TEOLLISUUDEN PILVIPOHJAISEN PUNNITUSPALVELUN LAITEHALLINNAN KEHITTÄMINEN

Case: Lahti Precision Oy

LAB-AMMATTIKORKEAKOULU
Insinööri (AMK)
Tieto- ja viestintätekniikka
Ohjelmistotekniikka
Kevät 2020
Veikko Lehmuskorpi

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Lehmuskorpi, Veikko	Opinnäytetyö, AMK	Kevät 2020
	Sivumäärä	
	36	

Työn nimi

Teollisuuden pilvipohjaisen punnituspalvelun laitehallinnan kehittäminen
Case: Lahti Precision Oy

Tutkinto

Insinööri (AMK)

Tiivistelmä

Työssä toteutettiin käyttöliittymä punnituspalvelun laitehallintaa varten Lahti Precision Oy:lle. Punnituspalveluun voidaan liittää useita erilaisia laitteita, joita on toimitettu pääosin ympäri Pohjois-Eurooppaa. Laitteilla on verkkoyhteys, jonka ansiosta niitä voidaan hallita etänä.

Laitteiden tiedot tallennetaan tietokantaan ja niitä käsitellään REST API -rajapinnan avulla. Käyttöliittymä toteutettiin Angular-sovelluskehyksellä (framework) hyödyntäen mikropalveluarkkitehtuuria.

Käyttöliittymästä pyrittiin tekemään keskeinen paikka kaikkea laitteen hallinnointia varten. Käyttöliittymä pidettiin myös mahdollisimman yksinkertaisena ja helppokäyttöisenä, karsimatta ominaisuuksia, sillä palvelulla on käyttäjiä useissa eri rooleissa.

Lopputuloksena oli toimiva käyttöliittymä, joka voitiin julkaista sisäiseen käyttöön. Käyttöliittymän kautta voitiin luoda, muokata ja tarkastella laitteita. Laitteet sisälsivät perustietojen lisäksi muun muassa muistiinpanoja ja tilatietoja laitteen kunnosta. Laitteiden tarkastelun ja hallinnan yhdistäminen yhteen koottuun paikkaan helpotti sekä asiakaspalvelijoiden että sovelluskehittäjien työtä.

Asiasanat

laitehallinta, punnituspalvelu, pilvipalvelu, mikropalvelut, konttiteknologia

Abstract

Author(s)	Type of publication	Published
Lehmuskorpi, Veikko	Bachelor's thesis	Spring 2020
	Number of pages	
	36	

Title of publication

Development of the asset management of an industrial weighing service
Case: Lahti Precision Oy

Name of Degree

Bachelor's Degree Programme in Information and Communications Technology

Abstract

The objective of the thesis was to implement a user interface for asset management for Lahti Precision Oy. Processes surrounding their weighing services include many kind of assets which have been delivered mainly to Northern Europe. All the assets are connected to the internet so that they can be managed remotely.

All information about assets is saved into a database and that information is handled by using a REST API. The user interface was implemented with the Angular framework utilising the microservice architecture.

The goal was that the interface would work as a central location for everything needed to manage assets. It was also kept as simple and user friendly as possible without sacrificing on features, to accommodate all the different level of end-users.

The end result was a successful and working user interface that could be released into internal use inside the team. Assets can be made, edited and inspected from the user interface. Aside from the basic asset data, the assets also contain, among other things, notes and status information about the asset health. Combining of the management and inspection of assets helps the work of customer servants and software developers.

Keywords

asset management, weighing service, cloud service, microservices, container technology

SISÄLLYS

LYHENTEET

API	Ohjelmointirajapinta (Application Programming Interface)
CLI	Komentoliittymä (Command-line Interface)
CMMS	Computerized Maintenance Management System
DOM	Dokumenttioliomalli (Document Object Model)
ERP	Toiminnanohjausjärjestelmä (Enterprise Resource Planning)
FaaS	Function as a Service
HTTP	Hypertekstin siirtoprotokolla (Hypertext Transfer Protocol)
IaaS	Infrastructure as a Service
IOT	Esineiden Internet (Internet of Things)
IP	IP-protokolla (Internet Protocol)
PaaS	Platform as a Service
REST	Representational State Transfer
SaaS	Software as a Service
XaaS	Anything as a Service

1 JOHDANTO

Opinnäytetyön toimeksiantaja toimii lahtelainen teollisuusalan yritys Lahti Precision Oy, joka valmistaa ja toimittaa punnitus-, annostus- ja palveluratkaisuja, jotka ovat käytössä eri teollisuudenaloilla. Yrityksen ydinosaamista ovat punnitus-, annostus- ja irtomateriaalien käsittelyjärjestelmät sekä automaatio ja tiedonhallinta. Merkittävä osa myynnistä koostuu yrityksen tarjoamista palveluista, kuten asennukset, huolto, varmennukset ja varaosapalvelut (Lahti Precision Oy 2020a).

Yrityksellä on tavoitteena olla asiakaslähtöinen ja olla jatkuvasti punnitusprosessissa mukana, jotta kaikki sujuisi jatkossakin, myös käyttöönoton jälkeen, eli pysyä pitkäaikaisena kumppanina. Yritys haluaa olla työprosessissa mukana, mahdollisimman aikaisesta vaiheesta alkaen, jotta varmistetaan yhteistyön optimaalinen toteutus ja pitkä, tehokas elinkaari projektille. (Lahti Precision Oy 2020b.)

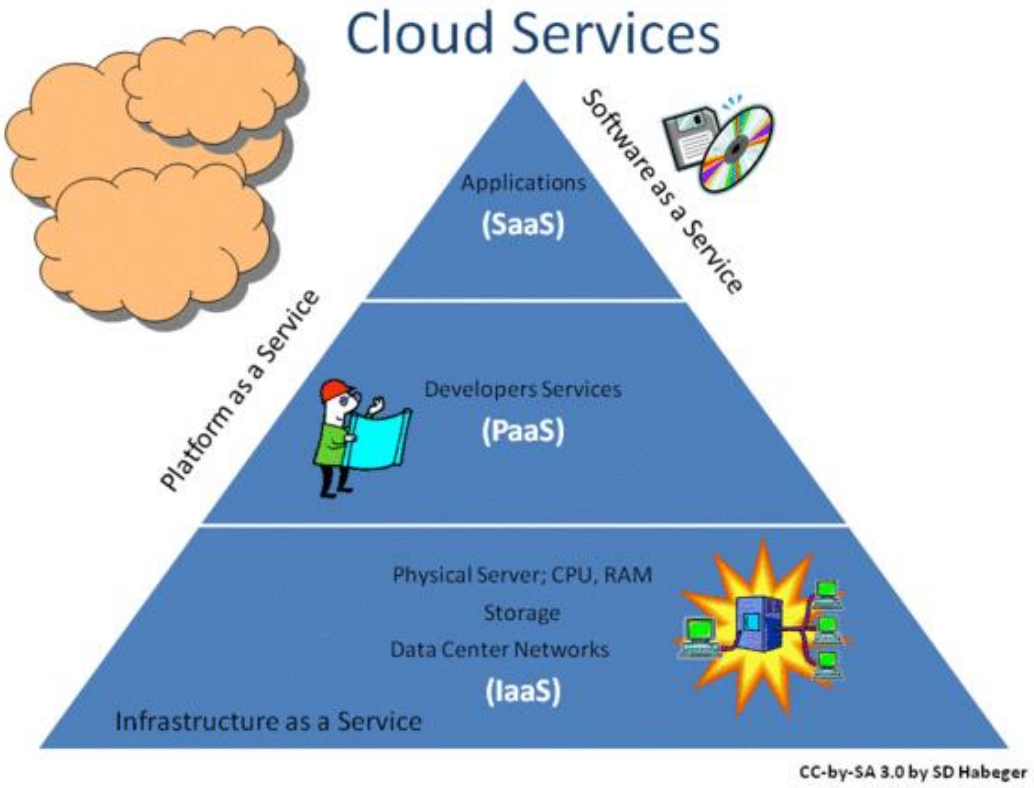
Lahti Precision on valmistanut ja toimittanut vaakoja asiakkailleen yli sata vuotta. Vaa'at ja siihen liittyvät mahdollisuudet ovat keskeisin osa yritystä, ja yrityksen tarkoituksena onkin pysyä kehityksen kärjessä myös jatkossa. Yrityksen tavoitteena on saada tulevaisuudessa kaikki vaa'at liitettäväksi pilvipohjaiseen mScales-punnituspalveluun, johon liitettyjä vaakoja voidaan käyttää entistä tehokkaammin ja niiden välittämiä tietoja voidaan hallita automaattisesti digitaalisessa muodossa. (Lahti Precision Oy 2020b.)

Opinnäytetyön tavoitteena on kehittää tapa hallita kaikkia yrityksen toimittamia laitteita sekä yrityksen sisältä että asiakkaiden puolelta. Keskeisenä tavoitteena on nähdä laitteiden tiedot mScales-palvelun käyttöliittymässä, jota voisi hyödyntää sekä ylläpitäjä että asiakas. Laitteiden tarkastelun lisäksi oleellinen osa hallintaa on laitteiden muokkaaminen ja järjestelmään lisääminen. Laitteiden perustietoja, kuten laitteen nimeä tai tyyppiä, tulisi pystyä muokkaamaan helposti mScales-palvelun kautta. (Lahti Precision Oy 2020b.)

2 TEOLLISUUS

2.1 Pilvipohjaiset palvelumallit

Pilvipohjaisia palvelumalleja on NIST-standardin mukaan kolmea eritasoista abstraktiota (kuva 1). Pilvipalveluilla voidaan kätevästi tarjota pääsy tarvittaviin resursseihin käytön mukaan. Tärkeää pilvipalveluiden jouhevassa toiminnassa on, että asiakas voi yksipuolisesti säännöstellä tarvittavien resurssien määrää ilman interaktiota palveluntarjoajan kanssa. Pilvipalvelut tarjotaan verkon välityksellä vakiomekanismeilla, joten niihin voi päästä käsiksi käytännössä millä tahansa verkkoon liitetyllä päätelaitteella. (NIST Special Publication 2011.)



Kuva 1. Eri pilvipalvelumallien tasot (Sdhabeger 2010)

2.1.1 Software as a Service

Software as a Service (SaaS) -palvelut ovat lisenssipohjaisia käytöstä maksettavia palveluja. Palveluiden ohjelmisto sijaitsee pilvessä, jota ylläpidetään palveluntarjoajan toimesta, ja ohjelmisto välitetään asiakkaille verkkoselaimen kautta. Asiakkaan näkökulmasta SaaS-palvelut ovat kustannustehokkaita ja helpompia ottaa käyttöön, koska palveluntarjoaja

vastaa ohjelmiston toiminnasta pilvessä. Tällaisen ohjelmiston käyttöönotto ei vaadi asentamista tai omia resursseja sen ylläpitoon. (Pilvi 2020.)

Hinnoittelumallin ansiosta ohjelmiston käyttöönotto on yleensä myös huomattavasti halvempaa ja turvallisempaa. Suurten riskialttiiden sijoitusten sijaan maksetaan käytöstä, ja käytön sekä maksamisen voi lopettaa myöhemmin, jos se on tarpeen. Käyttökulut syntyvät ajan mittaan silloin, kuin ohjelmisto on toivottavasti jo tuottamassa rahaa yritykselle, mikä tekee budjetoinnista helpompaa. Palveluiden saatavuus on yksi tärkeimmistä piirteistä. Ohjelmisto tarjotaan internetin välityksellä, jolloin se on saatavilla ja liitettävissä kaikkialle, mistä löytyy Internet-yhteys ja päätelaite. (Pilvi 2020.)

2.1.2 Infrastructure as a Service

Infrastructure as a Service (IaaS) -palvelut tarjoavat palveluiden suorittamiseen tarvittavia resursseja, kuten verkkoyhteyksiä, tallennustilaa ja palvelimia. Resurssit tarjotaan verkon välityksellä ja tyypillisesti niistä maksetaan käytön mukaan. Loppukäyttäjät (end users) eivät varsinaisesti ole tekemisissä fyysisten laitteiden kanssa, edes verkon välityksellä, vain yleensä yritykset tarjoavat rajapintoja laitteiden käsittelyä varten. Pilvipohjaisista palvelumalleista IaaS tarjoaa kaikkein alhaisimman tason resurssien hallinnoinnin. IaaS-palvelut mahdollistavat resurssien määrän lisäämisen tai vähentämisen sitä tarvittaessa. Alkukustannusten ei tarvitse olla suuret ja suurenkin määrän resursseja voi ottaa käyttöön vain hetkeksi. (IBM 2019.)

2.1.3 Platform as a Service

Platform as a service (PaaS) -palvelut ovat alustoja, jotka koostuvat useista eri pilvipalveluista. PaaS-palvelut ovat tyypillisesti suunnattu ohjelmistokehittäjiä varten. Alusta voi tarjota muun muassa työkalut ja resurssit sovellusten rakentamista, testaamista ja julkaisua varten. PaaS -palvelua hyödyntämällä yritys voi aloittaa sovelluskehityksen nopeammin ilman laajempia laitteisto- ja ohjelmistokustannuksia. (Rouse 2020b.)

2.1.4 Function as a Service

Function as a Service (FaaS) –palvelumalli on yksi keskeisistä serverless-arkkitehtuurimallin palveluista. Serverless-palvelut tarjoavat alustan ohjelmiston suorittamiselle, jossa käyttäjä maksaa jokaisen tapahtuman perusteella. FaaS- ja PaaS -palvelut ovat hyvin samanlaisia, mutta FaaS –palveluissa palveluntarjoaja hoitaa vieläkin isompaa osaa alustasta. (Watts 2018.)

