Hajautettu IoT-laite -toteutus pilvipalvelua hyödyntäen

Ville Oinonen

2021 Laurea

Laurea-ammattikorkeakoulu
Leppävaara

Hajautettu IoT-laite -toteutus pilvipalvelua hyödyntäen

Ville Oinonen
Tietojenkäsittely
Opinnäytetyö
Huhtikuu, 2021

Laurea-ammattikorkeakoulu **Tiivistelmä**
Tietojenkäsittely
Tietojenkäsittely tradenomitutkinto (AMK)

Ville Oinonen
Hajautettu IoT-laite -toteutus pilvipalvelua hyödyntäen
Vuosi 2021 Sivumäärä 43

Opinnäytetyön aiheena oli hajautettu IoT-laite -toteutus pilvipalvelua hyödyntäen. Toimeksiantajana oli Rauli Oinonen Consulting Oy, jolla oli tarve saada IoT-laite toimimaan palvelimesta käsin. Sen tuli mahdollistaa kotijärjestelmien ja IoT-laitteen etähallittavuuden käyttämisen kotiverkon ulkopuolella. Tavoite oli luoda palvelin, joka pystyi ohjaamaa IoT-laitteen toimintaa käyttöliittymän avulla.

Tuotteen ohjelmistokehityksen mallina toimi vesiputousmalli, joka sopi hyvin halutun lopputuloksen saavuttamiseen. Työn tavoite oli luoda käytettävyyden prototyyppi, jonka kehittämistä tullaan jatkamaan sille sopivalle sidosryhmälle. Tavoitetulos oli saada tietoa, miten projekti voidaan toteuttaa yksinkertaisesti ja kuinka siitä saadaan helposti kehitettävä. Projekti pidettiin yrityksen sisäisessä kehityksessä, eikä projektin opinnäytetyövaiheessa tarvinnut palvella asiakkaita.

Projektin tietoperusta muodostui yleisestä verkkokehityksestä ja projektin kehittämisessä käytetyistä rakenteellisista periaatteista. Käsittelin kehitysmenetelmän rakennetta sekä kerroin, miksi päädyin valitsemaan kyseisen menetelmän.

Projektin lopputulos oli toivotun mukainen ja se täytti sille asetetut määritelmät hyvin. Lopputuote oli IoT-laitteen näytön hallinta verkossa käyttäen käyttöliittymässä toimivaa nappia.

Asiasanat: IoT, Hajautetut järjestelmät, versionhallinta

Laurea University of Applied Sciences **Abstract**
Degree Program in Business Information Technology
Bachelor Thesis

Ville Oinonen

A distributed IoT implementation utilizing cloud platforms

Year 2021 Pages 43

The subject of this thesis project is a distributed IoT device implementation using cloud services. The client was Rauli Oinonen Consulting Oy, who needed to have an IoT device controlled from the server. This was to allow secure remote control of systems and IoT devices outside the local network. The goal was to create a server that could control the operation of an IoT hardware device through a web user interface.

The product development model utilized for this product, was the waterfall project lifecycle model, which was well suited to achieve the desired results. The aim of the work was to create a usability prototype, which would later be further developed for a suitable commercial use case. The goal was to get information on how the project could be implemented simply, and how it could be easily developed further. The project was conducted for the purposes of internal development of the company, and there was no need to serve customers during the thesis phase of the project.

The knowledge base of the project consisted of the general network development and the structural principles used in the development of the project. The structure of the development method was discussed as well as the reasons for choosing that method.

The project met its set goals and expectations well. The deliverable was a working system consisting of a web user interface that could securely control a remote IoT hardware device.

Keywords: IoT, Distributed systems, Version control

Sisällys

1 Johdanto

IoT-laitteet ovat nykypäivänä yhä enemmän ihmisten keskuudessa tekemässä automatiikkaa ja keräämässä dataa. Tässä opinnäytetyössä käyn läpi, kuinka palvelimeen voidaan helposti liittää IoT-laite, joka viestii palvelimelle omaa dataa samalla, kun vastaanottaa sitä. Näiden välisessä keskustelussa voimme antaa laitteelle käskyjä käyttäjän selainkäyttöliittymän kautta.

Aihe oli itselle mielenkiintoinen, sillä olen itse hyvin kiinnostunut älykotijärjestelmästä ja tämän työn avulla minulla olisi mahdollisuus luoda itselleni omanlainen älykotilaitteisto. Tämä tuo samalla hiukan lisää ymmärrystä verkkopalvelujen tuottamisesta myös fyysisiin tuotteisiin.

Opinnäytetyö antaa esimerkin palveluista, joiden avulla lukija voi taloudellisesti tehdä samankaltaisen työn ilman suurempia kustannuksia. Projektin kustannuksia olivat laitekustannukset, kuten ESP32-piirilevy.

Aluksi kerron projektin tavoitteen ja rajauksen. Pyrin tuomaan esille projektin käsittelemät osa-alueet ja kertoa, kuinka laajasti näitä käsitellään projektissa. Kerron lyhyesti projektin toimeksiantajan ja hänen taustansa. Projektin termistö on laaja. Kahden teknologian yhdistäminen turvallisesti vaatii paljon teknisiä menetelmiä. Projektin kehittämiseen olen ottanut useita eri periaatteita ja menetelmiä, joiden avulla pyrin mahdollistamaan jatkokehittämisen mutkattomuuden.

Toimeksiantaja toimii liiketalouden ja tietotekniikan konsulttina, jolla olisi toive saada IoT-laitteet keskustelemaan palvelimen kanssa ja toteuttamaan eri toimintoja palvelimelta käsin. Toimeksiantajalla on aikaisempaa kokemusta IoT-laitteiden käytöstä, mutta ei niinkään verkkopuolen osaamista.

Ensin esittelen työn teoreettisesti. Käyn läpi tekniikoita, kuinka ne toimivat sekä miten niiden avulla päästään haluttuun lopputulokseen. Käytän vesiputousmallimenetelmää, missä mallin eri vaiheet määrittelevät projektin etenemisjärjestyksen. Teorian jälkeen käyn läpi prosessin toteuttamisen vaiheet. Käsittelen mitä olen tehnyt, kuinka ratkaisin ongelman ja miksi otin kyseiset tekniikat käyttöön.

1.1 Työn tavoite ja rajaus

Opinnäytetyön tarkoitus on luoda protyyppi, joka käsittelee verkkopalvelimen ja IoT-laitteen välistä kommunikointia. Verkkopalvelimeen lisätään IoT-laitteen tuki, johon IoT-laite ESP32

WROOM-32 Development Board Module voi ottaa yhteyttä ja vastaanottaa dataa. Prototyypin tarkoituksena on tutkia, miten IoT-laitteen voi liittää julkiseen palvelimeen tietoturvallisesti. Esimerkkinä toimii piirilevyssä sijaitseva näyttö, jossa näkyvään tekstiin voi tehdä muutoksen verkkosivuilla toimivan napin avulla. Tämä kokonaisuus rakennetaan käytettävyysprototyyppinä (Feasibility Prototype). Lopputulos ei ole tarkoitettu vielä asiakkaalle, vaan sisäiseen testaamiseen tai mahdolliseen demoversion näyttämiseen eli demomalliin (Mockup). Konsepti antaa pohjan jatkokehittämiselle ja opettaa, kuinka helposti onnistuu viestinnän luominen kahden teknologian välillä. (Skinner 2020.)

Tämä opinnäytetyö antaa yksinkertaisen esimerkin IoT-laitteen ja palvelimen välisestä keskustelusta. Siinä ei käydä läpi IoT-laitteen muita ominaisuuksia, vain verkkoyhteyden ottaminen ja jonkinlainen tiedonsaannin vastaanottamisen antava ilmoituksen näyttäminen. Käsittelen kuitenkin, kuinka IoT-laitteen ja palvelimen välinen keskustelu voidaan tehdä tietoturvallisesti ja kuinka nämä kaksi osapuolta voivat keskustella keskenään. Käsittelen myös tietokannan käyttöönottamisen palvelimessa. Projektin voi luoda ilman tietokantaa, mutta projektin jatkokehitystä varten se on tärkeä osa turvallista kirjautumista.

Koska projektin tarkoitus on palvelun ja IoT-laiteen kehittäminen, visuaalista ulkoasua ei ole työstetty. Projektin esivalmisteltuja osia ovat palvelimeen kirjautuminen käyttöliittymän kautta kovakoodatuilla tunnuksilla ja itse käyttöliittymä. Näiden osien teknologian tulen käymään läpi projektissa, sillä ne luovat turvallisen ympäristön kehittää IoT-laitepalvelua jatkossa eteenpäin. Valmiita osia tulen edistämään palvelimeen ja siirrän käyttäjätunnukset tietokantaan. En selitä yksityiskohtaisesti niiden luontia, mutta tarvittavan teorian niiden kehittämiselle.

