyttö IoT-järjestelmissä lisääntyy, sillä niiden tietomalli soveltuu erinomaisesti IoT-järjestelmien tarpeisiin. Graafitietokantaa hyödyntävän IoT-järjestelmän väliohjelmisto- ja sovelluskerroksen suunnittelussa ja toteutuksessa huomioitavat seikat ovat hyvin tyypillisiä väliohjelmisto- ja sovelluskerroksella huomioitavia seikkoja, vaikka graafitietokantojen kyselykieli, tietokantajärjestelmä ja käytettävissä olevat ohjelmistokomponentit voivat vaikuttaa omalta osaltaan kerroksien suunnitteluun ja toteutukseen.

8 Yhteenveto

Tämän tutkielman tarkoituksena oli selvittää mitä tulee huomioida graafitietokantaa hyödyntävän IoT-järjestelmän väliohjelmisto- ja sovelluskerroksen suunnittelussa ja toteutuksessa. Lisäksi tutkielmassa haluttiin selvittää mitä graafitietokantavaihtoehtoja on saatavilla, soveltuuko graafitietokanta IoT-järjestelmän tietokantajärjestelmäksi ja miten dokumentoida tietokannan valinta ja muut arkkitehtuuripäätökset. Tutkielman tutkimus toteutettiin suunnittelututkimuksena. Suunnittelututkimuksessa suunniteltiin ja toteutettiin graafitietokantaa hyödyntävä IoT-järjestelmä, joka mittaa ja visualisoi olosuhteita.

Tutkimuksessa selvisi, että graafitietokantaa hyödyntävän IoT-järjestelmän väliohjelmisto- ja sovelluskerroksen suunnittelussa ja toteutuksessa on huomioitava, että järjestelmän arkkitehtuuri ja tietomalli on helposti laajennettavissa ja muokattavissa. Lisäksi on huomioitava, että väliohjelmistokerroksella karsitaan järjestelmälle tarpeeton ja ylimääräinen tieto. Tutkimus osoitti graafitietokannan soveltuvan hyvin IoT-järjestelmän tietokantajärjestelmäksi, koska graafitietokannan tietomalli on joustava ja helposti laajennettavissa. Graafitietokannan tietomallin visuaalinen esitystapa havainnollistaa hyvin eri entiteettien väliset suhteet ja riippuvuudet. Graafitietokantavaihtoehdoista Neo4J on paras IoT-järjestelmän graafitietokannaksi, sillä se on selkeästi suosituin, monipuolisin ja tuetuin. Tutkimuksessa kävi myös ilmi, että arkkitehtuuripäätösten, kuten IoT-järjestelmän tietokantajärjestelmän valinta, dokumentointiin ei ole yhtä kaikkiin projekteihin sopivaa mallia. Tästä johtuen tutkimuksessa luotiin arkkitehtuuripäätösten dokumentaatiomalli, joka soveltuu IoT-järjestelmän väliohjelmisto- ja sovelluskerroksen suunnitteluun ja toteutukseen.

Graafitietokantojen suosio jatkaa kasvuaan esineiden internetin kasvun rinnalla. Graafitietokantojen tietomalli soveltuu erinomaisesti IoT-järjestelmien tarpeisiin, joissa korostuvat esineiden ja asioiden väliset suhteet, laitteiden ja datan monimuotoisuus sekä tiedon valtava määrä. Haasteita graafitietokantojen suosion räjähdykselle tuottanee SQL:stä poikkeavat kyselykielet ja laajasti tuettujen graafitietokantajärjestelmien vähäinen määrä. Neo4J kaipaa rinnalleen haastajia graafitietokantajärjestelmissä. Lisäksi olisi toivottavaa, että Cypherista tulisi SQL:n kaltainen standardi graafitietokantajärjestelmille.

Lähteet

[1] ACTILITY. Actility thingpark and digita will take iot to new heights with a national lora network for finland. URL https://www.actility.com/actility-thingpark-and-digita-will-take-iot-to-new-heights-with-a-national-lora-network-for-finland/, viitattu 22.4.2023.