2.2 Esineiden internet

Esineiden internet (Internet of Things), lyhennettynä IoT, tarkoittaa fyysisten internetin avulla liitettyjen laitteiden muodostamaa tiedon keräämiseen ja jakamiseen tarkoitettua verkkoa. IoT -laitteet kykenevät toimimaan itsenäisesti, keräämään ja lähettämään tietoa verkoston sisällä, vähentäen ihmisen interventiota ja nopeuttamaan tiedonkulkua. Verkostoon voidaan liittää kaikenlaisia laitteita kodinkoneista teollisuuteen. Verkoston järkevällä hyödyntämisellä on selviä etuja yrityksille; laitteita ja resursseja voidaan monitoroida automaattisesti ja tehdä automaattisia korjauksia näiden tietojen perusteella. Suurimpia IoT-alustoja ovat muun muassa Microsoftin Azure, Siemensin MindSphere, Amazonin AWS IoT sekä Googlen Cloud IoT. Kaikki edellä mainituista ovat sovellusten ohjelmointiin, testaamiseen, käyttöönottoon ja ylläpitämiseen tarkoitettuja alustoja, jotka tarjoavat muun muassa tietokanta- ja palvelinresursseja. (Dignan 2018.)

2.3 Laitehallinta

Käsitteenä laitehallinnalla tarkoitetaan järjestelmällistä lähestymistapaa jonkin ryhmän hallussa olevien asioiden arvon realisointiin. Termillä voidaan viitata joko fyysiseen omaisuuteen, kuten laitteisiin, tai abstraktiin omaisuuteen, kuten materiaalioikeuksiin. Laitehallinta koostuu omaisuuden kehittämisestä, toimenpiteistä, huollosta sekä mahdollisimman kannattavasta hävittämisestä. (The Local Government & Municipal Knowledge Base.). Teollisuuden laitehallinta koskee yleensä fyysisiä laitteita, joka sisältää koko laitteen elinkaaren, suunnittelun, käyttöönoton, operoinnin, huollon, korjaukset ja hävityksen (ISO 55000 2014.).

2.3.1 Toiminnanohjausjärjestelmä

Toiminnanohjausjärjestelmät (Enterprise Resource Planning, ERP) ovat laajoja yrityksen ohjaamiseen tarkoitettuja tietojärjestelmiä, jonka avulla hallitaan yrityksen kirjanpitoa ja resursseja. Toiminnanohjausjärjestelmien ensisijaisia tarkoituksia ovat muun muassa tehokkuuden parantaminen, organisoituvuus ja läpinäkyvyys organisaation sisällä.

Viime vuosina ERP-järjestelmät ovat kehittyneet perinteisistä itse ylläpidetyistä ohjelmistomalleista pilvipalveluiksi, joihin päästään käsiksi verkon kautta. Pilvipalveluiden käyttöönotolla on selviä hyötyjä, varsinkin pienemmille yrityksille, jotka voivat ottaa järjestelmän käyttöön ilman suurempia resursseja, kuten investointeja tai ylläpitoon vaadittua teknistä osaamista. Pilvipalvelut mahdollistavat järjestelmien jatkuvan kehityksen ja vaiheittaiset uudistukset eliminoivat suurten päivitysten ja muutosten aiheuttamat kouluttamistarpeet ja -kulut. (Logistiikan maailma 2019.)

2.3.2 Kunnossapidon tietojärjestelmä

Kunnossapidon tietojärjestelmä (Computerized Maintenance Management System, CMMS) pitää kirjaa yrityksen huolto-operaatioista (Cato & Mobley 2002) ja sen on tarkoitus tehostaa ja helpottaa huoltohenkilökunnan työtä. Järjestelmästä selviää esim. laitteiden kunto ja toimenpiteitä vaativat laitteet sekä varaosien sijainti. Järjestelmä voi myös ehdottaa loogisia ja kustannustehokkaita toimenpiteitä laitekohtaisesti, kun järjestelmään on syötetty kaikki tarvittavat tiedot. Myös kunnossapidon tietojärjestelmät ovat nousseet suosioon teknologian kehittyessä ja monet ohjelmistopaketeista ovat nykyään pilvipohjaisia, kuten toiminnanohjausjärjestelmätkin. (Wireman 1994.)

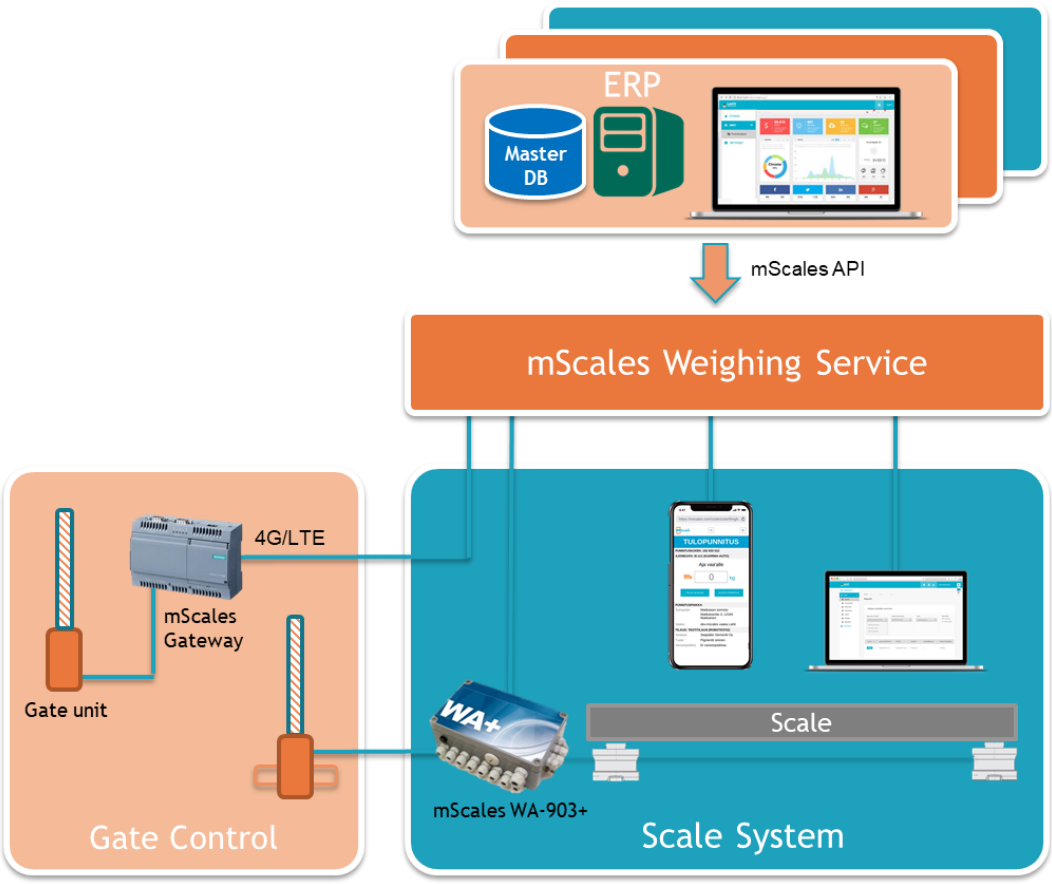
2.3.3 Vaakojen tiedonhallinta

Verkkoon liitettyjen laitteiden tuottaman suuren informaatiovirran käsittelyyn tarvitaan kehittyneitä ratkaisuja. Kaikki mahdollinen laitteiden kunnonvalvontaan liittyvä tieto tallennetaan ongelmakohtien havainnointia ja niiden jälkiselvittelyä varten. Tiedonhallintaan ja analysointiin tarvitaan laite- ja sovelluskohtaisia työkaluja. Esim. Vaakojen tapauksessa, laitteet lähettävät jatkuvasti painotietoa tilatietoineen. Vaakoja ja oheislaitteita voidaan myös diagnosoida etätyönä. Nämä tiedot yhdessä kertovat, onko laite valmiina punnitusta varten ja onko se ylipäätänsä toimintakunnossa. Vaa'an punnituksista säilytetään punnitustositteet, jotka mahdollistavat jäljitettävyyden lain edellyttämällä tavalla. mScales -verkkopalvelussa punnitustositteita voidaan tarkastella yksitellen tai niistä voidaan koostaa raportteja. Näin pysytään jatkuvasti selvillä saapuvasta ja lähtevästä materiaalivirrasta ja voidaan tehdä laskutus ajantasaisesti. (Lahti Precision Oy 2020c.)

3 TOIMINTAYMPÄRISTÖ

3.1 mScales

mScales on pilvipohjainen punnituspalvelu, johon voidaan liittää erilaisia vaakoja ja oheislaitteita merkistä riippumatta (kuva 2). mScales on moderni ja ketterä ratkaisu, jonka avulla vaa'at voidaan liittää toiminnanohjausjärjestelmään ja sen ohjaukseen. Sovellus on kaksiosainen ja se koostuu kuljettajan punnitukseen käytettävästä mobiilisovelluksesta sekä verkkopalvelusta yrityksen oman henkilökunnan ja ylläpitäjän tarpeisiin. (Lahti Precision Oy 2020c.)



Kuva 2. mScales sovellusesimerkki (Lahti Precision Oy 2020c)

Kuljettaja voi sovelluksen avulla punnita kuormansa nopeasti ja kätevästi omalla puhelimellaan nousematta ulos kulkuneuvostaan. Kuljettajalle lähetetään digitaalisesti uniikki punnituskoodi, joka on tullut mScales palvelusta ja jonka avulla haetaan kaikki kuorman sisältävät tiedot. Kuljettajan tarvitsee syöttää vain koodi sovellukseen ja ajaa vaa'alle. Enää ei myöskään tarvita fyysisiä punnituskuitteja. Paikalla oleva punnitusoperaattori seuraa punnitustapahtumaa ja hyväksyy punnituksen verkkopalvelussa. (Lahti Precision Oy 2020c.)

Punnitustapahtuma tallennetaan pilveen ja tapahtuman yksityiskohtaisia tietoja voidaan tarkastella ajantasaisena mistä tahansa. Tieto kulkee tapahtumapaikalta sitä tarvitseville saman tien, eikä punnituksia tarvitse kirjata käsin ja vasta sen jälkeen toimittaa eteenpäin. Digitaalisuus tiedon jakamisessa lisää myös turvallisuutta. mScales –palvelun tietoliikenne kulkee aina salattujen yhteyksien läpi ja on saatavilla vain yritykselle luoduilla henkilökohtaisilla käyttäjätunnuksilla. (Lahti Precision Oy 2020c.)

3.2 Laitehallinta

Toimeksiantaja haluaa paremman tavan hallita kaikkia toimittamiaan laitteitaan sekä yrityksen sisältä, että asiakkaiden puolesta. Keskeisenä tavoitteena on nähdä kaikkien laitteiden oleelliset tiedot mScales palvelusta, jota voisi hyödyntää sekä ylläpitäjä, että asiakas. Laitteiden tarkastelun lisäksi oleellinen osa on laitteiden muokkaaminen ja lisääminen. Laitteiden perustietoja, kuten laitteen nimeä tai tyyppiä voi helposti muokata mScales palvelun kautta, ja uusia asennettuja laitteita voi liittää yritykseen.

Laitelistaan halutaan näkyville kaikki laitteet yrityskohtaisesti taulukkomuodossa, jossa on laitteen nimi, tyyppi, toimipiste, tila sekä luonti- ja päivitysajat. Tavoitteena on myös tarjota kriittistä tietoa laitteista nopealla silmäyksellä, esimerkiksi laitteen tilaa kuvaavilla väreillä. Palvelusta voi nopeasti nähdä laitteen tilan ja palvelu ilmoittaa, jos jokin laite on epäkunnossa. Tieto laitteista halutaan saada mahdollisimman nopeasti liikkeelle. Ideaalisti data on reaaliaikaisesti päivittyvää.

Listan laitetta klikkaamalla pääsee laitteen yksityiskohtaiseen näkymään, jossa näytetään ihan kaikki laitteen antamat tiedot. Sivulla on tarkemmin tilatietoja ja muun muassa laitteen lokitiedot. Samalle sivulle pystyy myös liittämään laitteelle oleellisia liitteitä, kuten muistiinpanoja ja dokumentteja. Muistiinpanot hyödyttävät ylläpitäjiä sekä asentajia ja huoltajia, joiden lisäksi dokumenteista löytyy laitteen huoltosuunnitelma ja manuaalit. Kaikki mahdollinen laitteen mukana kulkeva tieto halutaan tuoda yhteen ja esittää yhdessä paikassa, helposti kaikkien osapuolten selattavissa. Tällä yksityiskohtaisella sivulla voi tarkastella myös kaikkia laitteen parametreja, jotka ovat oleellisessa osassa muun muassa laitetta asentaessa, jolloin tietyt laitteen tiedot, esimerkiksi IP-osoite, on parametroitu ja näkyvissä laitesivulla.

Toimintakunnon seuraaminen on tärkeä osa missä tahansa hallinnassa. Laitehallintaan halutaan reaaliaikaista tietoa laitteiden kunnosta sekä antureista. Mahdolliset virhetilanteet ovat saman tien näkyvillä laitesivulla ja myös yleistä kuntoa voidaan seurata samalta sivulta. Esimerkiksi vaa’an kohdalla tieto anturien toimintakunnosta on erittäin kriittinen tieto, sillä virheellinen anturin toiminta todennäköisesti johtaa myös virheellisiin painoluke-

miin. Jos järjestelmässä on tieto virhetilanteista, näiden aikana tapahtuneita tapahtumia voidaan setviä jälkikäteen ja mahdolliset virheelliset tapahtumat on helppo suodattaa kaikkien tapahtumien joukosta. Kaikista virheistä lähtee myös tieto asiakkaalle sekä ylläpitäjille.

3.3 Arkkitehtuuri

Mikropalveluarkkitehtuuri on tapa rakentaa sovelluksia, jotka oikeastaan koostuvat useammasta yksinkertaisesta sovelluksesta. Kaikkien erillisten ohjelmistojen keskustellessa keskenään ne muodostavat lopullisen sovelluksen. (Opensource.com 2019.)