Opinnäytetyön toimeksiantaja oli jo valmiiksi tutkinut ja valinnut sopivan laitteen projektiin. En vertaile eri IoT-laitteita keskenään, sillä suurin osa markkinoilla olevista laitteista pystyy tekemään saman asian. Markkinoilla on tuhansia eri versioita, joista pitää vain löytää laite, jolla on kyky yhdistyä verkkoon. Muut ominaisuudet ovat käyttäjän itse valittavia. Projekti mahdollistaa lukijan luomaan oman IoT-laiteympäristön kotiverkkoon sekä käyttää sitä myös julkisessa verkossa.

Opinnäytetyö olettaa, että lukijalla on jonkin verran kokemusta ja ymmärrystä verkkosivukehittämisestä. Pyrkimys on käsitellä aihetta päällisin puolin ja joitakin yksityiskohtia lähemmin, mutta itse koodia ei työssä käsitellä muuten kuin käsitteiden avulla.

Projektissa tietoturvallisuus huomioidaan tietoliikenne-, käyttö- ja laitteistoturvallisuudessa. Näiden osa-alueiden avulla voimme luotettavasti julkaista projektin lokaaliverkosta julkiseen verkkoon. (Laakso 2010, 28-29.)

1.2 Toimeksiantaja

Toimeksiantajana on Rauli Oinonen Consulting Oy, joka toimii liikkeenjohdon konsultointiyrityksenä. Yritys auttaa asiakkaita löytämään ratkaisuja tietotekniikassa ja liiketaloudellisissa asioissa. Raulilla on omana harrastuksenaan pienten elektronisten laitteiden rakentaminen.

Toimeksiantajan kanssa kävin läpi, minkälaista toiminnallisuutta verkkosivupalveluun halutaan ja kuinka tämä tulisi toteuttaa. Toimeksiantaja käyttää prototyyppialustana ESP32-mikrokontolleria, joka toimii projektin esimerkkinä. Olen alustavasti toteuttanut projektille palvelimen ja käyttöliittymän, jossa toimeksiantaja voi kirjautua sisään ja käyttää palvelinta. Opinnäytetyö toimii myös oppimismateriaalina toimeksiantajalle.

2 Terminologia

Hakusanoina toimivat englanninkieliset termit, joiden avulla sain enemmän sisältöä aiheesta ja termistö oli helpommin ymmärrettävissä. Google ja YouTube toimivat pääsääntöisinä hakulaitteina. Pohjana projektin verkkoviestintään käytin valmista Arduinon tarjoamaa lähdekoodia HTTP- ja HTTPS-viestinnän toteuttamisesta (Random Nerd Tutorials 2020). Tätä edistin itselleni sopivaksi kokonaisuudeksi, jolloin pystyin tekemään viestintää myös SSL-salauksen avulla.

En käyttänyt lähteenä painettua kirjallisuutta, sillä sain kaiken tarvitsemani aineiston verkosta. Jotkut lähteet ovat vanhoja, mutta vaikka teknologia kehittyy, sen tukipalat pysyvät samoina. Näitä kutsutaan nimityksellä standardi. (Korpela 2007.)

Toimeksiantajalla oli tietoa ja kokemusta IoT-laitteen kyvyistä. Laitteella oli kattava dokumentaatio, jolla pystyi helposti luomaan eri toiminnallisuuksia. Valmista lähdekoodia oli paljon.

IoT-laitteita on käytössä paljon ja moneen eri tarkoitukseen. Tätä tekniikkaa käytetään mm. ilmatieteenlaitoksen antureissa. Niissä kerätään tietoa verkon ylitse palvelimelle, jossa dataa voidaan analysoida. Teknologia on kehittymässä kovaa vauhtia ja sen myötä yhä useampi kodinkone on mahdollista yhdistää verkkoon. (Ilmatieteenlaitos 2020.)

2.1 Projektin kehitysmalli

Vesiputousmalli on kehitysmenetelmä, jossa projekti käydään vesiputoustyylisesti vaihe kerrallaan alaspäin. Mallin kehittämisen elinkaari on koottu tarkasti määritetyistä osista, joita

alkuperäisesti on kaksi, suunnittelu ja toteutus sekä testaus ja ylläpito. Nämä palat asettavat projektin toimimaan vaiheittain. (Helsingin Yliopisto 2009.)

Malli on ensimmäisiä kehitysmalleja, josta tuli nopeasti hyvin suosittu. Malli on saanut erilaisia muunnoksia sen vaiheista ja niiden erittelystä. Nämä muutokset tehdään yleensä riippuen projektin tyylistä, sekä kuinka menetelmä asettuu tiimin ja projektin toimenkuvaan. Sisältö kuitenkin pysyy jokaisessa mallissa samana. (Turotialspoint 2021.)

Vesiputousmalli eroaa ketteristä menetelmistä sen omasta kehittämisen elinkaaren kierteestä. Vesiputousmallissa niitä on vain yksi ja sen pituus voi olla viikosta vuosiin. Ketterän kehityksen mallissa näitä on useampi projektin aikana ja niiden kesto on yleensä lyhyempi, päivistä kuukauteen. (Santos 2020.)

2.2 IoT-laite

IoT (Internet of Things) on kokonaisuus, joka koostuu sensoreista, datan prosessoinnista, sen tallentamisesta ja siirtämisestä verkon ylitse palvelimelle käsiteltäväksi ja tallennettavaksi. Laite voidaan muuttaa vastaanottamaan dataa, joka sisältää sille tarkoitettuja käskyjä. Nämä käskyt voivat laukaista laitteelle tehtyjä toiminnallisuuksia. (Wyliodrin 2015.)

IoT-laite projektissa sisältää mikrokontrollerin, joka toimii kuin pieni tietokone. Tälle kontrollerille on annettu yksi toiminto, jonka se toistaa ollessaan päällä. (Wyliodrin 2015.)

2.3 SoC

SoC (Separation of concerns) on ohjelmistokehityksen periaate. Sen tarkoituksena on ohjata ohjelmistokehitystä modulaarisiin kokonaisuuksiin. Nämä kokonaisuudet käsittelevät vain yhden osan suurempaa kokonaisuutta. Periaatteessa on tärkeää hahmottaa moduulien välinen viittaus, jotta järjestelmä ei sotkeutuisi ja hankaloittaisi järjestelmän tietovirtaa. (Naumov 2020.)

Modulaarinen järjestelmä mahdollistaa joustavuuden mahdollisiin muutoksiin ja selkeyttää koodin kulkua. Työkokonaisuudet on helpompi jakaa, ilman niiden riippuvuutta toisiinsa. (Naumov 2020.)

2.4 Bluetooth

Bluetooth on lyhyen matkan tiedonsiirtotekniikka, joka on yleinen langattomissa laitteissa. Se kuluttaa vähemmän sähköä ja on nopeampi viestinnässä kuin Wlan-yhteys. Bluetooth-laitteiden välinen etäisyys on rajoittuneempi kuin Wlan-yhteyden avulla suoritettu viestintä. (Uy 2020.)

Tietotuvallisesti Bluetooth-yhteys on turvallisempi, sillä viestintä on kryptattua ja taajuuden vaihtelu on yleistä. (Uy 2020.)

2.5 ESP32

ESP32 on Wi-Fi- ja Bluetooth-yhteyden omaava mikrokontrolleriyksikkö, joka mahdollistaa datan käsittelyn ja lähettämisen verkon ylitse palvelimelle. Yksikkö pystyy prosessoimaan pientä määrää dataa ja käskemään piiriä toteuttamaan erilaisia toimintoja syöttämällä sähköä piirin eri osiin. (ESPRESSIF 2020.)

Tärkeä osa opinnäytetyötä on piirissä toimiva näyttö. Näyttö mahdollistaa tekstin tai muun visuaalisen asian piirtämisen ja antaa mahdollisuuden toiminnan vaiheiden seuraamiseen. Mikrokontrolleria voidaan myös seurata tietokoneeseen yhdistettynä Serial Monitorin kautta. (ESPRESSIF 2020.)

2.6 On-Premises

On-Premises on fyysinen kokonaisuus, joka toimii paikallisesti organisaation sisällä toimivana palvelimena. Ohjelmistolle on pystytettävä kaikki tarvittava infrastruktuuri, koska se toimii ainoastaan sisäverkossa, ellei sitä päästetä reitittimen kautta julkiseen verkkoon. Tätä voi verrata omaan tietokoneeseen, jossa palvelin pyörii ja se on liitettynä omaan kotiverkkoon. Palvelu ei siis toimi organisaation ulkopuolisella kolmannen osapuolen palvelimella, niin kuin muut pilvipalvelut. (Itewiki, N.d.)

2.7 IaaS

Infrastructure as a Service on palvelu, jossa käyttäjä ostaa laitteet ja niiden ylläpidon palvelimelta. Käyttäjä ei osta omia laitteita fyysisesti, vaan vuokraa tarvittavat laitteet käyttöönsä. Käyttäjä maksaa palvelusta ostettujen palvelujen määrän mukaan. Tämä ratkaisu on hyvä, mikäli yritys ei halua itse ostaa fyysistä laitetta prosessien suorittamiseen tai datan tallentamiseen. Tämä ratkaisu vaatii parempaa ymmärrystä verkkopalvelimen infrastruktuurista kuin PAAS- ja SAAS -pilvipalvelumuodoissa. IaaS-palveluja tarjoavat Google Apps, AWS EC2 ja Hubspot. (Watts, Raza 2019)

2.8 PaaS

Platform as a Service on pilvessä toimiva palvelu, jota kehittäjät voivat käyttää applikaatioiden julkaisussa. PaaS sisältää valmiin infrastruktuurin, jonka myötä kehittäjille jää vain applikaation luominen. Kehittäjät voivat valita palvelusta valmiita komponentteja, joilla palvelua voidaan laajentaa tarpeen mukaan. Tämä voi vähentää kustannuksia ja aikaa palveluiden kehittämisessä. (Watts, Raza 2019.)