[2] ACTILITY. Creating a microsoft azure connection. URL https://docs.thingpark.com/thingpark-x/latest/Connector/AZURE/, viitattu 22.4.2023.

[3] ACTILITY. Creating an aws connection. URL https://docs.thingpark.com/thingpark-x/latest/Connector/AWS/, viitattu 23.4.2023.

[4] ACTILITY. Thingpark enterprise platform for dedicated enterprise iot lorawan networks. URL https://www.actility.com/enterprise-iot-connectivity-solutions/, viitattu 15.4.2023.

[5] ACTILITY. Uplink message. URL https://docs.thingpark.com/thingpark-x/latest/Message/Uplink_Message/, viitattu 23.4.2023.

[6] AL-QASEEMI, S. A., ALMULHIM, H. A., ALMULHIM, M. F., JA CHAUDHRY, S. R. Iot architecture challenges and issues: Lack of standardization. Julkaisusarjassa *2016 Future technologies conference (FTC)* (2016), IEEE, 731–738.

[7] ALEX IVANOVS. The most popular front-end frameworks in 2023. URL https://stackdiary.com/front-end-frameworks/, viitattu 30.4.2023.

[8] AMAZON WEB SERVICES, INC. Aws app runner. URL https://aws.amazon.com/apprunner/, viitattu 23.4.2023.

[9] AMAZON WEB SERVICES, INC. Aws iot core. URL https://docs.aws.amazon.com/whitepapers/latest/aws-overview/internet-of-things-services.html#aws-iot-core, viitattu 23.4.2023.

[10] AMAZON WEB SERVICES, INC. Using aws lambda with aws iot. URL https://docs.aws.amazon.com/lambda/latest/dg/services-iot.html, viitattu 23.4.2023.

[11] AMGHAR, S., CHERDAL, S., JA MOULINE, S. Which nosql database for iot applications? Julkaisusarjassa *2018 International Conference on Selected Topics in Mobile and Wireless Networking (MoWNeT)* (2018), 131–137.

[12] ANGLES, R., ARENAS, M., BARCELÓ, P., HOGAN, A., REUTTER, J., JA VRGOČ, D. Foundations of modern query languages for graph databases. *ACM Computing Surveys (CSUR) 50*, 5 (2017), 1–40.

[13] ANTHAPU, R. *Graph Data Processing with Cypher*. "Packt Publishing ", 2022.

[14] ASIMINIDIS, C., KOKKONIS, G., JA KONTOGIANNIS, S. Database systems performance evaluation for iot applications. *International Journal of Database Management Systems (IJDMS) Vol 10* (2018).

[15] AZAD, P., NAVIMIPOUR, N. J., RAHMANI, A. M., JA SHARIFI, A. The role of structured and unstructured data managing mechanisms in the internet of things. *Cluster computing 23* (2020), 1185–1198.

[16] BERG, K. L., SEYMOUR, T., GOEL, R., ET AL. History of databases. *International Journal of Management & Information Systems (IJMIS) 17*, 1 (2013), 29–36.

[17] BRETT SMITH. A quick guide to gplv3. URL https://www.gnu.org/licenses/quick-guide-gplv3.html, viitattu 6.5.2023.

[18] CHAMBERLIN, D. D. Early history of sql. *IEEE Annals of the History of Computing 34*, 4 (2012), 78–82.

[19] CHAMBERLIN, D. D., JA BOYCE, R. F. Sequel: A structured english query language. Julkaisusarjassa *Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) workshop on Data description, access and control* (1974), 249–264.

[20] CHARLOTTE SKARDON. cypher examples. URL https://github.com/DotNet4Neo4j/Neo4jClient/wiki/cypher-examples, viitattu 7.5.2023.

[21] CHARLOTTE SKARDON JA TATHAM ODDIE. Neo4jclient. URL https://github.com/DotNet4Neo4j/neo4jclient, viitattu 30.4.2023.

[22] CHEN, P. P.-S. The entity-relationship modeltoward a unified view of data. *ACM transactions on database systems (TODS) 1*, 1 (1976), 9–36.

[23] CMAART. Datetime deserialized as null or default when using version >= 5. URL https://github.com/DotNet4Neo4j/Neo4jClient/issues/464, viitattu 7.5.2023.