Yksittäiset mikropalvelut ovat yleensä pieniä kokonaisuuksia, jotka keskittyvät vain yhteen toimialueeseen, eivätkä ne välitä muista mikropalveluiden toiminnasta, ne vain viestivät keskenään. Pienen ja yksinkertaisen kokonaisuutensa ansiosta mikropalveluita on helpompi kehittää ja huoltaa kuin yhtä laajaa ja kompleksia ohjelmistoa. Tämä toimintamalli on osoittautunut tehokkaaksi ja kannattavaksi etenkin isojen sovellusten kehitystiimeissä, varsinkin, jos on kyse jatkuvan toiminnan sovelluksesta, jossa tuotetta kehitetään ja viedään uuteen suuntaan jatkuvasti. (Opensource.com 2019.)

Uusia ominaisuuksia tai arkkitehtuurin muutoksia on helppo lähteä toteuttamaan uutena mikropalveluna tai vaihtamalla olemassa oleva mikropalvelu toiseen malliin. Yksittäisen palvelun toteutuksella ei sinänsä ole merkitystä, kunhan olemassa oleva palveluiden välinen viestintä pysyy samanlaisena. Ohjelmiston jakaminen osiin mahdollistaa erilaisten toimintatapojen käytön, esimerkiksi kokonaan eri ohjelmointikielen tai datamallin käytön palvelun sisällä. Isompi tiimi voidaan jakaa useaan pienempään tiimiin, jäsenet ja osaaminen voidaan jakaa työskentelemään tiettyjen palveluiden parissa. Useamman palvelun kehittäminen samaan aikaan eri tiimeissä edistää tehokkuutta. (Opensource.com 2019.)

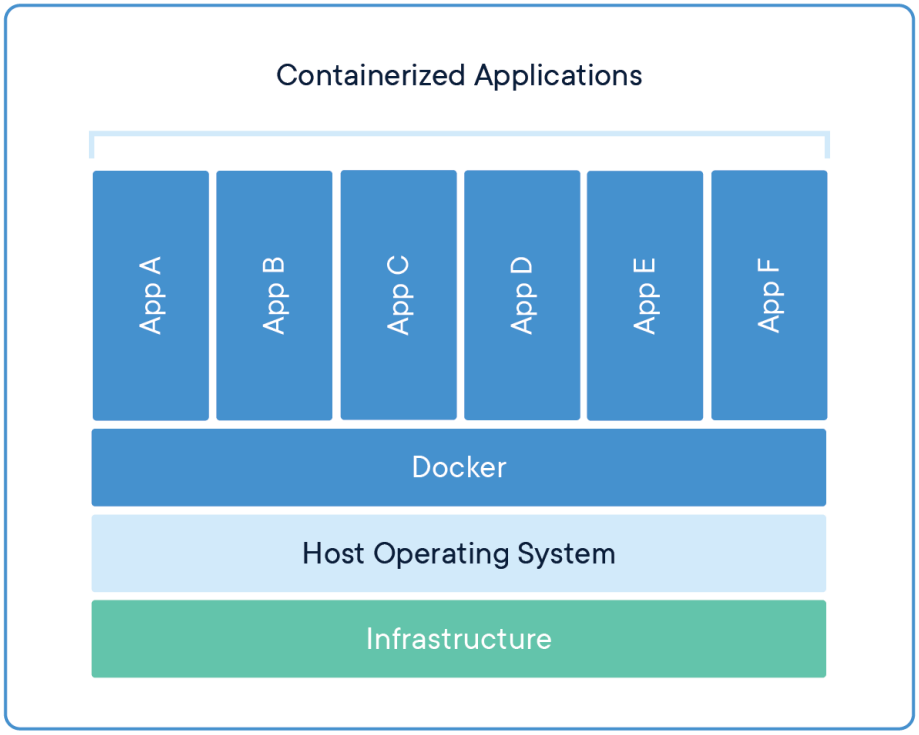
Sovelluksen jakaminen pienempiin erillisiin palveluihin ei hyödytä ainoastaan tekijöitä, vaan sillä on konkreettisia vaikutuksia jopa tuotannossa. Sovelluksella on yleensä liiketoiminnan kannalta oleellinen tarkoitus, ja siksi sovelluksen pitää toimia ja olla saatavilla jatkuvasti. Mikropalveluarkkitehtuurissa yhden palvelun kaatuminen ei tarkoita koko sovelluksen kaatumista, vaan muut palvelut jatkavat toimintaansa normaalisti, sillä välin kun kaatunut palvelu yritetään uudelleenkäynnistää. Riippuen kaatuneen palvelun tarkoituksesta ja kyseisellä hetkellä tapahtuneesta käytöstä, hetkittäisellä ongelmatilanteella ei välttämättä edes ole liiketoiminnan kannalta haittaavia vaikutuksia. Jos sovellus olisi rakennettu perinteisen monoliittirakenteen mukaan, jossakin sovellusosassa tapahtuneella virheellä olisi todennäköisesti kriittisiä vaikutuksia koko ohjelmistoon ja se voisi tarkoittaa koko sovelluksen kaatumista. Tällaisessa tilanteessa ei auta kuin yrittää käynnistää koko

sovellus uudestaan, jolloin palvelu ei ole käytettävissä ollenkaan. (Opensource.com 2019.)

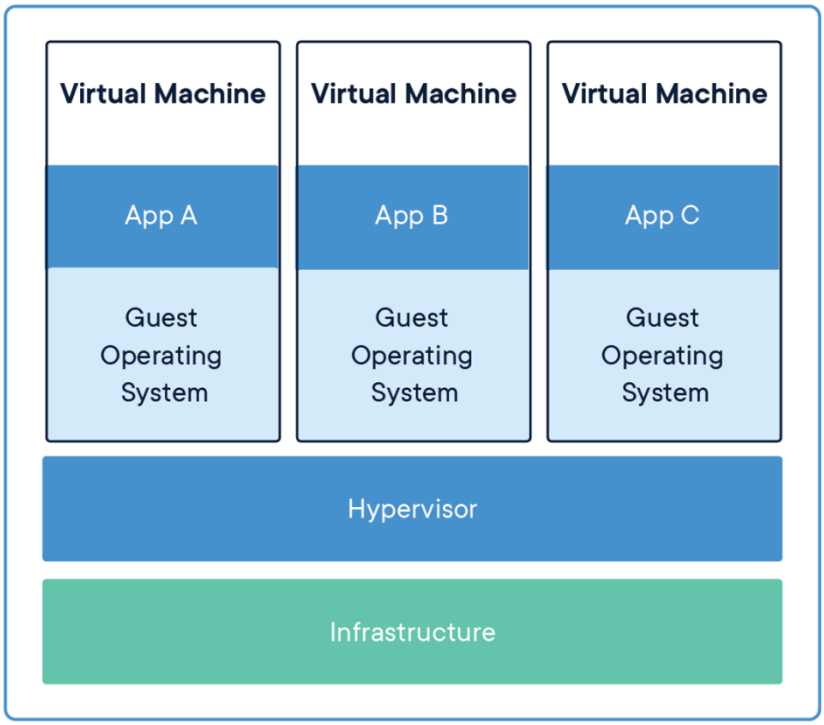
3.4 Konttiteknologia

Kontit ovat itsenäisiä ohjelmiston suoritusta varten rakennettuja standardisoituja ympäristöjä, joiden avulla varmistetaan, että ohjelmisto toimii aina samalla tavalla, riippumatta siitä, minkälaisessa ympäristössä kontti on käytössä (Docker 2020a).

Ohjelmisto ja koodin riippuvaisuudet kuten eri kirjastot pakataan kaikki yhteen, jolloin koko koodin suoritusympäristö on pakattu konttiin. Konttiteknologia eroaa virtualisoinnista sillä, että kontit eivät varsinaisesti sisällä käyttöjärjestelmää. Kaikki kontit voivat jakaa palvelimen käyttöjärjestelmän kernelin, toisinkuin virtualisoinnissa (kuva 2). Virtuaalikoneet tarvitsevat oman käyttöjärjestelmän joita suoritetaan yhdellä palvelimella (kuva 3). Kontit käyttävät siis paljon vähemmän resursseja kuin virtualisointi. Resurssitehokkuuden ansiosta useita kontteja voidaan ajaa samalla palvelimella, joka mahdollistaa mikroarkkitehtuurimallin toteuttamisen, jossa jokainen kontti on modularisoitu vastaamaan vain pienestä yksittäisestä muista konteista riippumattomasta sovellusosiosta. Modulaarisesti rakennettua sovellusta on helpompi hallita ja laajentaa, sillä yksittäisissä konteissa voidaan tehdä suurempiakin muutoksia, kunhan konttien välinen viestittely toimii odotetulla tavalla muutosten jälkeen. (Docker 2020a.)



Kuva 2. Docker konttiteknologia (Docker 2020a)



Kuva 3. Virtuaalikoneteknologia (Docker 2020b)

3.5 Yleisesti Docker sovelluksesta

Docker on avoimen lähdekoodin sovellus muiden sovellusten rakentamiseen sekä suorittamiseen kontteina.

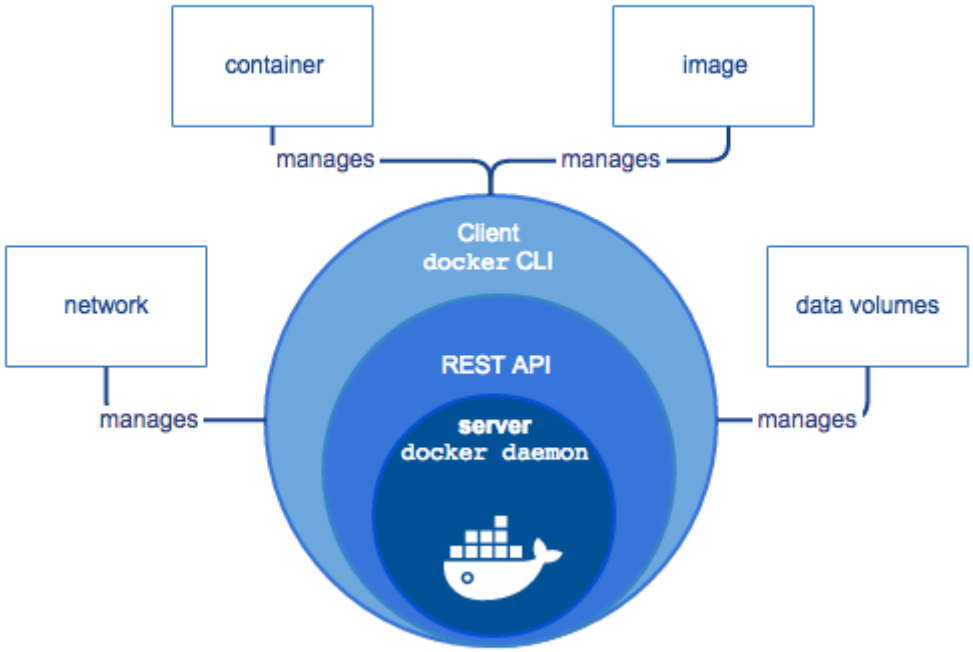
> *Docker on kehittäjille ja järjestelmänvalvojille tarkoitettu alusta sovellusten kehittämiseen, julkaisemiseen ja suorittamiseen. Docker mahdollistaa nopean sovellusten kasaamisen ja eliminoi koodinjulkaisun hankaluudet. Dockerin avulla saat ohjelmasi testattua ja julkaistua tuotantoon mahdollisimman nopeasti.* (Anand 2017.)

Docker nousi nopeasti suureen suosioon julkaistua konttiteknologiaa tarjoavan sovelluksen vuonna 2013 (Docker 2020c). Docker kehitti helposti suorituskykyisen, hallittavan sekä helposti julkaistavan Linux konttiteknologian ja julkisti sovelluksen avoimen lähdekoodin lisenssillä (Docker 2020d).

Yhtiö jatkoi sovelluksen kehittämistä julkisesti avoimen lähdekoodin harrastajien parissa ja vuonna 2015 Docker lahjoitti konttinäköistiedosto spesifikaation ja ajonaikaisen lähdekoodin Open Container Initiative (OCI) projektille vakiinnuttaakseen standardisointia kontti ekosysteemin kasvaessa ja kehittyessä. Vuonna 2017 Docker lahjoitti alan standardin ajonaikaisen konttiteknologian Cloud Native Computing Foundation (CNCF) projektille. (Docker 2020d.)

3.6 Docker Arkkitehtuuri

Konttien ja fyysisen laitteen käyttöjärjestelmän välissä on Docker Engine, docker komentokehote sekä asiakas-palvelinsovellus, joka koostuu taustalla jatkuvasti suoritettavasta palvelinohjelmistosta. Näiden lisäksi konttisovellukset käyttävät HTTP REST API-rajapintoja lähettääkseen käskyjä Docker taustaprosessille. Taustaprosessi ylläpitää konttien lisäksi muita Docker objekteja, kuten näköistiedostoja, tietoverkkoja sekä volyymeja (volume) (kuva 4). (Docker 2020e.)