2.9 SaaS

Software as a Service on palvelimessa pyörivä applikaatio, jossa asiakas voi ostaa tiettyjä palveluita käyttöönsä. Palvelua ei tarvitse asentaa. Palvelu on pilvessä ja toimii pääsääntöisesti selaimessa. Vastaavia palveluita ovat Google Workspace, Dropbox ja Netflix (Watts, Raza 2019.)

2.10 JWT

JWT (JSON Web Tokens) on avoin standardi, joka mahdollistaa tietoturvallisen datan siirtämisen kahden osapuolen välillä. Standardi käyttää JSON-formaattia datan siirtämisessä, mikä mahdollistaa datan helpon käytettävyyden. JWT:tä käytetään pääsääntöisesti käyttäjien tunnistautumisessa ohjelmistoissa. (JWT 2020.)

Standardista on luotu useita kirjastoja, jotka tukevat eri koodikieliä. Nämä kirjastot huolehtivat standardin helposta käytettävyydestä (JWT 2020). Tästä yleisin kirjasto on jsonWebtoken, joka käyttää JWT-standardia.

2.11 Token

Token on tunnus, jolla osapuolet voivat tunnistaa toisen osapuolen virallisuuden. Se koostuu useammasta merkistä. Tunnuksen voi purkaa vain sen luonut osapuoli.

Token sisältää kaiken tarvittavan tiedon osapuolten todennukseen. Tämä mahdollistaa palvelimen istuntotilojen ylläpitämistä. Tokenin avulla voi kryptatusti määrittää, mihin käyttäjällä on pääsyoikeudet. (AuthO 2016.)

2.12 GUID

GUID (Globally Unique Identifier) on laitteelle annettu tunniste. Tunniste koostuu numeroista ja kirjaimista. Tätä käytetään ohjelmistossa yksilöllistämään objekteja toisistaan. Datan siirtäminen vaatii tunnisteen, jotta sen alkuperä voidaan yhdistää tai siihen voidaan viitata tunnisteen avulla. Tunnuksia voidaan nähdä Windowsin rekisterimerkinnöissä ja erilaisissa tiedostotyypeissä. (TechTerms 2021.)

2.13 Node.js

Node.js on alustariippumaton Javascript-ajoympäristö. Se on kevyt ja tehokas, ja sillä voi luoda reaaliaikaisia applikaatioita. Alustariippumattomuuden ansiosta sitä voidaan käyttää kaikilla käyttöjärjestelmillä. Noden avulla lähdekoodin suorittaminen voidaan suorittaa palvelimella, jolloin verkkosivu voidaan lähettää valmiina pakettina käyttäjälle. (Tutorialspoint 2020.)

Alustan avulla voidaan nopeasti tuottaa yksinkertaisia palvelinratkaisuja. Monet verkkopalvelut tarjoavat Node.js:lle helpon ja suoraviivaisen julkaisuputken. Java mahdollistaa javascripti-koodin käyttämisen myös palvelimella. (Tutorialspoint 2020.)

2.14 Arduino

Arduino on avoin lähde elektronisille alustoille. Arduinon rakenne pohjautuu elektroniikkalaitteistoon ja ohjelmistoympäristöön, joiden avulla käyttäjä voi luoda omia mikrokontrollereita. Arduino on ohjelmointikieli mikrokontrollereille, joka käyttää C++ -pohjaista kieltä. (Arduino 2020.)

Arduinon tarkoitus on tuoda markkinoille helposti käytettävän ympäristön luoda omia mikrokontrollereita omiin tarpeisiin. Arduino-ohjelmointiympäristö on alustariippumaton, mutta mikrokontrollerit on rajoitettu Windows-käyttöjärjestelmiin. (Arduino 2020.)

Arduinoa käytetään sekä yritystasolla että harrastusmielessä. Arduino on halpa ja yksinkertainen ja siihen löytyy paljon valmista materiaalia, joten aloittelijakin oppii käyttämään mikrokontrollereita. (Arduino 2020.)

2.15 HTTPS

HTTP (Hypertext Transfer Protocol) on protokolla, jolla verkkoviestintä tapahtuu eri laitteiden välillä. Tästä on laajennettu versio, jossa liikenne salataan SSL (Secure Sockets Layer) -protokollan mukaisesti. Tämä versio on HTTPS. Tämä mahdollistaa viestinnän olevan kryptattu ja silloin ulkopuolisen on sitä hankalampi purkaa. (MDN Web Docs 2020.)

2.16 WLAN

WLAN (Wireless Local Area Network) on reitittimen ja siihen yhdistetyn laitteen välinen keskustelukaista. Wlanin kautta keskustelevat laitteet yleensä käyttävät Wi-Fi -standardia. Tämä mahdollistaa laitteiden välillä langattoman keskustelun ja yhteyden Ethernetiin. (TechTerms 2020.)

2.17 Rest API

API (Application programming interface) on applikaation rajapinta, jota käytetään yleensä verkkopalvelimissa datan saamiseksi, muuttamiseksi tai poistamiseksi palvelimesta. (Red Hat 2020.)

API:ssa voidaan käyttää lisämäärittelynä termiä REST (Representational State Transfer. REST on arkkitehtuuri, jonka tarkoituksena on määritellä, kuinka rajapinnan pitää toimia ja mitä sen pitää sisältää. Tämä määrittää rajapinnan toimimaan tilattomana keskusteluna, mikä

tarkoittaa, että käyttöliittymän ja palvelimen välinen keskustelu suoritetaan yksittäisillä http-kutsuilla. Jokainen kutsu toimii erillisenä toisistaan ja suorittaa yhden toimenpiteen, jonka jälkeen yhteys katkeaa. Datan siirtäminen toimii yhtenäisenä formaattina palvelimen ja käyttöliittymän välillä. Palvelimelta tuleva data sisältää kaiken tarvittavan, mitä käyttöliittymä vaatii sen esittämiseen. (Red Hat 2020.)

Termi Restful tarkoittaa lähes samaa asiaa kuin REST mutta se ilmaisee, että REST-arkkitehtuuri on otettu täysin mukaan API-suunnitteluun ja täyttää kaikki sille määritetyt vaatimukset. (Long 2015.)

2.18 JSON

JSON (JavaScript Object Notation) on datarakenneformaatti, jonka avulla data voidaan pitää objektimuodossa. Formaattia käytetään yleisesti datan siirtämisessä serverien ja applikaatioiden välillä, kuin myös applikaation sisäisen datan hallinnassa. (SquareSpace 2020.)

JSON:in edeltäjä on XML (Extensible Markup Language), jonka rakenne on erilainen ja se on hankalampi käsitellä. JSON mahdollistaa helpon ja yksinkertaisen käsittelyn avain (key) ja arvo (value) -pareilla. (SquareSpace 2020.)

2.19 Spike

Spike eli POC (Proof of concept) on ketterän menetelmän tyyli, jossa tehdään pienestä kokonaisuudesta yksikertainen kokonaisuus ja se tehdään testausmuodossa. Tämä ei ole valmis tuote vaan pienimuotoinen kokeilu, jossa tutkitaan, onko se mahdollinen ja onko hyötyä kehittää sitä pidemmälle. Projektin tarkoitus ei ole tehdä tarkkaa pohjatyötä ohjelmistolle. Projektin pituus on yleensä lyhyt ja se dokumentoidaan hyvin. Se antaa osviittaa siihen, paljonko mahdollinen valmis projekti voisi viedä aikaa ja resursseja. (Mistry 2019.)

2.20 Semanttinen versiointi

Semanttinen versiointi on säännös, kuinka versioiden numerointi jaetaan ja miten niitä lisätään. Säännöksen formaattina toimii kolmiosainen 0.0.0 -asetelma, jossa ensimmäinen numero kertoo tuotantoversion. Tämä numero määrittelee kaikki suuret muutokset projektissa. Mikäli projektiin tehdään muutoksia, mitkä eivät ole taaksepäin yhteensopivia, muutetaan version ensimmäistä numeroa. Mikäli uusi muutos on taaksepäin yhteensopiva, voidaan muuttaa keskimmäistä numeroa. Muutos viimeiseen numero-osioon tarkoittaa, että nykyiseen kokonaisuuteen tehdään virheen korjausmuutos, joka on taaksepäin yhteensopiva. (Preston-Werner 2020.)