[24] CODD, E. F. A relational model of data for large shared data banks. *Communications of the ACM 13*, 6 (1970), 377–387.

[25] DA CRUZ, M. A., RODRIGUES, J. J. P., AL-MUHTADI, J., KOROTAEV V., V., JA DE ALBUQUERQUE, V. H. C. A reference model for internet of things middleware. *IEEE Internet of Things Journal 5*, 2 (2018), 871–883.

[26] DENTON, J. W., JA PEACE, A. G. Selection and use of mysql in a database management course. *Journal of Information Systems Education 14*, 4 (2003), 401.

[27] DIÈNE, B., RODRIGUES, J. J., DIALLO, O., NDOYE, E. H. M., JA KOROTAEV, V. V. Data management techniques for internet of things. *Mechanical Systems and Signal Processing 138* (2020), 106564.

[28] D'SILVA, G. M., THAKARE, S., JA BHARADI, V. A. Real-time processing of iot events using a software as a service (saas) architecture with graph database. Julkaisusarjassa *2016 International Conference on Computing Communication Control and automation (ICCUBEA)* (2016), 1–6.

[29] ERIKSSON, M., JA HALLBERG, V. Comparison between json and yaml for data serialization. *The School of Computer Science and Engineering Royal Institute of Technology* (2011), 1–25.

[30] FARIN, J. Lorawan-teknologia. URL https://www.digita.fi/etusivu/palvelut-yrityksille/digitan-iot-palvelut/lorawan-teknologia/, viitattu 15.4.2023.

[31] FERNANDES, D., BERNARDINO, J., ET AL. Graph databases comparison: Allegrograph, arangodb, infinitegraph, neo4j, and orientdb. *Data 10* (2018), 0006910203730380.

[32] FOSIC, I., JA SOLIC, K. Graph database approach for data storing, presentation and manipulation. Julkaisusarjassa *2019 42nd International Convention on*

Information and Communication Technology, Electronics and Microelectronics (MIPRO) (2019), 1548–1552.

[33] GOKHALE, P., BHAT, O., JA BHAT, S. Introduction to iot. *International Advanced Research Journal in Science, Engineering and Technology 5*, 1 (2018), 41–44.

[34] GOSNELL, D., JA BROECHELER, M. *The Practitioner's Guide to Graph Data: Applying Graph Thinking and Graph Technologies to Solve Complex Problems*. "O'Reilly Media, Inc.", 2020.

[35] GUPTA, A., TYAGI, S., PANWAR, N., SACHDEVA, S., JA SAXENA, U. Nosql databases: Critical analysis and comparison. Julkaisusarjassa *2017 International conference on computing and communication technologies for smart nation (IC3TSN)* (2017), IEEE, 293–299.

[36] HASSAN, M. A. Relational and nosql databases: The appropriate database model choice. Julkaisusarjassa *2021 22nd International Arab Conference on Information Technology (ACIT)* (2021), IEEE, 1–6.

[37] HAXHIBEQIRI, J., DE POORTER, E., MOERMAN, I., JA HOEBEKE, J. A survey of lorawan for iot: From technology to application. *Sensors 18*, 11 (2018), 3995.

[38] HOLZSCHUHER, F., JA PEINL, R. Performance of graph query languages: comparison of cypher, gremlin and native access in neo4j. Julkaisusarjassa *Proceedings of the Joint EDBT/ICDT 2013 Workshops* (2013), 195–204.

[39] INFINITEGRAPH. Infinitegraph features. URL https://infinitegraph.com/technical-features/, viitattu 23.9.2023.

[40] JANSEN, A., JA BOSCH, J. Software architecture as a set of architectural design decisions. Julkaisusarjassa *5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)* (2005), IEEE, 109–120.

[41] JATANA, N., PURI, S., AHUJA, M., KATHURIA, I., JA GOSAIN, D. A survey and comparison of relational and non-relational database. *International Journal of Engineering Research & Technology 1*, 6 (2012), 1–5.

[42] JOHANNESSON, P., JA PERJONS, E. *An introduction to design science*, vol. 10. Springer, 2014.