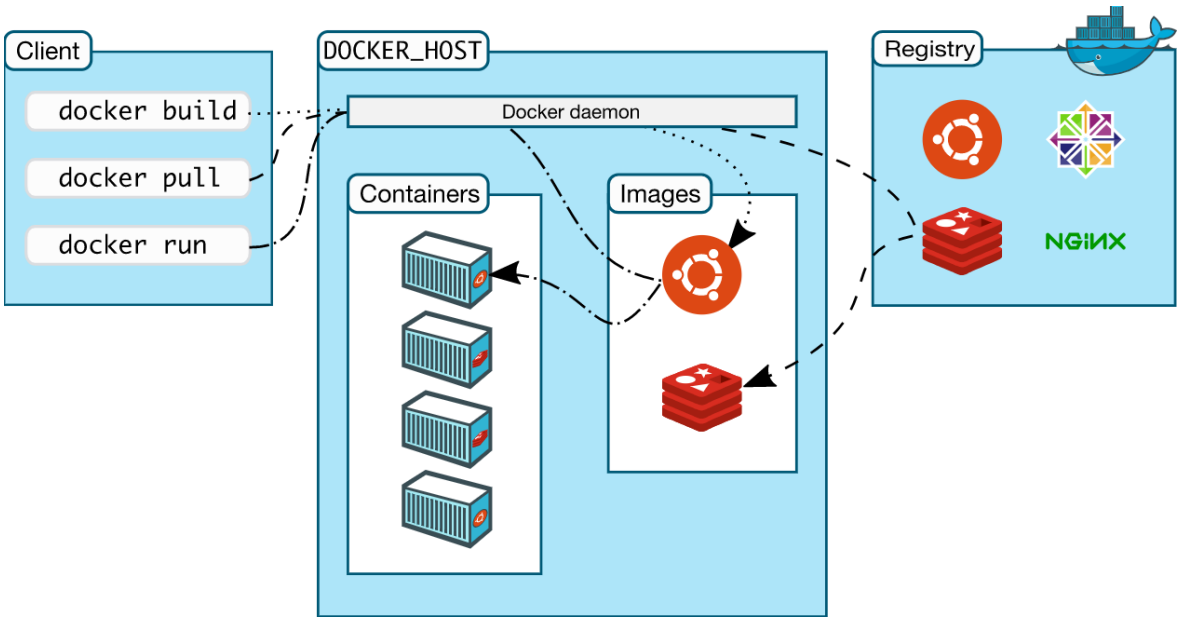


Kuva 4. Docker komponentit (Docker 2020e)

Docker näköistiedostot ovat konfigurointitiedostojen perusteella luotuja objekteja, joiden perusteella kontit rakennetaan. Näköistiedostot koostuvat tasoista, joissa määritellään kaikki sovelluksen suorittamiseen tarvittava, kuten sovelluskoodi, kirjastot sekä ympäristömuuttujat. (Rouse 2020a.)

Näköistiedostoja voi laajentaa tai yhdistellä ja rakentaa sovelluksen, joka koostuu useasta näköistiedostosta (kuva 5). Docker näköistiedostoja voi ladata julkisiin tai yksityisiin Docker tietolähteisiin (repositories). Yksi näistä on Dockerin tarjoama Docker Hub. Docker Hub on pilvipohjainen näköistiedostojen tietolähde, johon voi ladata julkisia tai yksityisiä näköistiedostoja. Palvelu sisältää sekä virallisia Dockerin luomia, että yksityisten henkilöiden luomia näköistiedostoja. Turvallisuuden lisäämiseksi näköistiedostot voidaan laittaa Docker Trusted Registry sovelluksen sisälle, joka tuo näköistiedostojen hallinta- ja pääsyn

hallinta toimintoja. Näköistiedostot muuttuvat lopulta konteiksi, kun ne suoritetaan Docker Engine sovelluksella. (Docker 2020f.)



Kuva 5. Docker arkkitehtuuri (Docker 2020f)

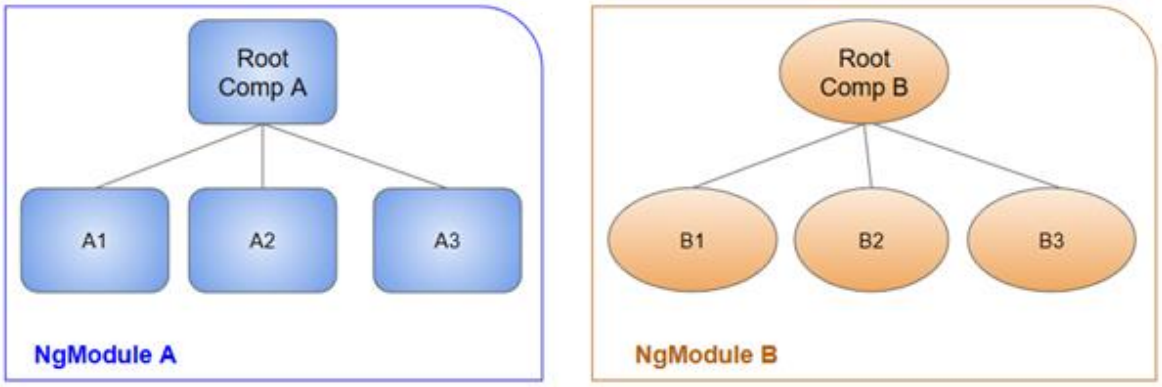
4 ANGULAR 2+

4.1 Yleisesti Angular-sovelluskehyksestä

Angular, kutsutaan usein myös nimellä Angular 2+ (M 2018), on sovelluskehys (framework) yhden sivun web-sovellusten luomista varten. Angular on rakennettu TypeScript-ohjelmointikielellä ja sillä voi luoda sovelluksia HTML- ja TypeScript -ohjelmointikielillä (Google 2020a).

4.2 Modulaarisuus

Angular sovellukset koostuvat moduuleista, joista käytetään nimetystä ”NgModuuli” (NgModule). Moduuli määrittää komponenttiryhmän joka vastaa tietystä sovelluksen toimialasta, työtavasta tai samankaltaisista ominaisuuksista. Komponenttiryhmät määrittelevät ryhmän näkyvyysalueen ja ne voivat koostua komponenttien lisäksi palveluntarjoajista ja muista kooditiedostoista (kuva 5). Moduulit voivat hyödyntää toisten moduulien ulospäin tarjoamia ominaisuuksia sekä tarjota ominaisuuksia muiden moduulien käytettäväksi. Isommat Angular applikaatiot koostuvat yleensä useasta moduulista hierarkisessa muodostelmassa, jossa ylimmällä tasolla on yksi moduuli. Moduulien hierarkista tasomäärää ei ole rajoitettu, mutta jokainen Angular applikaatio koostuu vähintään yhdestä moduulista. (Google 2020b.)



Kuva 5. Angular moduulin näkyvyysalue (Google 2020b)

4.2.1 Ominaisuusmoduulit

Ominaisuusmoduulit ovat sovelluksen rakennetta varten ja ne koostuvat tiettyä ominaisuutta koskevasta kooditiedostoista. Tiettyihin ominaisuuksiin rajatut moduulit tekevät sovelluksesta helpommin hallittavan. Angular moduulit luokitellaan yleensä viiteen eri ryhmään:

- toimialueominaisuusmoduulit
- reittiominaisuusmoduulit
- reititysmoduulit
- palvelumoduulit
- pienoisohjelmamoduulit (Widget modules). (Google 2020c.)

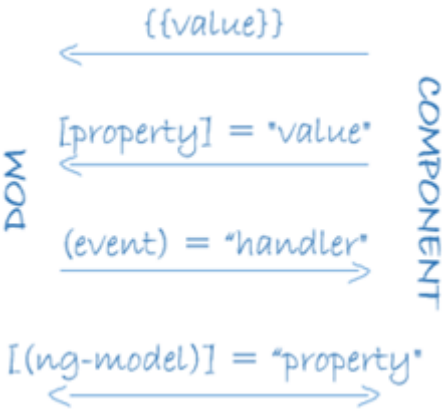
Ominaisuusmoduulit eivät ole pakollisia ja sovelluksen voi rakentaa pelkällä yhdellä päätason moduulilla (root module). Ominaisuuksien pilkkominen moduuleiksi on kuitenkin hyvä käytäntö, joka helpottaa kehittäjien ja tiimien yhteistyötä. (Google 2020c.)

4.2.2 Komponentit

Angular komponentit ovat TypeScript luokkia, jotka vastaavat yksittäisestä sivun näkymän osiosta. Luokka on vuorovaikutuksessa näkymän kanssa tarjoamansa API: n avulla, joka koostuu määritteistä ja metodeista. Angular huolehtii komponenttien luomisesta, päivittämisestä ja tuhoamisesta dynaamisesti sovellusta navigoitaessa, joka tarkoittaa käytännössä sitä, että vain tarpeelliset komponentit ovat aktiivisia kullakin hetkellä. (Google 2020d.). Komponenteilla on Angularin hallinnoima elinkaari ja se myös huolehtii komponenttien lisäämisestä, päivittämisestä ja poistamisesta sivun dokumenttioliomallista, joka tekee HTML sivusta interaktiivisen (Google 2020e).

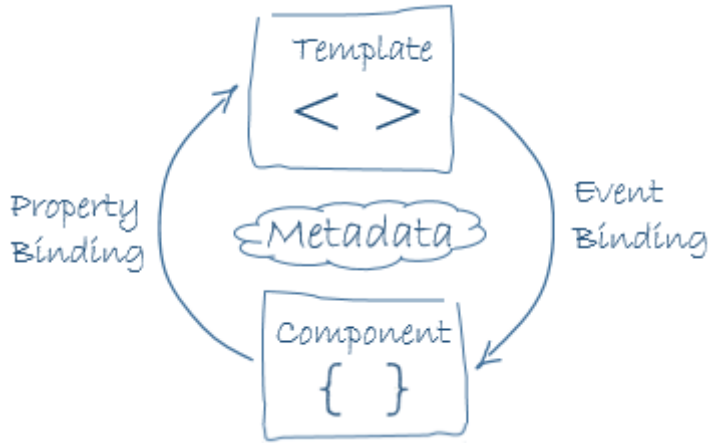
Komponenttien tarjoamat näkymät ovat HTML tiedoston tapaisia mallitiedostoja, jotka kertovat missä muodossa Angularin tulee renderöidä ne. Tavallisen HTML rakenteen lisäksi mallit voivat käyttää tietojen sidontaa (data binding) yhdistääkseen sovelluksen ja dokumenttioliomallin datan, putkia (pipes) tietojen transformointiin ennen renderöintiä sekä direktiivejä, jotka kontrolloivat mitä tietoja näytetään. Tietojen sidontaa on neljää erityyppistä (kuva 6):

- tapahtumasidos
- yksisuuntainen sidos
- kaksisuuntainen sidos
- interpolointi. (Google 2020d.)



Kuva 6. Tietojen sidontatyypit (Google 2020d)

Tietojen sidontatyyppi määrittää tiedon kulkusuunnan dokumenttioliomallista komponenttiin. Kaksisuuntainen sidos on yksi hyödyllisimmistä sidostyypeistä, se yhdistää tapahtumasidoksen sekä yksisuuntaisen sidoksen yhdellä notaatiolla. Komponentti tarjoaa ensin arvon mallille ja käyttäjän suorittamat tapahtumat saavat tiedon kulkemaan takaisin komponenttiin. Tämä saa arvon päivittymään komponentin sisällä ja se tarjotaan uudestaan malliin käyttäjän päivittämänä (kuva 6). Putket mahdollistavat komponentin tarjoaman tiedon transformoinnin syöttämällä komponentin tarjoaman tiedon putkifunktion läpi ja palauttamalla muunnetun arvon. Putkifunktion palauttaman arvon voi ketjuttaa seuraavaan putkifunktioon. (Google 2020d.)

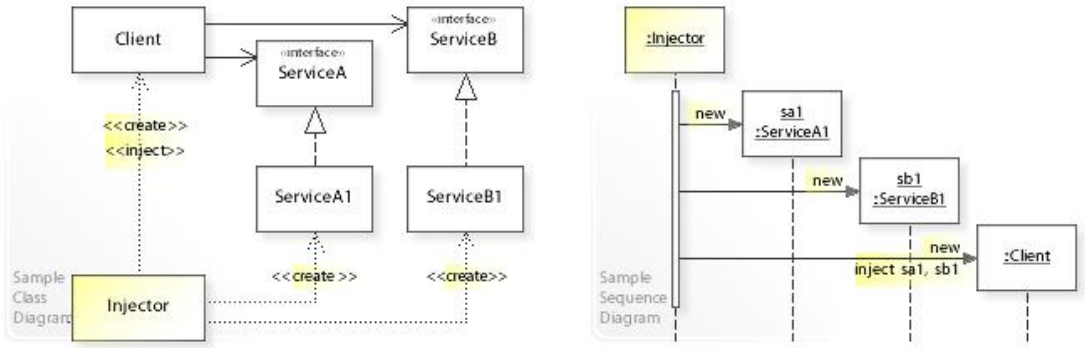


Kuva 7. Kaksisuuntainen sidos (Google 2020d)

Mallien näkymiä voidaan kontrolloida dynaamisesti direktiiveillä, joita on kahta erityyppistä, rakenteellinen direktiivi ja määrite direktiivi. Rakenteelliset direktiivit käyttävät sovelluslogiikkaa renderöintiin ja ne muokkaavat näkymän ulkoasua lisäämällä, poistamalla tai korvaamalla dokumenttioliomallin elementtejä. Rakenteellisten direktiivien avulla voidaan muun muassa olla renderöimättä sisältöä, jota sovelluksen käyttäjällä ei ole oikeus nähdä. Määrite direktiivit muokkaavat olemassa olevien HTML elementtien toimintaa usein hyödyntämällä komponentin tarjoamaa tietoa. (Google 2020d.)

4.3 Palvelut ja riippuvuusinjektio (dependency injection)

Angular erottaa komponentit palveluista modulaarisuuden ja uudelleenkäytettävyyden lisäämiseksi. Palvelu on laaja termi, joka sisällyttää minkä tahansa sovelluksen tarvitseman arvon, funktion tai ominaisuuden. Tyypillisesti palvelut ovat luokkia jotka käsittelevät hyvin pieniä toiminta-alueita. Komponentin tulisi vastata pelkästään käyttäjäkokemuksen tarjoamisesta, ja sen saavuttamiseksi komponentti voi käyttää palvelun tarjoamia ominaisuuksia. Kun ominaisuudet on eritelty palveluiksi, useat komponentit voivat käyttää samoja palveluita. (Google 2020f.). Palveluiden käyttämiseen Angular hyödyntää riippuvuusinjektio suunnittelumallia, jota varten Angular sisältää oman sovelluskehyksen (Google 2020g). Komponentti pyytää riippuvuuksia ulkoiselta lähteeltä, jolloin injektori joko luo palvelun tai käyttää jo olemassa olevaa instanssia, ja palauttaa sen komponentille (kuva 8). Kun komponentti on saanut kaikki pyytämänsä palvelut injektorilta, Angular voi luoda objektin komponentista käyttäen näitä palveluita. (Google 2020f.)