2.21 Pipeline

Pipelinen tarkoituksena on luoda kehitykselle putki, jonka läpi koko projekti siirtyy koodauksesta lopputulokseen eli julkiseen verkkoon asiakkaan nähtäväksi ja kokeiltavaksi. Tämä putki koostuu kahdesta osasta: DEV (Development) ja OPS (Operations). Putki luo loputtoman kehityssilmukan. Silmukka sisältää suunnittelun, koodauksen, rakentamisen, testauksen, julkaisun, käyttöönoton, käytön ja seurannan. (Danicic 2020.)

Projektista olen jättänyt automaatiotestauksen pois, sillä se ei tuo lisäarvoa työlle. Testaus suoritetaan lokaalissa ympäristössä manuaalisesti ja lopuksi vielä verkossa. (Danicic 2020.)

2.22 RFC

RFC (Request for Comments) on IETF-organisaation (Internet Engineering Task Force) ylläpitämiä standardikokonaisuuksia, jotka määrittävät erilaisia internetin käytäntöjä ja teknillisiä järjestelmämenettelyjä. (Korpela 2004.)

2.23 Serialization ja Deserialization

Serialization tarkoittaa datan rakenteellista muuttamista siten, että sen siirtäminen tai tallentaminen olisi yksinkertaista. Tämä toimenpide tehdään pääsääntöisesti koodissa objekteille, sillä emme halua tallentaa tai lähettää kaikkea objektissa olevaa sisältöä. Yleensä datan muuttamisessa luodaan objektin DTO eli data transfer object (Martin Fowler. N.d). Objectista otetaan talteen vain se mitä halutaan. (Divine 2019.)

Deserialization on serialization vastakohta. Siinä vastaanotettu data halutaan ottaa ohjelmistossa käyttöön. Tässä prosessissa objekti luodaan uudelleen ja sille annetaan DTO:n antamat arvot. Uusi objekti sisältää toiminnallisuuden ja sitä voidaan käyttää ohjelmistossa tavallisena objektina. (Divine 2019.)

Kun dataa halutaan siirtää, voidaan objektista luoda DTO, jonka parsiminen kirjainjonoksi tapahtuu yksinkertaisemmin ja helpommin. Tämä vähentää lähetettävän ja tallennettavan datan määrää. Myös dataa siirrettäessä tiedetään, missä muodossa se on. Tämä toiminnallisuus on mukana myös JSON-rakenteessa, jossa objektit muutetaan merkkijonoksi. (Divine 2019.)

2.24 Domain

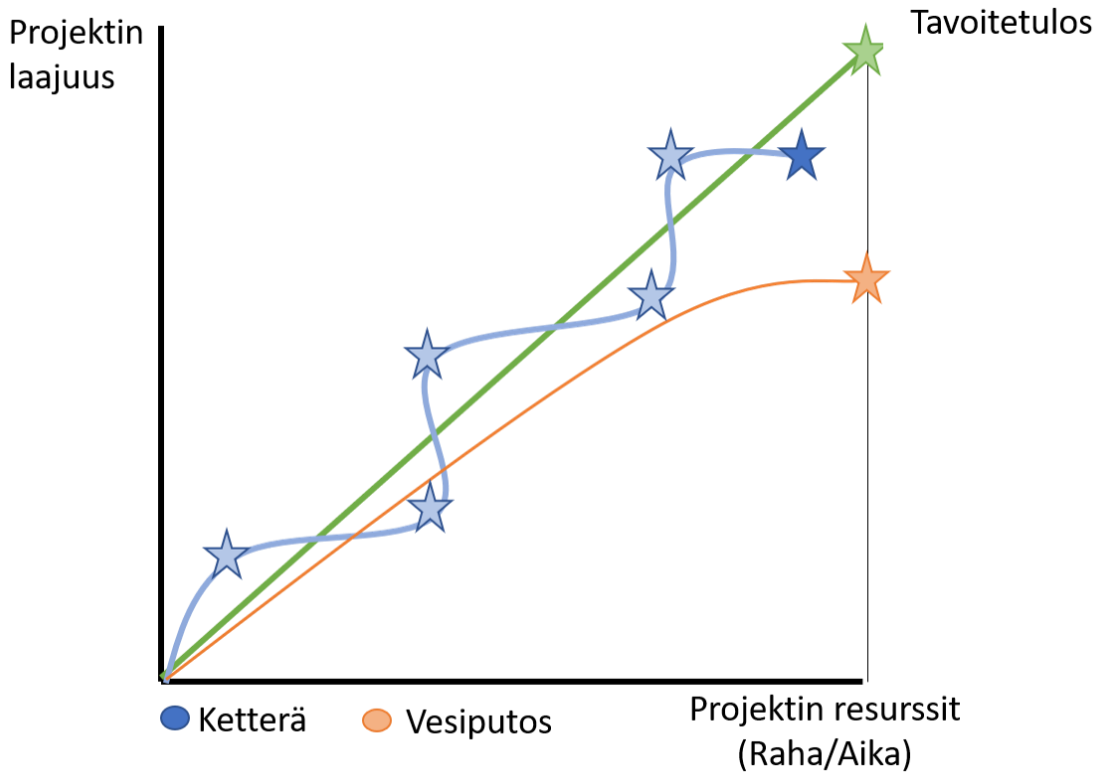
Domain on verkkotunnus, joka on kirjaimista koostuva nimi. Sen avulla käyttäjän on helpompi ja selkeämpi yhdistyä numeroista koostuvan IP-osoitteeseen (Internet Protocol Address). Domain tarjoaa verkko-osoitteen selkäkielisen nimen. (Wpbeginner 2020.)

Verkkotunnuspalvelu DNS (Domain Name System) ylläpitää yhteyksiä nimien ja IP-osoitteiden välillä. Verkkotunnus vaatii käyttäjältä IP-osoitteen, johon kyseinen nimi viittaa ja ohjautuu. Tämä lisää viestinnän turvallisuutta, sillä osoite on todennettu DNS-palvelun kautta. (Wpbeginner 2020.)

3 Teoreettinen rakenne ja Verkkopalvelun kehittämisen vaiheet

Projektissa voidaan käyttää hyväksi verkkopalvelun kehittämisen elinkaarta, jonka avulla pystyy jakamaan kehittämisen neljään osaan. Nämä osat määrittävät, miten eri vaiheissa projektia kannattaa edetä ja mitä kukin vaihe sisältää. (Digi- ja väestötietovirasto 2014.)

Projektissa kannattaa käyttää ketterän kehityksen menetelmiä, mikäli projektitiimin koko on suurempi kuin pari henkilöä. Näin projektissa mukana olevat henkilöt voivat paremmin pitää toisensa ajan tasalla ja tehdä muutoksia ongelmatilanteiden tullessa. Projektin koko ja tavoite ovat tärkeitä valintatekijöitä, sillä projektin resurssit voivat olla rajalliset ja haluttuun lopputulokseen pääsemiseksi ei välttämättä tarvitse tehdä täydellisiä ratkaisuja. (Koski N.d.)



Kuva 1: Vertailu ketterän menetelmän ja vesiputousmallin välillä

Ketterässä kehityksessä tuotetta on tarkoitus tuoda käyttäjälle useammassa vaiheessa, kun taas vesiputousmallissa tuote tuodaan vasta lopussa. Kuva visualisoi, miten projektin koko, tavoite ja resurssit voivat määrittää projektissa käytettävää kehitysmenetelmää (kuva 1).

Vesiputousmalli sopii hyvin projektiin, jossa vaatimukset on määritelty tarkasti jo alussa ja niiden muuttaminen matkan varrella ei ole tarpeellista. Muutoksia voidaan tehdä vielä suunnitteluvaiheen jälkeen, mikäli tämä on välttämätöntä projektin etenemiselle ja tavoitteen saavuttamiselle. (Blomvist 2018.)

Vesiputousmallin vaiheet ovat suunnittelu- ja vaatimusmäärittely, hankinta ja kilpailutus, toteutus ja testaus sekä käyttöönotto. Suunnittelu- ja vaatimusmäärittelyssä katsotaan toimeksiantajan kanssa, kuinka projekti tullaan toteuttamaan ja mitkä ovat projektin vaatimukset. Hankinta ja kilpailutus -vaiheessa vertaillaan eri palvelujen tarjontaa ja hinnoittelua sekä tehdään tarpeellisia hankintoja. Toteutus ja testaus -vaiheessa projekti etenee määrittelyjen ja suunnitelman mukaisesti. Toteutus-vaiheessa pyritään tuottamaan toivottu lopputulos. Testaamisella pyritään löytämään virheitä sekä ennaltaehkäisemään lopputuloksessa ilmeneviä virheitä. Näiden ehkäiseminen on tärkeää loppukäyttäjää ja jatkokehitystä ajatellen. Käyttöönotossa tuote luovutetaan toimeksiantajalle ja arvioidaan tavoitteiden toteutuminen. (Digi- ja väestötietovirasto 2014.)

3.1 Suunnittelu- ja vaatimusmäärittely

Suunnitteluvaiheessa selvitetään, mitä projektilla halutaan saavuttaa, mihin sitä käytetään ja miksi projekti toteutetaan. Projektissa kannattaa käyttää jotain sovelluksen kehitysmenetelmää, jonka mukaan projekti etenee. (Digi- ja väestötietovirasto 2014.)