[43] JYVÄSKYLÄN YLIOPISTO. Tietotekniikan ja koulutusteknologian tutkinto-ohjelmien pro gradu -tutkielma. URL https://www.jyu.fi/it/fi/ohjeita-opiskelijalle/opiskelu/pro-gradu-tutkielma/tietotekniikka, viitattu 19.5.2023.

[44] KHANNA, A., JA KAUR, S. Internet of things (iot), applications and challenges: a comprehensive review. *Wireless Personal Communications 114*, 2 (2020), 1687–1762.

[45] KIM, W. Object-oriented databases: Definition and research directions. *IEEE Transactions on knowledge and Data Engineering 2*, 3 (1990), 327–341.

[46] KIVELÄ, I., JA JYRKKÄ, P. Opiskelijat mukana ruoste-projektissa. URL https://cinetcampus.fi/uutiset/opiskelijat-mukana-ruoste-projektissa/, viitattu 16.4.2023.

[47] KÜÇÜKKEÇECI, C., JA YAZICI, A. Big data model simulation on a graph database for surveillance in wireless multimedia sensor networks. *Big data research 11* (2018), 33–43.

[48] KUSHWAH, R., BATRA, P. K., JA JAIN, A. Internet of things architectural elements, challenges and future directions. Julkaisusarjassa *2020 6th International Conference on Signal Processing and Communication (ICSC)* (2020), IEEE, 1–5.

[49] LI, T., LIU, Y., TIAN, Y., SHEN, S., JA MAO, W. A storage solution for massive iot data based on nosql. Julkaisusarjassa *2012 IEEE International conference on green computing and communications* (2012), IEEE, 50–57.

[50] MA, M., WANG, P., JA CHU, C.-H. Data management for internet of things: Challenges, approaches and opportunities. Julkaisusarjassa *2013 IEEE International conference on green computing and communications and IEEE Internet of Things and IEEE cyber, physical and social computing* (2013), IEEE, 1144–1151.

[51] MADAKAM, S., LAKE, V., LAKE, V., LAKE, V., ET AL. Internet of things (iot): A literature review. *Journal of Computer and Communications 3*, 05 (2015), 164.

[52] MCALVIN-KINECTIFY. Working with aura db. URL https://github.com/DotNet4Neo4j/Neo4jClient/issues/445, viitattu 7.5.2023.

[53] META OPEN SOURCE. React. URL https://react.dev/, viitattu 30.4.2023.

[54] MICROSOFT. App service. URL https://azure.microsoft.com/en-us/products/app-service/, viitattu 15.4.2023.

[55] MICROSOFT. Azure functions. URL https://azure.microsoft.com/en-gb/products/functions, viitattu 15.4.2023.

[56] MICROSOFT. Azure iot hub. URL https://azure.microsoft.com/en-us/products/iot-hub, viitattu 15.4.2023.

[57] MICROSOFT. Build in the cloud free with azure for students. URL https://azure.microsoft.com/en-us/free/students/, viitattu 23.4.2023.

[58] MICROSOFT. Build serverless applications faster with your azure free account. URL https://azure.microsoft.com/en-gb/free/serverless/, viitattu 23.4.2023.

[59] MICROSOFT. Build. test. deploy. URL https://dotnet.microsoft.com/en-us/, viitattu 15.4.2023.

[60] MICROSOFT. Common web application architectures. URL https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures, viitattu 30.4.2023.

[61] MICROSOFT. Continuous deployment to azure app service. URL https://learn.microsoft.com/en-us/azure/app-service/deploy-continuous-deployment?tabs=github, viitattu 23.4.2023.

[62] MICROSOFT. Key vault. URL https://azure.microsoft.com/en-us/products/key-vault, viitattu 7.5.2023.

[63] MICROSOFT. .net programming languages. URL https://dotnet.microsoft.com/en-us/languages, viitattu 30.4.2023.

[64] MICROSOFT. Register a new device in the iot hub. URL https://learn.microsoft.com/en-us/azure/iot-hub/iot-hub-create-through-portal#register-a-new-device-in-the-iot-hub, viitattu 22.4.2023.