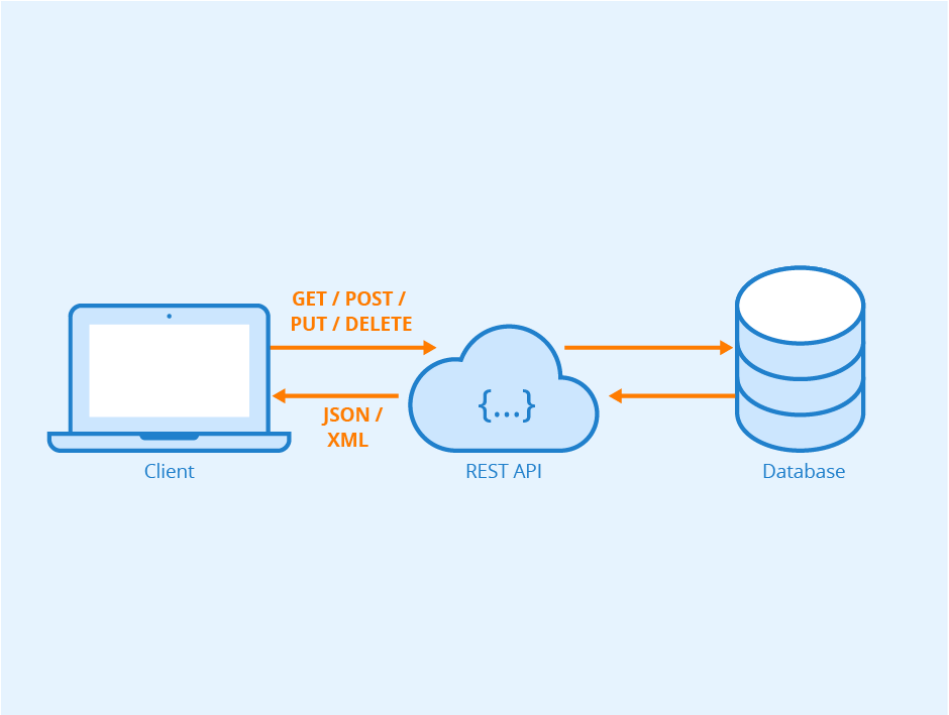


Kuva 8. Riippuvuusinjektio (Vanderjoe 2016)

5 REST

5.1 Yleisesti REST-arkkitehtuurimallista

REST (Representational State Transfer) on ohjelmistoarkkitehtuurityyli verkkopalveluiden luomista varten, joka määrittää viisi ehtoa palvelimelle asiakkaan pyyntöjen prosessointiin ja vastaamiseen (Richardson & Ruby 2007). Verkkopalveluita, jotka noudattavat näitä ehtoja, kutsutaan RESTful verkkopalveluiksi. RESTful verkkopalvelut tarjoavat ennalta määrättyjä yhtenäisiä tilattomia operaatioita resurssien hakemista ja manipulointia varten (kuva 8). (W3 2004.)

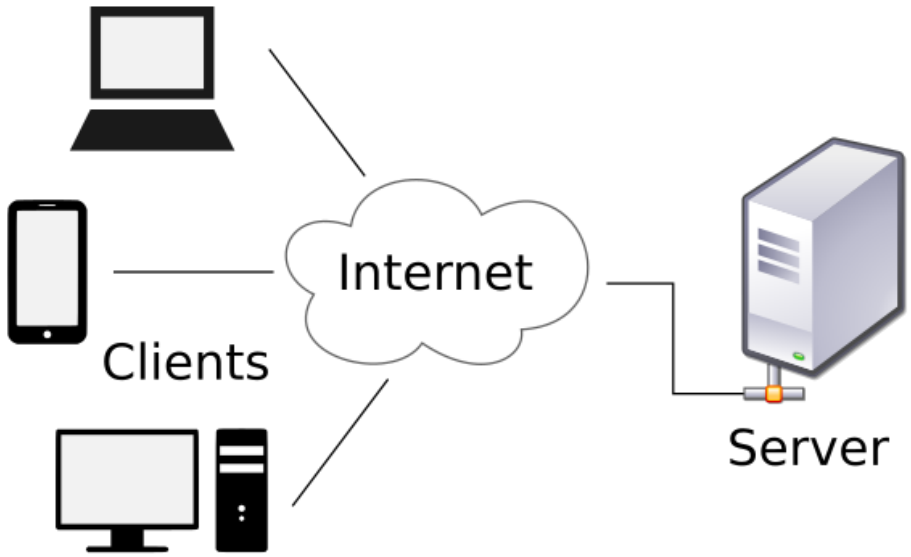


Kuva 8. REST API (Seobility 2020a)

5.2 Asiakas-palvelin (Client-server)

Asiakas-palvelin (Client-server) -malli on arkkitehtuurimalli, jonka mukaan ohjelmiston osat jaetaan palvelimiin, jotka tarjoavat resursseja asiakkaille. Palvelimia ja asiakkaita on oltava vähintään kaksi, toinen hoitaa tiedon säilyttämisen ja toinen sen esittämisen. Useimmat tietokantasovellukset ovat asiakas-palvelin -sovelluksia, sillä niiden yleinen tarkoitus on tarjota samaa dataa usealle eri käyttäjälle. Useat verkkosovellukset, kuten web ja sähköposti, ovat kaksitasoisia asiakas-palvelin –sovelluksia. Kaksitasoisella sovelluksella tarkoitetaan ohjelmiston jakamista kahteen osaan, palvelimeen ja asiakkaaseen. Esim. Käyttäjän navigoitaessa verkkosivulle, käyttäjä saa palvelimelta verkkosivun sisäl-

lön, jonka käyttäjän päätelaite osaa tulkita ja esittää oikeassa muodossa (kuva 9). (Reese 2000, 128–129.)

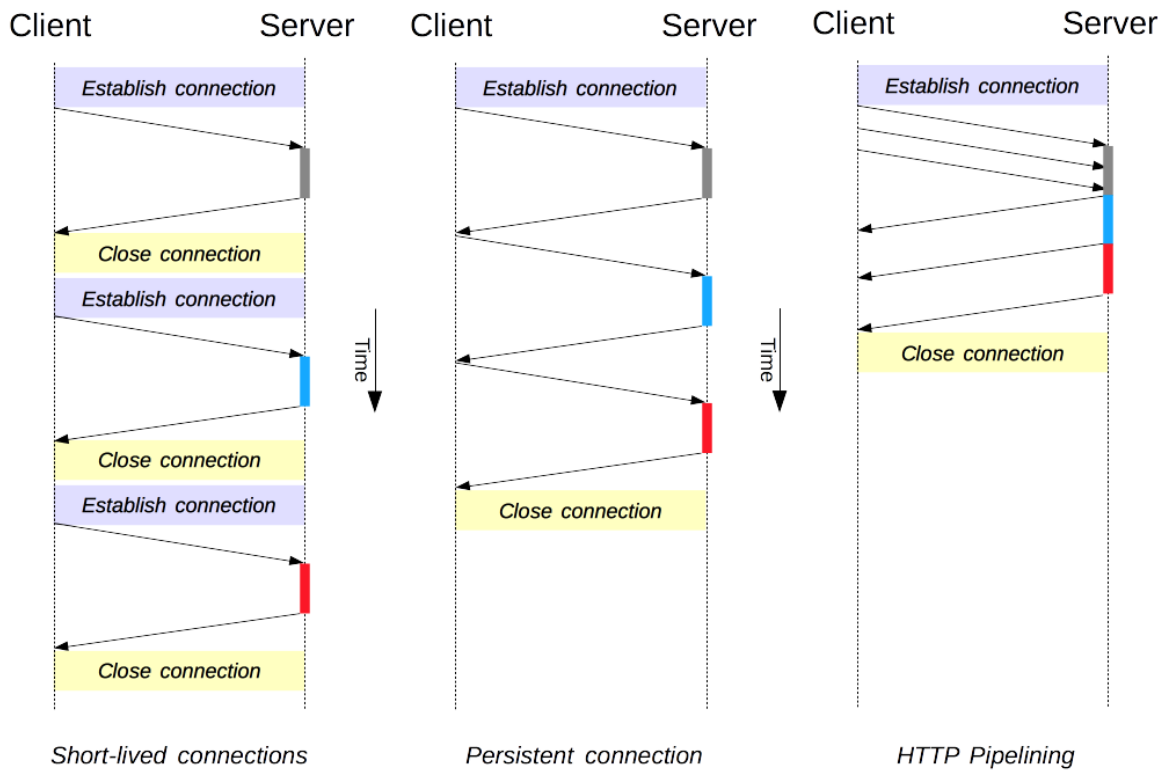


Kuva 9. Asiakas-palvelin malli (Vignoni David & Calimo 2020)

Asiakas-palvelin –mallin tarkoituksena on ohjelmiston toimintojen jakaminen erillisiin osiin, joka pitää komponentit yksinkertaisina ja koko sovelluksen skaalautuvana. Kun käyttöliittymä vastaa ainoastaan tiedon esittämismuodosta, käyttöliittymää on helpompi lähteä muuttamaan tai siirtämään toiselle alustalle. Komponenttien jakaminen mahdollistaa niiden riippumattoman kehittämisen. (Fielding 2000.)

5.3 Tilaton protokolla

Tilattomalla protokollalla tarkoitetaan protokollaa, jossa vastaaja, usein palvelin, ei ylläpidä sessioinformaatiota. Jokaiseen pyyntöön tulee pystyä vastaamaan pyynnön mukana tulleella informaatiolla, ilman edellisten pyyntöjen sisältöä. Tilattoman protokollan sovellukset ovat sopivia laajan käytön (high volume) sovelluksissa, kun palvelimen ei tarvitse käyttää resursseja sessioinformaation säilyttämiseen. IP- ja HTTP –protokollat (kuva 10) ovat esimerkkejä tilattomasta protokollasta laajassa käytössä. (IETF 2014, 11.)



Kuva 10. HTTP/1.1 yhdistämismallit (Mozilla 2019)

RESTful –mallin sovelluksissa palvelin voi esim. tallettaa sessioinformaation tietokantaan hetkellistä todennusta varten, mutta varsinaista sovelluksen tilaa ylläpidetään asiakkaan puolella. Pyynnön lähetettyä, sovellus on siirtymätilassa ja vastauksen saatuaan sovellus on uudessa tilassa. Kun sovellus on valmis siirtymään toiseen tilaan lähetettyä pyynnön, sovellus on siirtymätilassa. Vastauksen saatua sovellus on uudessa tilassa ja vastaus sisältää metodit uuteen tilaan siirtymistä varten. (Fielding 2013.)

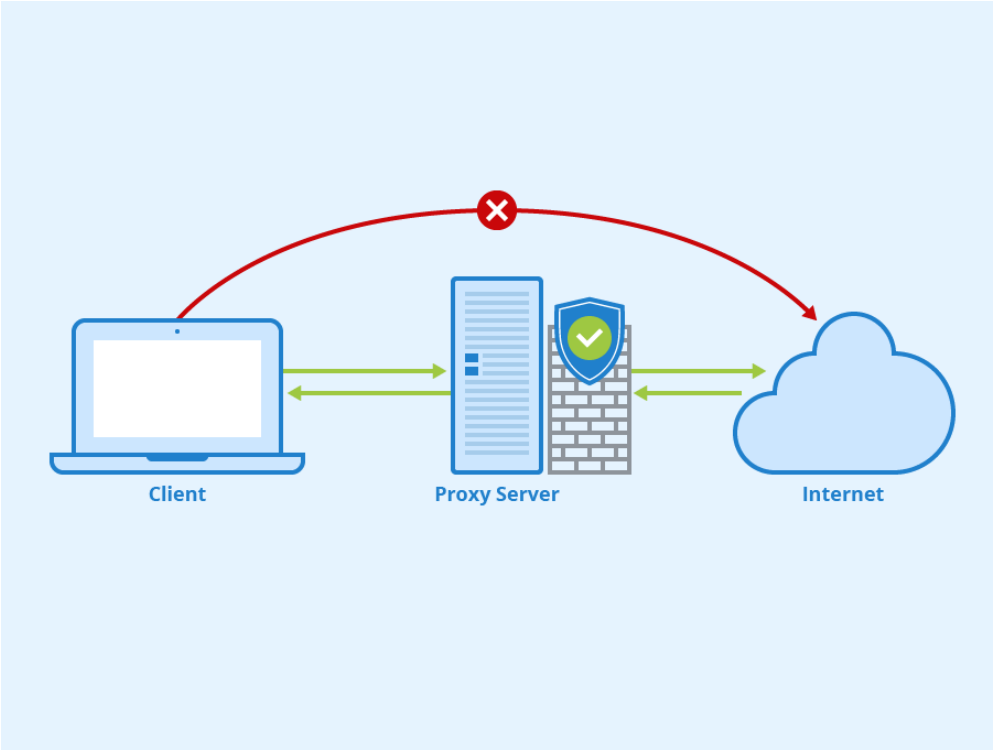
5.4 Välimuisti

Verkkovälimuisti (Web cache) voi tallentaa sen läpi kulkevia verkkodokumentteja, kuten verkkosivuja tai kuvia, ja tarjota niitä käyttäjälle vähentääkseen palvelimen kuormaa (Shinder 2008). HTTP-pyyntöjen tulee joko implisiittisesti tai eksplisiittisesti määritellä voiko niitä tallentaa välimuistiin, vanhentuneiden tai virheellisten tietojen tarjoamisen välttämiseksi. Hyvin toteutettu sekä ylläpidetty välimuistipalvelin vähentää asiakas-palvelin -interaktiota, ja se tehostaa skaalautuvuutta ja tehokkuutta. (Lange 2016.)