Määritellään laatukriteerit, joiden mukaan mitataan lopputuloksen onnistumista. Näiden on hyvä olla mahdollisia ja tarpeeksi laajoja kriteerejä. Projektin määrittelyä voidaan käsitellä käyttäjän näkökulmasta tai kehityksen näkökulmasta. Projektin tarkoituksesta ja sidosryhmistä riippuen käyttäjän näkökulmaa ei aina tarvita. Projektin kuluessa voidaan tehdä lisäkriteereitä jatkokehitystä varten. (Digi- ja väestötietovirasto 2014.)

Teknologiaprojektiin kannattaa määrittää teknisiä vaatimuksia. Niitä voivat olla eri laitteiden tukeminen, toiminnallisuus tai kuinka hyvin työ pitää olla dokumentoitu. Vaatimukset kannattaa asettaa käsittelemään projektin eri osa-alueita. (Digi- ja väestötietovirasto 2014.)

Suunnitteluvaiheessa toteutusidea on hyvä tuoda ilmi graafisesti. Tämä helpottaa projektin jäseniä ymmärtämään haluttu kokonaisuus. Suunnitteluvaiheessa on tärkeää ottaa huomioon mitä tekniikoita, palveluita tai säädöksiä projekti vaatii. Suunnitelmaan on hyvä sisällyttää hallintamalli, joka sisältää projektin tarvittavat työvaiheet. Hallintamallin avulla pyritään

välttämään turhia työvaiheita ja pitämään kehitysputki selkeänä. Mikäli muutoksia tulee, on nämä perusteltava ja dokumentoitava. (Digi- ja väestötietovirasto 2014.)

3.2 Hankinta ja kilpailutus

On-Premises antaa käyttäjälle enemmän mahdollisuuksia hallita tietokonemaisesti kaikkia askelia nettiverkon ja applikaation välillä, mutta tämä vaatii käyttäjältä enemmän infrastruktuurin ylläpitämisen hallitsemista. Tämä voi olla halvempi ratkaisu isoissa palveluissa, mutta pienessä palvelussa tämä ratkaisu voi olla hidastava tekijä. (Watts, Raza 2019.)

Mikäli projekti halutaan siirtää julkiseen verkkoon pilvipalvelimelle, on järkevää kilpailuttaa eri palveluita ja niiden tarjontaa. PaaS ja IaaS -palvelujen välillä ei ole suurta hintaeroa, sillä molemmissa palvelutyypeissä on myös ilmaisia käyttöasteita. Hinta koostuu eri palveluissa ostettujen komponenttien määrästä ja palvelimen käytön määrästä. Molemmat kokonaisuudet antavat samanlaisen pohjan, mutta IaaS tarjoaa mahdollisuuden toteuttaa infrastruktuuriosia itse. PaaS antaa käyttäjälle paremman ja suoraviivaisemman tavan julkaista omia projekteja automaattisesti. IaaS tarjoaa käyttäjälleen mahdollisuuden laajempaan applikaatiojärjestelmään, sillä käyttäjä voi valita tarvittavan käyttöjärjestelmän. (Ahonen 2014.)

3.3 Toteutus ja testaus

Toteutuksessa valitaan ohjelmistosuunnittelun periaatteet, joiden mukaan yhdistetään suunnitteluvaiheessa päätetyt teknologiat. Varmistetaan hankintavaiheen palvelut ja resurssit. Jokainen työvaihe määritellään ja asetetaan projektissa toimivat roolit ja tehtävät. Tehtäviä voidaan ulkoistaa, mikäli projektin sisäinen taito ja osaaminen ei riitä työn suorittamiseen tai sen laatu voi kärsiä. Otetaan huomioon myös taloudellinen ja ajallinen näkökulma. Projektissa voidaan esimerkkisi käyttää pilvipalveluita, jotka ovat ulkoistettuja palveluntarjoajia. (Digi- ja väestötietovirasto 2014.)

Projektin määrittelyyn tehdään muutoksia toteutuksen alussa, mikäli hankinta ja kilpailutus -vaiheessa on ilmennyt huomioitavaa suunnitteluvaiheessa tehtyyn määrittelyyn. (Digi- ja väestötietovirasto 2014.)

Testausta tehdään koko projektin kehitysajan. Teknologiat, joihin projekti nojautuu saattavat muuttua tai niissä havaitaan tietovuotoja. Näiden seuraaminen on tärkeää ja niihin on tarvittaessa pystyttävä reagoimaan nopeasti. Testaus on erittäin tärkeää ennen tuotteen julkaisua erityisesti, mikäli palvelua tulee käyttämään eri sidosryhmät. Projektin edetessä tekijällä saattaa tulla kehittämiseen ”sokeus”, jolloin tämä ei välttämättä tule ajatelleeksi kaikkia mahdollisia lopputuloksia, mitä projekti joutuu käsittelemään. Testaukseen kannattaa

ottaa mukaan mahdollisen sidosryhmän edustajia, jolloin käytännönpuolen ongelmat tulevat paremmin ilmi ja loppukäyttäjä saa paremman käyttäjäkokemuksen. (Perälä 2016.)

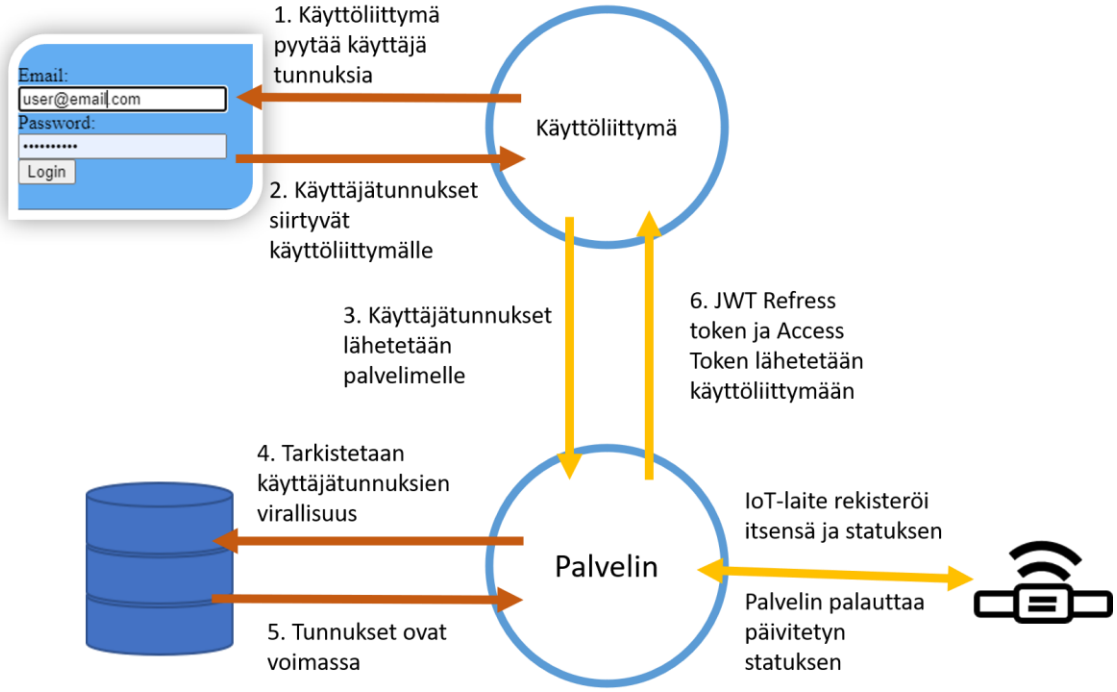
Aliluvuissa käyn läpi projektin etenemiseen tarvittuja tekniikoita sekä kuinka teoriassa eri osat olivat tärkeitä projektin toteuttamisessa. Jokaisessa tekniikassa pyrin tuomaan SoC-periaatteen tärkeyttä, jonka mukaan projektin rakenne on suunniteltu. Osien välillä pyritään pitämään kokonaisuudet mahdollisimman erillään, jolloin jokainen kokonaisuus on mahdollista vaihtaa eri tekniikkaan helposti. Projektin teorian dokumentointi selittää, miten eri tekniikat vaikuttavat projektissa. (Naumov 2020.)

3.3.1 JWT Applikaatiossa

Palvelimessa otin käyttöön jsonWebtoken-kirjaston, joka toimii JWT-standardia käyttäen. Tämän kirjaston avulla käyttäjän varmentaminen palveluun onnistuu kätevästi ja helposti. Se hoitaa tokenien luonnin, seuraamisen, varmentamisen ja myös niiden tuhoamisen. Jokainen token voi sisältää kryptattua tietoa, jonka tokenin luonut applikaatio voi avata omalla salaisella tunnuksellaan. Kirjaston luomassa tokenissa on kolme kerrosta tiedon tallentamiseen. Header sisältää tyypin ja kryptaus kielen. Payload sisältää applikaation käyttäjän tiedot, joita käyttöliittymä voi käyttää sille sallitun datan saamiseen tietokannasta. (JWT 2020.)