[65] MICROSOFT. Supported languages in azure functions. URL https://learn.microsoft.com/en-us/azure/azure-functions/supported-languages, viitattu 23.4.2023.

[66] MICROSOFT. Use react with asp.net core. URL https://learn.microsoft.com/fi-fi/aspnet/core/client-side/spa/react?view=aspnetcore-7.0&tabs=visual-studio, viitattu 30.4.2023.

[67] MISRA, G., KUMAR, V., AGARWAL, A., JA AGARWAL, K. Internet of things (iot)–a technological analysis and survey on vision, concepts, challenges, innovation directions, technologies, and applications (an upcoming or future generation computer communication system technology). *American Journal of Electrical and Electronic Engineering 4*, 1 (2016), 23–32.

[68] NAVANI, D., JAIN, S., JA NEHRA, M. S. The internet of things (iot): A study of architectural elements. Julkaisusarjassa *2017 13th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS)* (2017), IEEE, 473–478.

[69] NEO4J INC. Building neo4j applications with .net. URL https://graphacademy.neo4j.com/courses/app-dotnet/?ref=guides, viitattu 30.4.2023.

[70] NEO4J INC. Creating an instance. URL https://neo4j.com/docs/aura/auradb/getting-started/create-database/, viitattu 6.5.2023.

[71] NEO4J INC. Neo4j auradb. URL https://neo4j.com/cloud/platform/aura-graph-database/, viitattu 15.4.2023.

[72] NEO4J INC. Neo4j auradb overview. URL https://neo4j.com/docs/aura/auradb/, viitattu 6.5.2023.

[73] NEO4J INC. Neo4j bloom. URL https://neo4j.com/product/bloom/, viitattu 6.5.2023.

[74] NEO4J INC. Neo4j browser user interface guide. URL https://neo4j.com/developer/neo4j-browser/, viitattu 6.5.2023.

[75] NEO4J INC. Neo4j pricing. URL https://neo4j.com/pricing/, viitattu 6.5.2023.

[76] NEO4J INC. Using neo4j from .net. URL https://neo4j.com/developer/dotnet/, viitattu 30.4.2023.

[77] OPENWEATHER. Current weather data. URL https://openweathermap.org/current, viitattu 6.5.2023.

[78] OPENWEATHER. Openweather. URL https://openweathermap.org/, viitattu 15.4.2023.

[79] ORDONEZ, C., SONG, I.-Y., JA GARCIA-ALVARADO, C. Relational versus non-relational database systems for data warehousing. Julkaisusarjassa *Proceedings of the ACM 13th international workshop on Data warehousing and OLAP* (2010), 67–68.

[80] PETROV, A. *Database Internals: A Deep Dive into How Distributed Data Systems Work*. "O'Reilly Media, Inc.", 2019.

[81] POKORNÝ, J. Graph databases: their power and limitations. Julkaisusarjassa *Computer Information Systems and Industrial Management: 14th IFIP TC 8 International Conference, CISIM 2015, Warsaw, Poland, September 24-26, 2015, Proceedings 14* (2015), Springer, 58–69.

[82] RAUTMARE, S., JA BHALERAO, D. Mysql and nosql database comparison for iot application. Julkaisusarjassa *2016 IEEE international conference on advances in computer applications (ICACA)* (2016), IEEE, 235–238.

[83] REACT-CHARTJS-2. react-chartjs-2. URL https://react-chartjs-2.js.org/, viitattu 30.4.2023.

[84] ROBINSON, I., WEBBER, J., JA EIFREM, E. *Graph databases: new opportunities for connected data*. "O'Reilly Media, Inc.", 2015.

[85] RYS, M. Scalable sql: How do large-scale sites and applications remain sql-based? *Queue 9*, 4 (2011), 30–37.

[86] SENSATIVE AB. Adding strips to a lorawan network. URL https://public.yggio.net/docs/strips/multi-sensor/#joining-strips-to-your-network, viitattu 16.4.2023.

[87] SENSATIVE AB. Battery calculator for strips lorawan sensors. URL https://sensative.com/sensors/strips-sensors-for-lorawan/sensative-strips-lora-battery-calculator/, viitattu 16.4.2023.