5.5 Kerrosrakenteinen järjestelmä

Kerrosrakenteisessa järjestelmässä komponentit on ryhmitelty tasoihin hierarkkisesti niin, että alemman tason komponentit tarjoavat toimintoja ja palveluita, jotka tukevat ylempien tasojen toimintoja ja palveluita. Jatkuvasti laajentuvia ja monimutkaistuvia järjestelmiä

voidaan rakentaa lisäämällä tai korvaamalla tasoja. (General Services Administration 1996.). Asiakkaan ei tulisi olla tietoinen onko se yhdistänyt suoraan loppupalvelimeen vai välikerroksen palvelimeen (proxy). Välipalvelimen tai kuormantasaajan lisääminen tai poistaminen asiakas-palvelin –ketjun välistä ei pitäisi vaatia muutoksia kummankaan puolen ohjelmistoissa. Jokaisella välipalvelimella tulee olla oma logiikkansa, esim. palomuurin väliin lisäämisen (kuva 11) ei pitäisi sekoittaa liiketoiminta- ja turvallisuuslogiikkaa keskenään. (Lange 2016.)



Kuva 11. Välipalvelin asiakkaan ja palvelimen välissä (Seobility 2020b)

5.6 Yhtenäinen rajapinta

Yhtenäinen rajapinta määrittää neljä ehtoa, jotka yksinkertaistavat ja irtaannuttavat arkkitehtuuria, jotta jokainen osa-alue voisi kehittyä itsenäisesti muista riippumattomana. Näitä ovat

- resurssien identifiointi pyynnöissä
- resurssien manipulointi
- itseään kuvailevat viestit
- hypermedia sovelluksen tilakoneena (Hypermedia as the Engine of Application State).

Yksittäisillä resursseilla tulee olla uniikki identifioiva tunnus, jotta juuri tiettyyn resurssiin voidaan viitata. Tunnuksen avulla tietty resurssi voidaan etsiä kaikkien resurssien joukosta ja sitä voidaan manipuloida, tai poistaa resurssi kokonaan. Jos asiakkaalla on resurssin tiedot, asiakkaalla tulisi olla tarpeeksi informaatiota sen manipulointia varten. Jokaisen viestin täytyy olla itseään kuvaava viestin prosessointia varten. Viestien tulisi myös sisältää kaikki resurssien mahdollistamat toiminnot, jolloin asiakassovelluksen ei tarvitse olla tietoinen sovelluksen rakenteesta. Esim. Pankkitilin saldo haetaan HTTP-pyynnöllä käyttäen identifioivaa tunnusta (kuva 12). Vastaus sisältää eri toimintoja osoitteen muodossa riippuen saldon määrästä. Nämä osoitteet puolestaan tarjoavat seuraavan operaation edellisen resurssin manipulointia varten. (Fielding 2000.)

```
GET /account/12345 HTTP/1.1

HTTP/1.1 200 OK
<?xml version="1.0"?>
<account>
    <account_number>12345</account_number>
    <balance currency="usd">100.00</balance>
    <link rel="deposit" href="/account/12345/deposit" />
    <link rel="withdraw" href="/account/12345/withdraw" />
    <link rel="transfer" href="/account/12345/transfer" />
    <link rel="close" href="/account/12345/close" />
</account>
```

Kuva 12. Pankkitilipyynnön tarjoamat operaatiot (REST CookBook 2020)

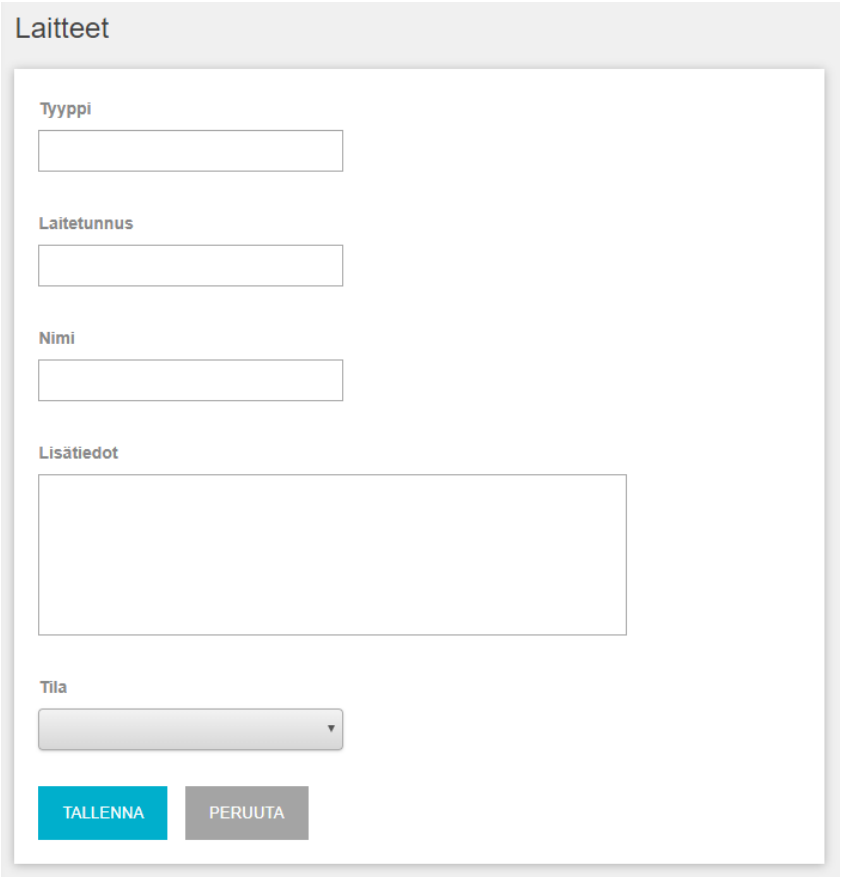
6 OHJELMISTOKEHITYS

6.1 Käyttöliittymä

Laitehallinnan käyttöliittymästä pyrittiin tekemään mahdollisimman yksinkertainen ja helppokäyttöinen, uhraamatta toiminnallisuutta. Käyttöliittymällä haluttiin tuoda kaikentyyppiset laitteet yhteen paikkaan, josta niitä olisi vaivatonta hallinnoida. Lomakkeet ovat nopeasti täytettäviä, sillä tiedot pohjautuvat jo muualla syötettyihin tietoihin, josta käyttäjä voi helposti valita oikean vaihtoehdon. Laitehallinnan sivuilla voi siis lisätä, muokata ja tarkastella laitteita. Ensimmäisessä vaiheessa otetaan käyttöön tarvittava minimitoiminnallisuus ja kehitystyötä jatketaan perustuen saatuihin kokemuksiin.

6.1.1 Laitepohjien luominen

Laitteet lisätään järjestelmään laitemallien pohjalta, eli ensin luodaan malli tietynlaiselle laitteelle. Mallilomakkeeseen syötetään laitteen tyyppi, laitetunnus, nimi, mahdolliset lisätiedot sekä oletus käyttöönottotila (kuva 13).



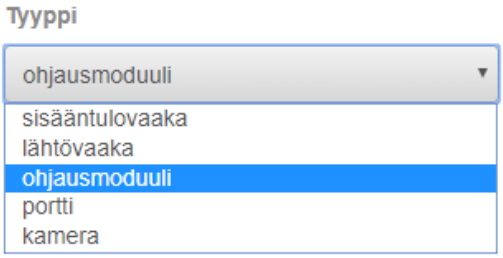
Kuva 13. Laitepohjan luontilomake

Laitepohjan luotua, laitepohjalle voidaan syöttää parametreja. Parametreja voidaan hyödyntää muun muassa laitteen käynnistämisessä. Laiteparametrit ovat alkuvaiheessa joko string tai JSON -tyyppisiä arvoja (kuva 14).

Kuva 14. Laitepohjan parametrit

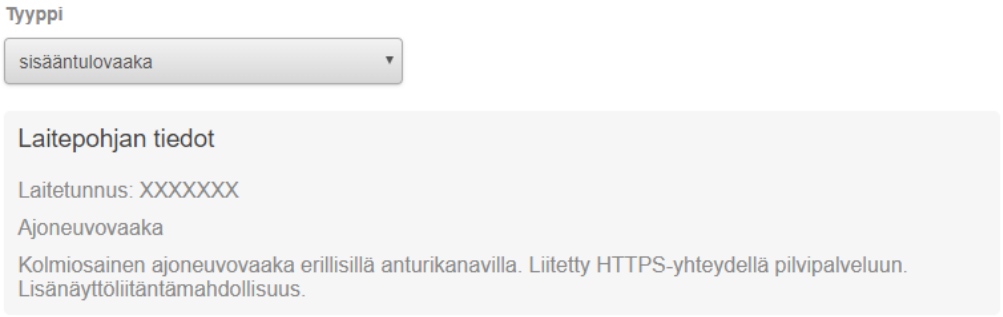
6.1.2 Laitteiden lisääminen

Kun järjestelmään on luotu vähintään yksi laitepohja, voidaan lisätä uusi laite. Laitteet luodaan samankaltaisen lomakkeen kautta, kuten laitepohjat, mutta täytettävänä on hieman eri kenttiä. Laitetyyppi valitaan listasta, jossa näkyy kaikki yritykselle saatavissa olevat laitepohjat (kuva 14).



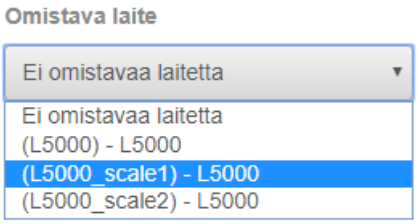
Kuva 14. Laitetyypin valinta

Valittua laitteen tyypin, käyttäjä näkee kyseisen pohjan perustiedot (kuva 15), joka voi antaa mallia laitteen tietoja täyttäessä. Luotava laite myös perii automaattisesti laitepohjan parametrit. Laitepohjien tarkoitus oli nopeuttaa laitteiden luontia.



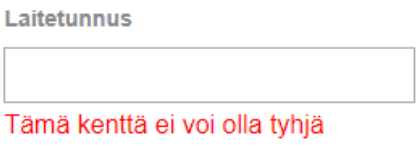
Kuva 15. Laitepohjan tiedot

Laitteet ovat joko päälaitteita tai näiden alilaitteita, laitteita voi siis järjestellä hierarkkisesti. Jos kyseessä on alilaite, valitaan listasta omistava laite (kuva 16), joka näyttää kaikki yrityksen laitteet. Tästä listasta on kuitenkin suodatettu pois kaikki virheelliset valinnat, esim. jos laitteella on jo alilaite, ei kyseistä laitetta voi merkata omistavaksi laitteeksi.



Kuva 16. Omistavan laitteen valinta

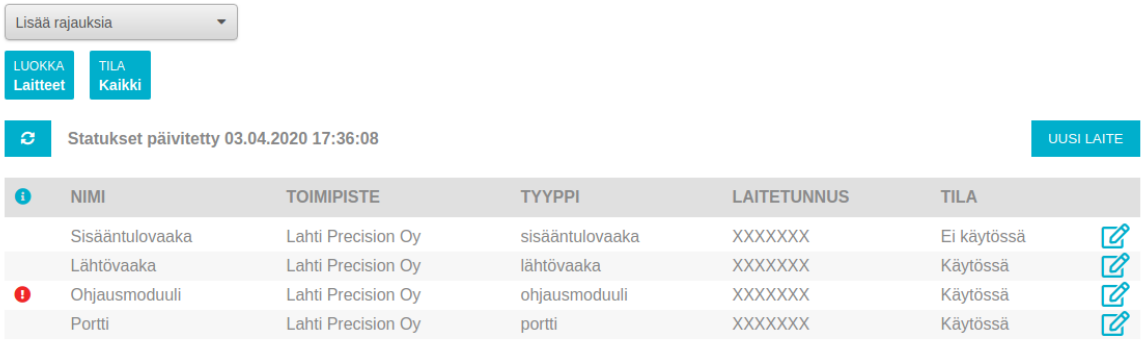
Laitteelle voidaan myös määritellä toimipiste listasta, jossa näkyy kaikki yrityksen toimipisteet, tai laitteelle ei tarvitse määritellä toimipistettä ollenkaan. Loput lomakkeen kentistä ovat tavallisia tekstikenttiä, johon käyttäjä saa syöttää vapaasti sisältöä. Osa kentistä on pakollisia, joten ennen laitteen tallentamista, kaikille näille kentille suoritetaan validointi ja niistä ilmoitetaan käyttäjälle (kuva 17). Validointivirheet katoavat kun kentät on täytetty oikein.



Kuva 17. Validointivirheen näyttäminen

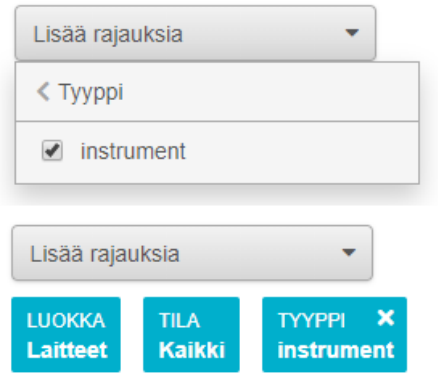
6.1.3 Laitteiden tarkastelu

Kaikki yrityksen laitteet koottiin taulukkomuotoiseen listaan, jossa on sarakkeet laitteen statukselle, nimelle, toimipisteelle, tyypille, laitetunnukselle ja tilalle (kuva 18).



Kuva 18. Laitelista

Laitelistaan saattaa kertyä suuri määrä laitteita, joten listaan toteutettiin rajausmenetelmiä. Listattavia laitteita voi suodattaa luokan, tilan, tyypin sekä toimipisteen mukaan. Oletuksena rajauksista on valittu luokkana laitteet ja tilaan kaikki. Luokkavalintaa vaihtamalla on mahdollista näyttää vain laitepohjat ja tilavalintaa muuttamalla näytetään joko kaikki, käytössä olevat tai ei käytössä olevat laitteet. Tyyppi- ja toimipistevalinta ovat monivalintoja kaikista yrityksen laitetyypeistä ja toimipisteistä. Käyttäjä voisi siis listata esim. pelkästään autovaa'at ja portinohjaimet, jotka sijaitsevat joko Kuopiossa tai Turussa.