Datan saamiseksi on suositeltavaa tehdä kaksinkertainen varmenne. Tämä mahdollistaa kirjautumis-tokenin (Refress Token) tallentamisen selaimen lokaaliseen tietokantaan sekä sen käyttämisen uuden tokenin saamisessa. Tallennettava kirjautumis-token on JWT:n luoma kryptattu tunnus, joka tallennetaan myös palvelimen tietokantaan. Sen elinkaari on pidempi kuin tavallisen datan pyyntökäyttö-tokenin (Access token). Se ei sisällä käyttäjän tietoja. Sen avulla käyttöliittymä voi pyytää uutta tokenia, mikäli kirjautumis-token on vielä käyttökelpoinen. Uuden tokenin saaminen vaatii uudelleen tunnistautumisen palvelimeen. Kun käyttäjä kirjautuu ulos palvelusta, JWT-kirjasto tuhoaa käyttäjälle luomansa tokenin. (JWT 2020.)

Pyyntökäyttö-token on tarkoitettu käyttäjän oikeuksien pyytämiseen ja datan hakemiseen palvelimelta. Tällä tokenilla on lyhyt elinkaari, jonka jälkeen uuden pyytäminen vaatii kirjautumis-tokenin käyttämistä. Elinkaari voi olla minuutista muutamaan minuuttiin. Pituuden määrittäminen kannattaa mitata henkilön keskimääräiseen tarpeeseen palvelimessa. (JWT 2020.)



Kuva 2: Kirjautuminen ja käyttöoikeus IoT-laitteen hallintaan

Vaikka palvelu ei tulisi käyttämään IoT-laitteita tulevaisuudessa, kirjautumisjärjestelmää voi käyttää muissa projekteissa samalla tavalla ja se voidaan pitää erillään muista toiminnallisuuksista. Kerran luotua järjestelmää ei tarvitse kehittää uudelleen.

Viestinnässä applikaation rajapinnan kanssa on suositeltavaa, että liikkuva data on määritelty JSON-muotoon. Se mahdollistaa yksinkertaisen rakenteen datan siirtämisessä. Se myös ennaltaehkäisee rajapinnassa mahdollisesti tapahtuvien virheiden määrää. (Red Hat 2020.)

HTTPS-yhteys ei tarvitse domain-nimeä, mutta on suositeltavaa tehdä varmennus DNS-palvelun kautta, jolloin yhteyteen luodaan lisää turvallisuutta. (MDN Web Docs 2020.)

Kun kirjautuminen on luotu JWT:n avulla, on mahdollista siirtää tunnistautumislogiikka omaksi kokonaisuudeksi jatkokehitystä varten. Tämä ajatus on osa SoC-periaatteen käyttämistä. Projekti voi kehittyä omana applikaationa, mutta silti tarjota saman toiminnallisuuden datan saamiseksi palvelimelta. Samalla logiikalla toimii myös Googlen ja Facebookin kirjautuminen, kun käytämme toisen osapuolen sivustoa.

3.3.2 Tietokanta

Tietokannat antavat helpon tavan tallentaa kryptatusti muistiin paljon tietoa. Tämä myös mahdollistaa laitteen tunnistamisen, vaikka palvelin kaatuisi. Näin laitteet voivat jatkaa toimintaansa ilman uudelleen rekisteröintiä. Tämä myös pitää yllä vanhan tilan, mikä mahdollistaa laitteen toiminnan jatkumisen siitä, mihin se oli jäänyt. (SolarWinds MSP 2019.)

Applikaation ajonaikaista muistia pystyy myös käyttämään tarvittaessa, mikäli muistin käyttö on vähäistä. Ajonaikaista muistia ei pitäisi käyttää käyttäjätietojen tallentamiseen, mikäli niitä ei kryptata. Tämä on tärkeää ottaa huomioon jo applikaation Spike-vaiheessa, erityisesti silloin, kun applikaatio julkaistaan julkiseen verkkoon. Myös applikaation ajonaikainen muisti häviää, mikäli palvelimen tarjonta kaatuu tai palvelin käynnistyy uudelleen. Ajonaikainen muisti voi olla kallista, mikäli muistin käyttöä tulee paljon kerrallaan. Pilvipalveluntarjoajat hinnoittelevat sovelluksille annettua muistin käyttöä. Muistia voidaan käyttää pienissä projekteissa väliaikaisen tiedon tallentamiseen, jos applikaation kehitys on vasta Spike-tasolla. Nopeita testimuotoisia kokonaisuuksia ei aina kannata luoda valmiina. Mikäli projekti huomataan hyödyttömäksi, yritys on käyttänyt resursseja turhaan. (ProtectedIT N.d.)

Projektiin kannattaa tehdä yksinkertainen toiminnallisuus tietokannan ja rajapinnan väliin. Tämä toimii SoC-periaatteen mukaisesti työkaluna kahden eri tekniikan välissä. Työkalun idea on mahdollisuus vaihtaa tietokantaa. Keskustelu useamman eri tietokannan avulla mahdollistaa suurienkin muutoksien tekemisen. On järkevää ottaa huomioon applikaation tarpeet. Jos ei vielä tarkasti tiedetä, mitä tuote tulee tarvitsemaan, pystytään tietokantojen välillä tekemään jatkokehityksen kannalta tarvittavia muutoksia. (Cloud66 2016.)

3.3.3 IoT-laite

Projektiin otettiin käyttöön IoT-laite ESP32 WROOM-32 Development Board Module, joka toimii hyvin tarvittavassa testaamisessa. Laite käyttää pohjana Arduinoa, joka on helppokäyttöinen laitteisto ja ohjelmisto (Arduino 2020). Laite mahdollistaa Wi-Fi-yhteyden, jolla saadaan yhteys palvelimelle.

Datan siirrossa voidaan käyttää ESP32 luomaa HTTPS-kutsua, jonka avulla voidaan siirtää laitteen sen hetkinen tila ja tarvittavat varmenteet laitteen tunnistautumisessa. Palvelin vastaa, onko tilassa tapahtunut muutoksia. ESP32:n vastaanotettua vastauksen se suorittaa sille annetut toiminnot. Toiminnot tehdään vain, jos laite on lähettänyt pyynnön palvelimelle. Viestin mukana siirtyy laitteen oma GUID, joka toimii laitteen tunnistusliitteenä. Tämä GUID on manuaalisesti asetettu laitteelle. Viesti kryptataan SSL-tason avulla, joka huolehtii, ettei viestien välistä viestintää voi helposti purkaa. Tietoa laitteen ja palvelimen välillä ei ole välttämätöntä kryptata itse, ellei tiedonsiirto laitteelle suorita mahdollisesti haitallisia toiminnallisuuksia. Projektissa tämä ei ollut tarpeellinen toimenpide. (Wyliodrin 2015.)

ESP32 sisältää kätevän pienen näytön, joka mahdollistaa datan näyttämisen, ilman että laitteeseen tarvitsee juottaa erillisiä valoja. Näytöltä voi lukea, missä tilanteessa laite on, milloin laite lähetti pyynnön tai milloin se vastaanotti pyynnön. Myös mahdolliset virhetilat näkyvät näytöllä.

3.3.4 Versionhallinta ja PaaS-palvelu

Github on palvelu, joka on Microsoftin omistama. Se antaa käyttäjilleen laajan valikoiman versionhallintatyökaluja sekä välineitä projektien ylläpitämiseen ja seuraamiseen. Palvelu tarjoaa suoraviivaisen kehitysputken työkaluja, joilla voi asettaa projektien julkaisun eri pilvipalveluihin helposti. Pilvipalvelin tarvitsee vain SSH-yhteyden haluttuun arkistoon. Tämä arkisto voi olla salattu tai julkinen. Arkiston ja pilvipalvelimen välille voidaan asettaa automaattinen julkaisuprosessi. Se päivittää tehdyt muutokset pilvipalvelimen viimeisimpään versioon. (Github 2020.)

Projektissa voidaan käyttää Githubin-palvelun versionhallinnan julkaisutyökalua, joka voidaan päivittää aina uuden applikaatioversion yhteydessä. Tähän osioon voidaan kirjoittaa käytännöllisesti, mitä on tullut uutta, mitä muuttunut ja mitä korjattu edellisen julkaisun jälkeen. Tämän ideana on antaa projektin ulkopuoliselle ymmärrys, mitä on saatu aikaan ja missä vaiheessa projekti on. Julkaisuun voi lisätä tarvittaessa tiedettyjä bugeja, mutta nämä pitäisi korjata ennen julkaisua. Versionumerointi voidaan aloittaa semanttista versionhallintaperiaatetta käyttäen 0.1.0 -versiosta. Tuotetta voidaan jo käytännössä käyttää, mutta projektia ei kuitenkaan vielä ole tarkoitettu tuotantokäyttöön. (Gazar 2015.)

Github mahdollistaa useamman projektin versionhallinnan yhdessä palvelussa, josta voidaan yksinkertaisesti julkaista jokainen projekti. Tässä voidaan hyödyntää SoC-periaatetta, kun kirjautumiselle on oma API-palvelu ja käyttäjille on luotu oma käyttöliittymä. Myös datan hallitsemiseen tarkoitettu API voidaan asettaa omaksi kokonaisuudeksi, jolloin kaikki osat antavat yhden kokonaisuuden, mitä asiakas voi käyttää.