[88] SENSATIVE AB. Configuration strips. URL https://public.yggio.net/docs/strips/multi-sensor/#configuration-strips, viitattu 16.4.2023.

[89] SENSATIVE AB. Strips configuration application. URL https://strips-lora-config-app.service.sensative.net/profiles, viitattu 16.4.2023.

[90] SENSATIVE AB. strips-lora-translator-open-source. URL https://gitlab.com/sensative/strips-lora-translator-open-source, viitattu 16.4.2023.

[91] SENSATIVE AB. Strips multi-sensor +comfort for lorawan. URL https://sensative.com/sensors/strips-sensors-for-lorawan/strips-lorawan-ms-comfort/, viitattu 15.4.2023.

[92] TECHNOSTACKS. Top 15 react chart libraries for your web projects. URL https://technostacks.com/blog/react-chart-libraries/, viitattu 30.4.2023.

[93] TEOREY, T. J., YANG, D., JA FRY, J. P. A logical design methodology for relational databases using the extended entity-relationship model. *ACM Computing Surveys (CSUR) 18*, 2 (1986), 197–222.

[94] TIWARI, S. *Professional nosql*. John Wiley & Sons, 2011.

[95] TUDORICA, B. G., JA BUCUR, C. A comparison between several nosql databases with comments and notes. Julkaisusarjassa *2011 RoEduNet international conference 10th edition: Networking in education and research* (2011), IEEE, 1–5.

[96] TYREE, J., JA AKERMAN, A. Architecture decisions: Demystifying architecture. *IEEE software 22*, 2 (2005), 19–27.

[97] UDARA BIBILE. N-tier architecture in asp.net core. URL https://chathuranga94.medium.com/n-tier-architecture-in-asp-net-core-d1f1b14f2010, viitattu 30.4.2023.

[98] UETA, K., XUE, X., NAKAMOTO, Y., JA MURAKAMI, S. A distributed graph database for the data management of iot systems. Julkaisusarjassa *2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)* (2016), 299–304.

[99] VAN HEESCH, U., AVGERIOU, P., JA HILLIARD, R. A documentation framework for architecture decisions. *Journal of Systems and Software 85*, 4 (2012), 795–820.

[100] VAN HEESCH, U., AVGERIOU, P., JA HILLIARD, R. Forces on architecture decisions-a viewpoint. Julkaisusarjassa *2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture* (2012), IEEE, 101–110.

[101] VAN HEESCH, U., ELORANTA, V.-P., AVGERIOU, P., KOSKIMIES, K., JA HARRISON, N. Decision-centric architecture reviews. *IEEE software 31*, 1 (2013), 69–76.

[102] VICKNAIR, C., MACIAS, M., ZHAO, Z., NAN, X., CHEN, Y., JA WILKINS, D. A comparison of a graph database and a relational database: a data provenance perspective. Julkaisusarjassa *Proceedings of the 48th annual Southeast regional conference* (2010), 1–6.

[103] WIERINGA, R. *Design Science Methodology for Information Systems and Software Engineering*. 01 2014.

[104] YANG, Z., YUE, Y., YANG, Y., PENG, Y., WANG, X., JA LIU, W. Study and application on the architecture and key technologies for iot. Julkaisusarjassa *2011 International Conference on Multimedia Technology* (2011), IEEE, 747–751.

[105] ZHANG, J., MA, M., WANG, P., JA SUN, X.-D. Middleware for the internet of things: A survey on requirements, enabling technologies, and solutions. *Journal of Systems Architecture 117* (2021), 102098.

[106] ZHONG, C.-L., ZHU, Z., JA HUANG, R.-G. Study on the iot architecture and gateway technology. Julkaisusarjassa *2015 14th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES)* (2015), 196–199.

[107] ZHU, Z., HUANG, R.-G., ET AL. Study on the iot architecture and access technology. Julkaisusarjassa *2017 16th International Symposium on Distributed Computing and Applications to Business, Engineering and Science (DCABES)* (2017), IEEE, 113–116.