Kuva 19. Laitelistan rajaus

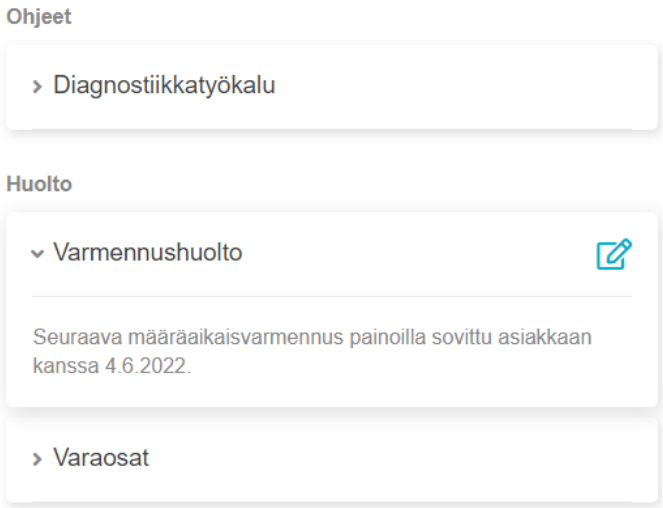
Jokaisen laiterivin kohdalla on muokkauspainike (kuva 18), josta pääsee nopeasti muokkaamaan laitteen tietoja.

6.1.4 Laitteiden muokkaaminen

Painettua laitteen muokkauspainiketta, käyttäjä ohjataan melko samannäköiselle lomakkeelle kuin laitetta luodessa. Laitteen tiedot on täytetty sivulle valmiiksi ja sivulla on muutama uusi kenttä. Laitteen perimiä parametreja voi muokata, niin ettei muokkauksilla ole vaikutusta laitepohjan parametreihin. Parametrit listataan taulukkomuodossa (kuva 18), kuten laitteet, sarakkeina parametrin nimi, arvo, sekä luonti- ja päivityspäivämäärä. Para-

metreja voi muokata nopeasti suoraan listasta, tai klikkaamalla rivin muokkauspainiketta, joka vie omalle sivulle. Suurempien JSON parametrien muokkaaminen on huomattavasti helpompaa omalla sivulla, jossa on JSON arvojen muokkausta varten tarkoitettu editori. Editori osaa myös huomauttaa syntaksivirheistä, ja arvot validoidaan ennen niiden tallentamista.

Muokkaussivulta laitteelle voi myös luoda muistiinpanoja, jotka ovat HTML sisältöä. Muistiinpanolle annetaan nimi ja tyyppi, ja laitteen sivulla ne ryhmitellään tyypin mukaan (kuva 20). Muistiinpanoihin voi kirjoittaa muun muassa huolto-ohjeita tai muuta oleellista tietoa laitteesta.



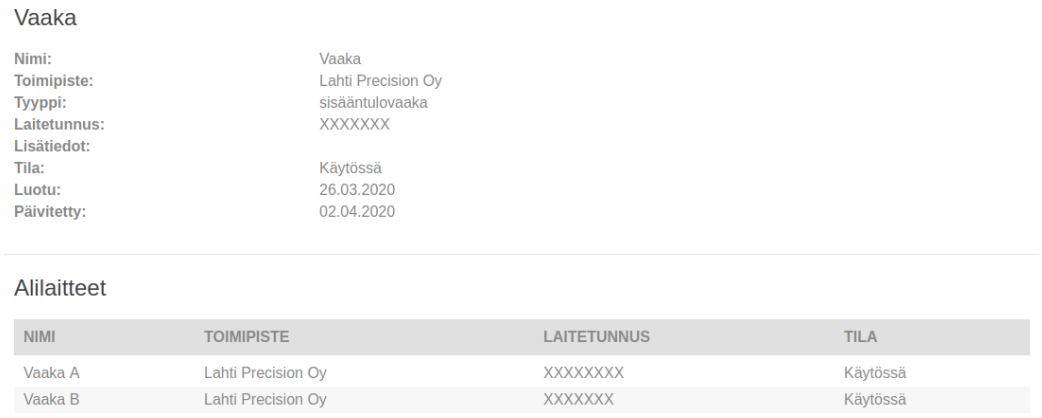
Kuva 20. Laitteen muistiinpanot

6.2 Tietojen tallentaminen ja hyödyntäminen

Laitteet ilmoittavat tilastaan jatkuvasti ja tämä tieto tallennetaan tietovarastoon (database). Kun tieto kaikkien laitteiden kunnosta on selvillä, se voidaan välittää käyttäjälle yhdellä näkymällä, josta selviää nopeasti kaikkien laitteiden tila. Yhdenkin laitteen epäkunto on kriittinen tieto, siksi tieto kaikkien laitteiden tiloista piti tuoda selkeästi esille.

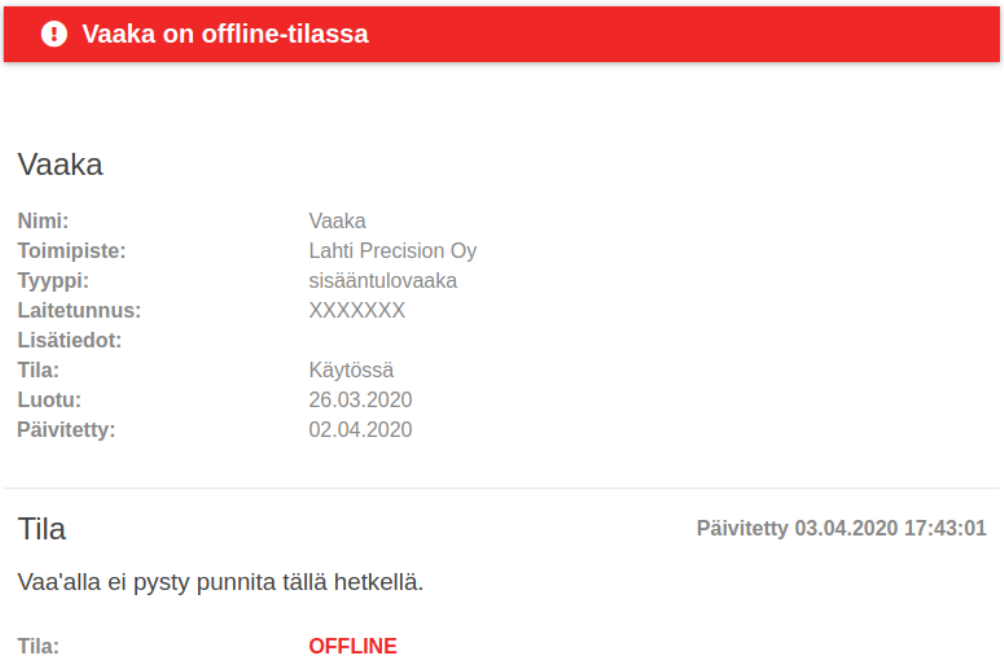
6.2.1 Yksittäisen laitteen tiedot

Jokaiselle laitteelle on oma yksityiskohtaisempi näkymä, jossa näytetään kaikki laitteen perustiedot, parametrit, muistiinpanot, tilatiedot sekä kyseisen laitteen mahdolliset alilaitteet. Jos laitteella on alilaitteita, sivulla on taulukko seuraavan tason laitteista (kuva 21). Nämä taulukon rivit toimivat myös linkkeinä kyseisiin laitteisiin, jotta tasojen välinen navigointi olisi helpompaa. Ylemmän tason laitteeseen pääsee takaisin klikkaamalla otsikon viereistä linkkiä, jossa näytetään laitteen nimi johon ollaan palaamassa.



Kuva 21. Laitteen alilaitteet

Laitesivulle saavuttua, kyseisen laitteen tilatietoja haetaan jatkuvasti. Esim. autovaa'an tapauksessa, sivulla näytetään muun muassa vaa'an status, eli onko vaaka valmiina punnitukseen vai ei, ja painolukema. Vaa'asta saadaan myös kaikki sen lähettämät vikaviestit sekä tieto sen kunnosta. Jos laite on virhetilassa, siitä ilmoitetaan huomiota herättävällä viestillä (kuva 22), eikä vaa'alla voida punnita silloin. Vaaka voi myös olla vastaamatta viesteihin, jolloin se on oletettavasti offline-tilassa. Tästä ilmoitetaan samalla tavalla kuin virhetilasta, tosin eri viestillä.



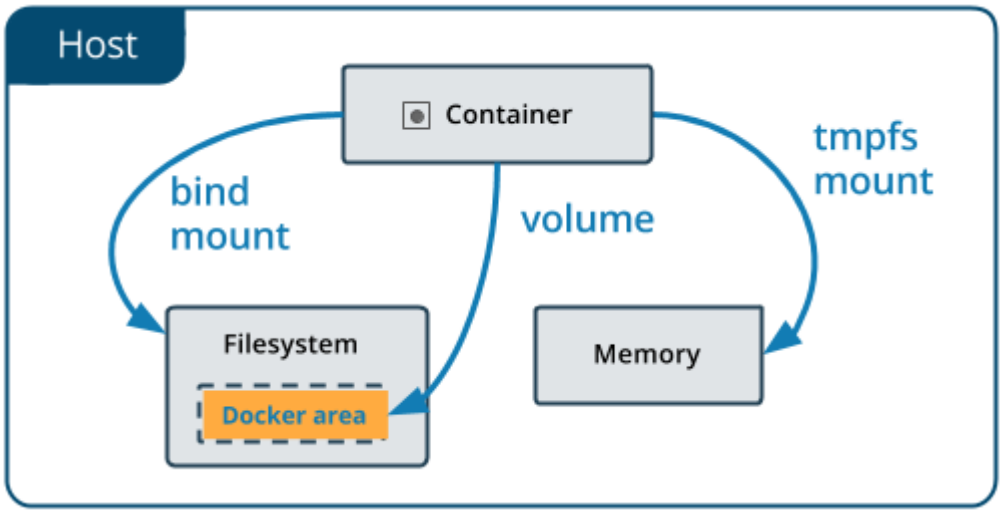
Kuva 22. Virheilmoitus laitteen epäkunnosta

6.2.2 Laitteiden tilatiedot

Myös kaikkien yrityksen laitteiden kuntoa voidaan seurata yhdeltä sivulta. Laitelistassa näytetään jokaisen laitteen kollektiivinen kuntotila, eli voiko vaakaa käyttää punnitusta varten (kuva 18).

6.3 Prosessin kehitys

Laitehallinnan sivutuotteena toteutui myös ohjelmointiprosessin kokonaisuuden kehitystä. Sovellus koostuu useasta osiosta, jotka on jaettu Docker kontteihin ja ennen kehittämistä kaikki kontit täytyy rakentaa ja laittaa käyntiin. Kontin rakennusvaiheessa käydään läpi ja ladataan kaikki kontin riippuvuudet. Aikaisemmin, jos yksittäiseen sovellukseen tehtiin muutoksia, jouduttiin koko kontti rakentamaan uudestaan, joka tarkoitti käytännössä jatkuvasti samojen pakettien lataamista ja asentamista. Tämä ongelma ratkaistiin Docker volyymien avulla, joiden avulla konttien generoima data saadaan pidettyä olemassa myös kontin suorituksen jälkeen. Tämä mahdollisti asennettavien pakettien tallettamisen volyymeihin, jolloin kontti osasi hakea pakettien datan suoraan volyymista sen sijaan, että ne tarvitsi ladata aina ennen kontin suorittamista.



Kuva 8. Docker volyymit (Docker 2020)

Sama volyymimenetelmä mahdollisti nopeamman muutosten päivityksen, sillä kehittäjän tiedostorakenne voitiin linkittää kontin sisään volyymina, jolloin muutokset ohjelmistoon päivittyivät saman tien konttiin (kuva 6). Tätä edistettiin vielä lisäyksellä, joka osasi kuunnella tiedostorakenteen muutoksia ja koodi voitiin automaattisesti kääntää muutosten jälkeen, joka nopeutti kehitysvaihetta huomattavasti. Käytännössä heti tiedoston tallennettua, tiedosto muuttui myös kontin sisällä tiedostorakenteen linkityksen avulla, jolloin oh-

jelma osasi käynnistää käännöksen itsestään ja muutokset sovelluksessa saatiin näkyviin saman tien.

7 TOTEUTUSPROSESSI

7.1 Konseptointi ja tavoitteet

Fyysiset laitteet haluttiin pilveen käyttöliittymästä hallittavaksi tietokannasta hallinnan sijaan. Käyttöliittymään haluttiin täysi kontrolli kaikista laitteista, jolloin laitehallinnasta tulisi keskeinen työkalu koko henkilöstön käyttöön. Laitehallinnasta haluttiin myös tehdä keskeinen paikka kaikelle laitteen tilatiedoille, eli esim. laitteen kunnolle ja sensoritiedoille.

7.2 Lyhyen aikavälin tavoite

Ensimmäisiä tavoitteita laitehallinnan suhteen oli sen kehittäminen mScales-kehitystiimin käytettäväksi työn helpottamiseksi. Laitteita pystyisi lisäämään mScales palveluun paljon helpommin ja nopeammin käyttöliittymästä kuin tekemällä muutoksia suoraan tietokantaan. mScales kehittäjät toimivat usein laitteiden kanssa ja vaikka heidän täytyykin päästä hallitsemaan niitä, kun tiedot laitteista olivat hankalasti saatavissa, joutuivat he useasti asiakaspalvelun rooliin, tai ainakin välikädeksi. Asiakaspalveluun otettiin yhteyttä laitteen kunnosta tai muista ongelmista, eikä asiakaspalveluvastaavalla ollut keinoa päästä käsiksi laitteen tietoihin, joten ongelma siirtyi kehittäjälle. Siksi laitehallintaa tarvittiin kehittäjän käytettäväksi sekä työn siirtämiseen oikealle vastaavalle. Asiakaspalveluvastaava voisi käydä läpi listaa yrityksen laitteista ja nähdä, missä kunnossa ne ovat, tai katsoa laitteen lokitietoja ja nähdä sieltä mikä voisi olla vikana.