3.3.5 Testaus

Projektia on testattava kehitystyön aikana, manuaalisesti tai automaattisesti, jotta projektin edetessä ilmenevät virheet saadaan korjattua tai ennaltaehkäistyä. Manuaalinen testaaminen sopii jokaiseen projektiin. Automaattiset testit ovat kannattavia, jos samaa asiaa on tarkoitus testata useamman kerran tai halutaan tarkkailla, tapahtuuko kehityksessä regressiota. Ne pitävät huolen, että projektiin jo tehdyt osat toimivat myös kehitystyön edetessä. Koska automaattisten testien tekeminen vie aikaa, kannattaa projektissa olla tarve niiden käyttämiseen. Niiden käyttö voi pitemmällä aikavälillä säästää aikaa. (Tuovinen 2018.)

Projektin staattista testausta on järkevä suorittaa jokaisessa kehitysprojektissa. Se tehdään ilman ohjelmistokoodin suorittamista. Tarkoituksena on tutkia koodista suunnittelu-, logiikka- ja koodausvirheitä. Dynaamisessa testaamisessa taas ohjelmistoa ajetaan, jolloin testaaja tulkitsee koodia sen ajoaikana. Tässä voidaan käyttää työkaluina koodausympäristöjen debuggeri-työkaluja. (Tuovinen 2018.)

Node.js-projektissa kannattaa ottaa käyttöön Typescript. Typescript on ohjelmointikieli, joka toimii javascriptissa oliokielenä. Typescript auttaa luomaan javascript-koodin SoC-periaatteen mukaisesti. Se auttaa henkilöä jakamaan koodin omiin itse päätettyihin kokonaisuuksiin. Typescript tyypittää koodin ja tekee javascriptistä tyypitetyn oliokielen. Tämä auttaa koodin kirjoittamisessa ja ilmoittaa reaaliaikaisesti, mikäli koodissa on mahdollisesti tapahtumassa virhe. Omien objektien rajapintojen asettaminen auttaa muita käyttämään tekijän tekemää koodia. Typescriptin avulla voidaan antaa objektille oma ulkoasu. Siinä määritellään, mitä objekti voi sisältää, mitä sillä pitää olla ja mitä se voi antaa. (Sharma 2018.)

3.4 Käyttöönotto

Ennen käyttöönottoa on tärkeää testata tuotetta mahdollisella asiakkaalla tai asiakasryhmällä, jotta mahdolliset toimintavirheet ja käytettävyysongelmat voidaan havaita ajoissa. Testauksella varmistetaan, että palvelun julkaisuun on varattu tarpeeksi resursseja mahdolliselle asiakaskunnalle, palvelu on nopeasti ja helposti skaalautuva tilanteen muuttuessa. Testausta pitää tehdä paljon eri laitteilla ja käyttäjillä. Palvelun rasittamisessa mitataan, kuinka paljon palvelu tai pilvipalvelu kestää kävijöitä ja tehdään siitä jatkosuunnitelmia. Suunnittelu jatkotoimille on hyvä tehdä ajoissa, jotta mahdolliset seisonta-ajat olisivat pienet. (Digi- ja väestötietovirasto 2014.)

Ennen julkaisua on tärkeää tarkistaa, että applikaation teknisten osien dokumentaatio on tehty kattavasti ja julkaisusta on tiedotettu siitä riippuvaisille sidosryhmille. Sopimukset ovat ajan tasalla ja tuote on käytettävyyden ja esteettömyyden vaatimuksien mukainen. (Digi- ja väestötietovirasto 2014.)

4 Projektin toteuttaminen käytännössä

Projektissa toimeksiantajalla oli toive saada hänen valitsemansa IoT-laite toimimaan palvelimella julkisessa verkossa. Toimeksiantajan kanssa kävin läpi mitä projektin lopputulokseksi haluttiin. Projektissa otettiin käyttöön vesiputousmalli, jonka avulla pystyin suorittamaan työn pääsääntöisesti itsenäisesti ja tekemään heti alussa suunnitelman tavoitteen toteuttamiseksi. Ketterän kehityksen menetelmät olisivat voineet olla mahdollinen valinta, mikäli projektin tiimi olisi ollut isompi. Projektin tekniset päätökset hoidin itse ja tiedotin niistä toimeksiantajalle.

Vesiputousmalli sopi hyvin projektiin, sillä vaatimukset projektille oli määritelty jo alussa ja niiden muuttaminen matkan varrella ei ollut tarpeellista. Vaatimuksia olivat halpa, yksinkertainen ja helppo sisäistää. Projektissa tehtiin muutoksia suunnitelmaan vain tarkentavasti, sillä ohjelmistojen ja kehitysympäristöjen kanssa ei tarvinnut tehdä muutoksia.

Projektille ei määritelty asiakasta. Projektin suurin sidosryhmä oli eri palvelujentarjoajat, joiden tarjoamia palveluita vertasin. Vertailuna toimi palvelujen hinnoittelu, sillä lähes kaikki tarjosivat samankaltaisen palvelun. Palvelimen valitsemiseen tärkeitä mitattavia arvoja olisivat suorituskyky, saatavuus, turvallisuus sekä tuki- ja ylläpitomahdollisuus. Nämä arvot eivät olleet tässä projektissa tarpeellisia haluttuun lopputulokseen nähden. Projektia ei sidottu yhteen palveluntarjoajaan, jolloin se voidaan tarvittaessa siirtää sitä parhaiten palvelevaan jatkokehityksen yhteydessä. (Lehtinen N.d.)

Hain aineistoa netistä englanninkielisillä hakusanoilla, koska niillä löysin aineistoa kattavammin. Työn vaiheet suoritin pääosin itsenäisesti. Toimeksiantaja auttoi IoT-laitteen käyttöönotossa ja opasti, kuinka laitetta käytetään Arduino-ympäristössä.

Projekti oli laaja, sillä siinä luotiin SaaS-tyyppinen applikaatioinfrastruktuuri. Rakenne koostui palvelimesta, jonka ympärillä koko applikaatio toimi. Käyttöliittymän avulla käyttäjä voi ohjata IoT-laitetta palvelimen välityksellä. Tähän lisäsin tietokannan turvaamaan käyttäjätunnuksia ja mahdollisia IoT-laitteen hallintatietoja. Loin IoT-laitteen kommunikoinnin verkon ylitse palvelimelle ja takaisin. Lopuksi tein semanttisen versioinnin julkaistavasta applikaatiosta ja julkaisin applikaation palvelimeen.

Aliluvuissa käyn läpi projektin osat kehitysvaiheittain. Jokaisessa vaiheessa kerron, mitä huomioita projektin aikana tein ja miksi päädyin toteuttamaan tämän kokonaisuuden. Projektin esittely etenee vesiputousmallin ja teknologioiden käyttöönottojärjestyksessä.