Vaa’at ja punnitusjärjestelmät voivat toimia jopa useita kymmeniä vuosia, kunhan niitä huolletaan säännöllisesti. Ajantasainen punnituselektroniikka ja suoraan pilveen liitetty vaaka mahdollistaa laitteen etäkäytön ja se tekee huoltamisesta ja virheiden selvittämisestä yksinkertaisemman prosessin kaikille osapuolille. Laitteen etähuolto voidaan aloittaa saman tien toimistolta, ja tilanteesta riippuen jopa korjata ilman erillisiä kenttätoimenpiteitä. Vanhan mallisten laitteiden, jotka eivät ole verkkoon kytkettyjä, huoltaminen ja vianselvittäminen ovat hitaita ja kalliita prosesseja. Ensiksi kentälle, joka voi olla hyvinkin kaukana, täytyy saada oikea henkilö tutkimaan ongelmaa oikeilla työkaluilla. Mahdollisesta viasta on vaikeaa saada selvää, jos sen raportoinut henkilö ei tunne laitetta hyvin. Ongelma voi olla hyvinkin yksinkertainen tekninen syy, jota varten ei tarvitse alkaa huoltamaan laitetta vaan ongelma voidaan korjata käsittelemällä vaa’an ohjelmistoa. Tällaisissa tapauksissa laitteeseen etäyhdistäminen on selvästi järkevämpi ratkaisu, kun tuhlata resursseja ja molempien osapuolien aikaa matkustamiseen. Nopeilla huolloilla vältytään pidemmiltä tuotannon katkoksilta.

7.3 Pitkän aikavälin tavoite

Pidempää aikaväliä miettien, laitehallinta halutaan käyttöön myös mScales-tiimin ulkopuolisille, eli että jokainen asiakas voisi hallita omia laitteitaan. Tästäkin on hyötyä molemmille osapuolille, sillä yrityskohtaisen hallinnan siirryttyä huolto-organisaatiolta asiakkaalle, riittää aikaa enemmän muuhun työhön. Hallinnan lisäksi asiakkaat voivat tietenkin seurata laitteiden tilaa ja lokitietoja, ihan kuten ylläpitäjätkin. Tieto laitteiden kunnosta ja virheistä tulee kaikille tietoon nopeasti ja vältytään turhilta tietokatkoilta.

Reaaliaikaisen laitetietojen lisäksi halutaan kerätä historiadataa kaikista laitteista, jotta voidaan seurata toiminnan kehitystä laajemmallakin aikavälillä. Tällaisen tiedon avulla voidaan helposti selvittää pullonkaulat ja laitteet tai tietyt tapahtumat, jotka tuottavat runsaasti ongelmia. Dataa voi tutkia käsin, mutta yksi suuri potentiaalinen mahdollistaja olisi tekoälyn ja koneoppimisen integrointi, jonka avulla ongelmatilanteet voitaisiin huomata ja mahdollisesti jopa reagoida tekoälyn avulla, ennen kuin ongelma on ehtinyt paisua kriittiseksi. Koneoppiminen on yksi tekoälyn käyttösovelluksista, jossa sovellus oppii edellisen datan perusteella. Koneoppimisessa tunnistetaan kuvioita ja toimitaan niiden perusteella. Tekoälyä hyödyntävät ohjelmat voisivat seurata laitteiden tilaa jatkuvasti ja ennustaa, mitä seuraavaksi tulee tapahtumaan ja onko laitteen kunto menossa huonompaan suuntaan. (Expert System Team 2017.)

8 YHTEENVETO

Tavoitteena oli luoda käyttöliittymä laitehallinnalle yrityksen sisäiseen sekä asiakkaiden käytettäväksi. Laitehallinnan kautta haluttiin tarkastella sekä muokata ja lisätä laitteita järjestelmään. Tässä tavoitteessa onnistuttiin ja keskeiset ongelmat saatiin ratkottua. Nykyisessä tilanteessa oleva laitehallinta on jo otettu tuotantokäyttöön yrityksen sisällä ja se on valmis vietäväksi asiakkaan käyttöön.

Vaiheittaisessa käyttöönotossa todettiin, että laitehallinnan kuuluisi olla dynaaminen ja joustava ja näin se lopulta toteutettiinkin. Ennen laitehallinnan käyttöönottoa, laitteet syötettiin tietokantaan käsin, eli laitteita oli jo jonkin verran järjestelmässä. Kun laitehallinta otettiin käyttöön, nämä käsinsyötetyt laitteet tulivat näkyviin ja hallittaviksi uudessa käyttöliittymässä ilman lisätoimenpiteitä.

Käyttöliittymän selkeyden ansiosta uusia laitteita lisätessä tai vanhoja muokatessa, tiedot menevät varmasti oikeaan paikkaan. Selkeä ja helppokäyttöinen käyttöliittymä vähensi käsinsyötöstä johtuvia virheitä ja vähensi työn määrää, sillä kaiken voi tehdä samasta paikasta. mScales tiimin käyttökokemukset ovat olleet positiivisia ja hallintaa on kehuttu helppokäyttöiseksi. Laitteiden hallinta on ollut helppoa myös henkilöille, jotka eivät olleet aikaisemmin asiaan perehtyneitä.

Seuraavaksi laitehallintaan tullaan tekemään laajempaa laitteiden lokitietojen seuraamista ja keräämistä, sekä muiden tilatietojen seurantaa. Muistiinpanojen ja parametrien lisäksi laitteisiin tulee olemaan mahdollista liittää dokumentteja, kuten manuaaleja. Laitehallinta tullaan ottamaan käyttöön entistä laajemmin koko organisaation sisällä ja sitä aletaan myös viedä asiakkaiden käytettäväksi. Laitehallinnan kehitystä aiotaan viedä eteenpäin pitkällä aikavälillä. Asiakkaiden antamien kommenttien perusteella tullaan tekemään parannuksia ja muutoksia.

LÄHTEET

Anand, S. 2017. What is Docker Container – An Introduction [viitattu 3.3.2020]. Saatavissa: https://acadgild.com/blog/what-is-docker-container-an-introduction

Cato, W. & Mobley, K. 2002. Computer-managed Maintenance Systems: A Step-by-step Guide to Effective Management of Maintenance, Labor, and inventory [viitattu 24.3.2020]. Saatavissa: https://books.google.fi/books?id=1QocqKlXzTgC&dq=cmms&redir_esc=y&hl=en

Docker. 2020a. What is a Container? [viitattu 3.3.2020]. Saatavissa: https://www.docker.com/resources/what-container

Docker. 2020b. The Industry-Leading Container Runtime [viitattu 3.3.2020]. Saatavissa: https://www.docker.com/products/container-runtime

Docker. 2020c. Use volumes [viitattu 3.3.2020]. Saatavissa: https://docs.docker.com/storage/volumes/

Expert System. 2017. What is Machine Learning? A definition [viitattu 3.3.2020]. Saatavissa: https://expertsystem.com/machine-learning-definition/

Federal Standard 1037C, 1996. Telecommunications: Glossary of Telecommunication Terms. Yhdysvallat: General Services Administration. [viitattu 24.3.2020]. Saatavissa: https://www.its.bldrdoc.gov/fs-1037/fs-1037c.htm

Fielding, R. 2000. Architectural Styles and the Design of Network-based Software Architectures [viitattu 23.3.2020]. Saatavissa: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Google. 2020a. Introduction to Angular concepts [viitattu 20.3.2020]. Saatavissa: https://angular.io/guide/architecture

Google. 2020b. Introduction to modules [viitattu 20.3.2020]. Saatavissa: https://angular.io/guide/architecture-modules

Google. 2020c. Feature Modules [viitattu 20.3.2020]. Saatavissa: https://angular.io/guide/feature-modules

Google. 2020d. Introduction to components [viitattu 20.3.2020]. Saatavissa: https://angular.io/guide/architecture-components

Google. 2020e. Lifecycle hooks [viitattu 20.3.2020]. Saatavissa: https://angular.io/guide/lifecycle-hooks

Google. 2020f. Introduction to services and dependency injection [viitattu 24.3.2020]. Saatavissa: https://angular.io/guide/architecture-services

Google. 2020g. Dependency injection in Angular [viitattu 24.3.2020]. Saatavissa: https://angular.io/guide/dependency-injection

IBM. 2019. IaaS (Infrastructure-as-a-Service) [viitattu 25.3.2020]. Saatavissa: https://www.ibm.com/cloud/learn/iaas#toc-what-is-ia-k7Negcdh

IETF. 2014. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing [viitattu 23.3.2020]. Saatavissa: https://tools.ietf.org/pdf/rfc7230.pdf

ISO 55000:2014, 2014. Asset management. Yhdistynyt kuningaskunta: British Standards Institution [viitattu 24.3.2020].

Lahti Precision Oy 2020a. Yritys [viitattu 1.3.2020]. Saatavissa: https://lahtiprecision.com/yritys/

Lahti Precision Oy 2020b. Digitaalisella punnituspalvelulla saat tehoja liiketoimintaan. Esite.

Lahti Precision Oy 2020c. mScales Arkkitehtuuri. Esite.

Lange, K. 2016. The Little Book On REST Services [viitattu 24.3.2020]. Saatavissa: https://www.kennethlange.com/books/The-Little-Book-on-REST-Services.pdf

Dignan, L. Top cloud providers 2018: How AWS, Microsoft, Google, IBM, Oracle, Alibaba stack up [viitattu 3.3.2020]. Saatavissa: https://www.zdnet.com/article/top-cloud-providers-2018-how-aws-microsoft-google-ibm-oracle-alibaba-stack-up/

Logistiikanmaailma. 2020. Toiminnanohjausjärjestelmä [viitattu 1.3.2020]. Saatavissa: http://www.logistiikanmaailma.fi/logistiikka/ohjausjarjestelmat/toiminnanohjausjarjestelma/

M, M. 2018. AngularJS and Angular2+: a Detaild Comparison [viitattu 20.3.2020]. Saatavissa: https://www.sitepoint.com/angularjs-vs-angular/

Mozilla. 2019. Connection management in HTTP/1.x [viitattu 23.3.2020]. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/HTTP/Connection_management_in_HTTP_1.x

NIST Special Publication 800-145. 2011. The NIST Definition of Cloud Computing. Yhdysvallat: U.S. Department of Commerce [viitattu 25.3.2020]. Saatavissa: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

Opensource.com. 2020. What are microservices? [viitattu 3.3.2020]. Saatavissa: https://opensource.com/resources/what-are-microservices

Pilvi. 2020. Mikä on SaaS-palvelu? [viitattu 1.3.2020]. Saatavissa: https://www.pilvi.com/fi/mika-on-saas-palvelu/

IETF. 2014. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing [viitattu 23.3.2020]. Saatavissa: https://tools.ietf.org/pdf/rfc7230.pdf

Pilvi. 2020. Mikä on SaaS-palvelu? [viitattu 1.3.2020]. Saatavissa: https://www.pilvi.com/fi/mika-on-saas-palvelu/

REST CookBook. 2012. What is HATEOAS and why is it important for my REST API? [viitattu 24.3.2020]. Saatavissa: http://restcookbook.com/Basics/hateoas/

Reese, G. 2000. Distributed Application Architecture [viitattu 23.3.2020]. Saatavissa: https://web.archive.org/web/20110406121920/http://java.sun.com/developer/Books/jdbc/ch07.pdf

Richardson, L. & Ruby, S. 2007. RESTful Web Services. O'Reilly Media, Inc. [viitattu 23.3.2020]. Saatavissa: https://archive.org/details/restfulwebservic00rich_0

Rouse, M. 2020a. Docker image [viitattu 3.3.2020]. Saatavissa: https://searchitoperations.techtarget.com/definition/Docker-image

Rouse, M. 2020b. Platform as a Service (PaaS) [viitattu 24.3.2020]. Saatavissa: https://searchcloudcomputing.techtarget.com/definition/Platform-as-a-Service-PaaS

Sdhabeger. 2010. Cloud Services [viitattu 25.3.2020]. Saatavissa: https://commons.wikimedia.org/wiki/File:Cloud_Services.gif

Seobility. 2020a. REST API [viitattu 23.3.2020]. Saatavissa: https://www.seobility.net/en/wiki/REST_API

Seobility. 2020b. Proxy Server [viitattu 24.3.2020]. Saatavissa: https://www.seobility.net/en/wiki/Proxy_Server

Shinder, T. 2008. Understanding Web Caching Concepts for the ISA Firewall [viitattu 24.3.2020]. Saatavissa: https://www.seobility.net/en/wiki/Proxy_Server

The Local Government & Municipal Knowledge Base. 2020. Asset Management [viitattu 24.3.2020]. Saatavissa: http://www.lgam.info/asset-management

Vignoni, D. & Calimo. 2011. Client-server-model [viitattu 23.3.2020]. Saatavissa: https://en.wikipedia.org/wiki/File:Client-server-model.svg

W3. 2004. Relationship to the World Wide Web and REST Architectures [viitattu 23.3.2020]. Saatavissa: https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest

Watts, S. 2018. Serverless vs PaaS: Is Serverless the New PaaS? [viitattu 25.3.2020]. Saatavissa: https://www.bmc.com/blogs/serverless-paas/

Wireman, T. 1994. Computerized Maintenance Management Systems [viitattu 24.3.2020]. Saatavissa: https://books.google.fi/books?id=1QocgKlXzTgC&dq=cmms&redir_esc=y&hl=en

Yahoo. 2013. VS: REST. Keskustelupalstakommentti. Lähettäjä Fielding, R. Lähetetty 7.2.2013 [viitattu 23.3.2020]. Saatavissa Yahoo keskustelupalstalla: http://tech.groups.yahoo.com/group/rest-discuss/message/5841