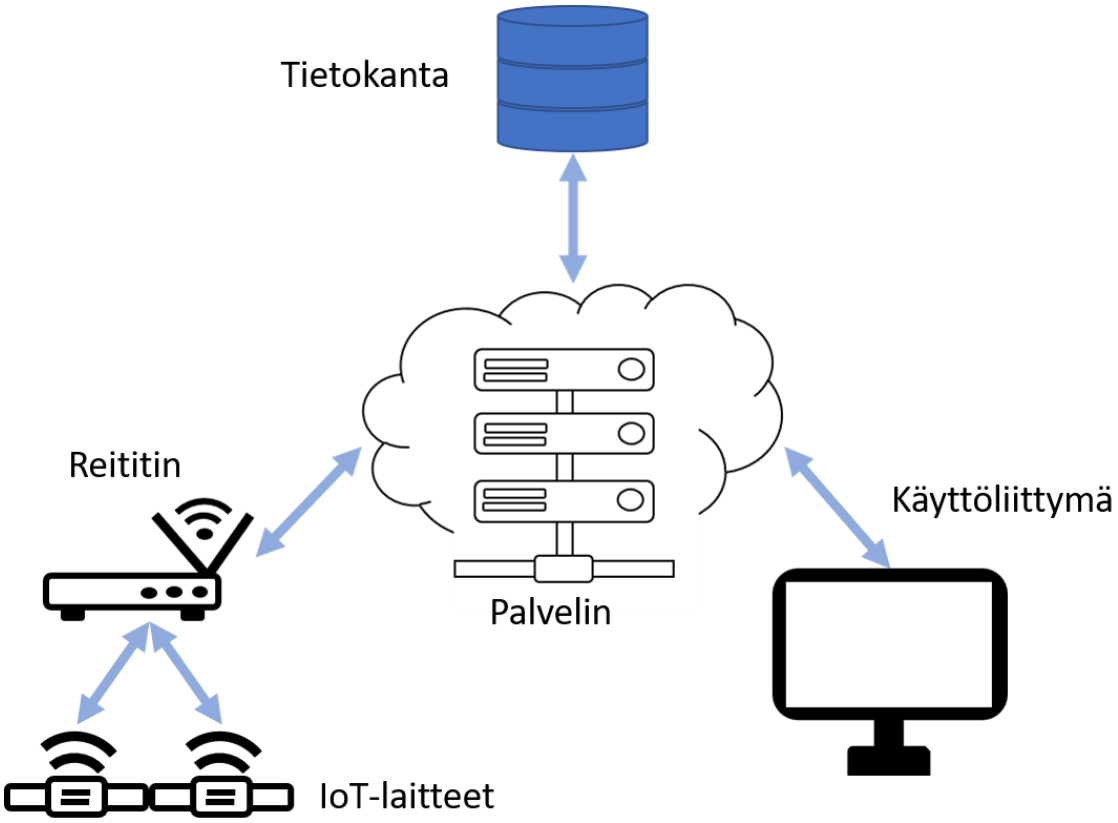
4.1 Suunnitelman rakentaminen

Suunnitelmaa varten tutkin, miten konsepti tulisi rakentumaan. Katsoin, mitä tekniikoita ja palveluja tarvitsen ja mitä asioita minun pitää opiskella. Käytin Googlea ja Youtubea pääsääntöisinä lähteiden hankintavälineinä. Hakusanoina teknologiapinon etsimiseen olivat fullstack, IoT-device ja web development. Sanojen avulla sain muutamia eri teknologiapinoja, joista minulla oli suurimmalta osalta jo aikaisempaa kokemusta.

Projektia halutaan mahdollisesti jatkokehittää, joten otin siihen SoC-näkökulman. Sen avulla pystyin ajattelemaan jokaisen osan omana moduulina. Nämä moduulit jaottelin omiksi kokonaisuuksiksi, jotka sisälsivät yhden tai useamman tekniikan. Nämä moduulit muutin myöhemmin pienemmiksi kokonaisuuksiksi. Rakenne pysyisi kasassa, vaikka haluaisin vaihtaa tietokannan toiseen teknologiaan tai haluaisin irrottaa kirjautumisjärjestelmän omaksi palveluksi.

Toimeksiantajan kanssa asetimme laatukriteerit, jotka olivat teknisiä kriteereitä: tietoturvallinen, helposti omaksuttava ja vähäiset ylläpitokustannukset. Nämä asetimme arvioitaviksi osaksi projektin onnistumisen määrittämiseen.

Lähteinä rakenteen suunnitteluun ja toteuttamiseen toimivat kollegat ja verkko. Toteutuskeinoja oli useita. Niistä valitsin sellaisen keinon, jossa projektin pystyi toteuttamaan yksinkertaisesti, koska tuote tulisi toimimaan Spike-prototyyppinä.



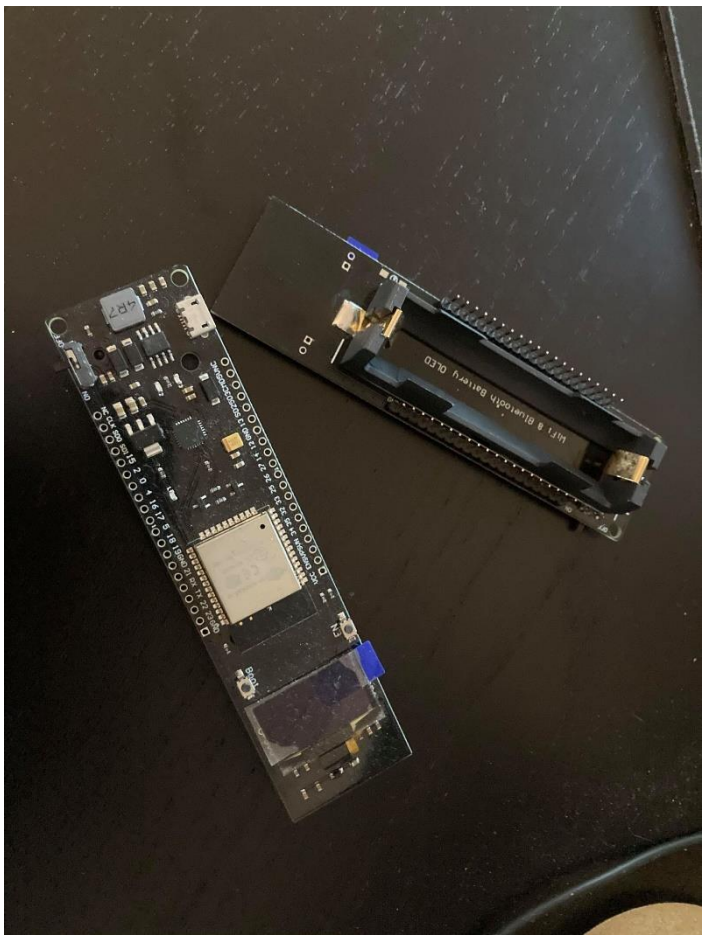
Kuva 3: Teknologioiden välinen viestintä

Kuvassa kaavio arkkitehtuurista palvelimen ympärillä (kuva 3). Välikätenä viestinnässä toimii palvelin, joka vastaa IoT-laitteiden lähettämiin HTTPS-pyyntöihin. IoT-laitteet käyttävät reititintä väliasemana verkkoon yhdistymisessä.

4.1.1 Pilvipalvelimien kilpailutus ja palvelujen IoT-laitteen hankinta

Projektin ainoa hankinta oli IoT-laite. Tämän toimeksiantaja oli tehnyt ennen projektin aloittamista, joten säästin aikaa laitteiden vertailusta. Laitteita on markkinoilla paljon ja jokaisella on omat hyvät ja huonot puolensa. Lähes jokaisesta mallista voi saada haluamansa kokonaisuuden pieniin projekteihin. Myös itse voi rakentaa kokonaisuuksia eri komponenteista, mikäli oikeaa ei löydy.

Välineet kehittämiselle minulla oli jo valmiina omasta takaa. Kilpailutuksessa vertailin eri palvelujentarjoajien tarjontaa ja hintaa. Kilpailutettava osa työn suorittamiseen oli pilvipalvelut, jotta applikaatio saataisiin julkiseen verkkoon pyörimään ja sitä voitaisiin testata ilman lokaalia toteutusta.



Kuva 4: ESP32 - WiFi Bluetooth Module With 0.96 Inch OLED Display 18650 Lithium Battery Holder Development Board

Projektin ainoa maksullinen osa oli kuvan mukainen ESP32-development board (kuva 4). Käytetyt PaaS-palvelimet olivat ilmaisia, sillä niitä käytettiin niin pienimuotoisesti. Koska projektin ei ole tarkoitus palvella suurta joukkoa ihmisiä tai laitteita, voidaan datan määrän pitää pienenä ja näin ollen ilmaisena. Projekti kuitenkin pysyy helposti laajennettavana ja jokaista projektin osaa voidaan laajentaa moneen eri palveluun.

Tietokantoja on useita ja niitä voidaan käyttää etäyhteytenä tai suoraan palvelimen kanssa juuressa. Vaihtoehtojen kartoituksen päätteeksi päädyin valitsemaan MongoDB-pilvitietokannan. MongoDB Atlas tarjoaa 500 megatavua ilmaista tietokantatilaa, mikä on sopiva pienille prototyyppipalveluille. Tätä tietokantaa voidaan helposti tarpeen tullen laajentaa, mikäli sivusto otetaan asiakaskäyttöön. (MongoDB 2020.)

Vertailin PaaS ja IaaS -pilvipalvelutyyppien hintoja. Hintaero näiden kahden välillä oli pieni. Molemmissa palvelutyypeissä oli myös ilmaisia käyttöasteita. PaaS antaa käyttäjälle paremman ja suoraviivaisemman tavan julkaista omia projekteja automaattisesti. IaaS tarjoaa käyttäjälleen mahdollisuuden laajempaan applikaatiojärjestelmään, sillä käyttäjä voi valita tarvittavan käyttöjärjestelmän. Spike-ajatuksessa taas kyseisille ominaisuuksille ei ole tarvetta, sillä käytin jokaisella alustalla toimivaa Node.js-ympäristöä, joka mahdollistaa minkä tahansa alustan käyttämisen. (Ahonen 2014.)

IoT-laite pystyi käyttämään Wi-Fi- ja Bluetooth -ominaisuuksia, joiden vertailussa päädyin ottamaan Wi-Fi-yhteyden. Se oli yksinkertaisempi ja helpompi keino toteuttaa viestintä. Teknologioiden välillä oli eroavaisuuksia datansiirrossa ja kantavuudessa, mutta molemmat pystyivät suorittamaan vaadittavan määrän ongelmitta. Mikäli laitteita olisi useampi samalla alueella, molempien teknologioiden käyttö samanaikaisesti olisi järkevää. Tässä tilanteessa ”Mesh” protocollien käyttö olisi suuri tekijä. (Behrtech 2020.)

Vertailtavia pilvipalveluita olivat AWS, Elastic Beanstalk, Windows Azure, Heroku, Force.com, Google App Engine, Apache Stratos ja OpenShift. Vertailussa otin huomioon dokumentaation laadun ja palvelun suoraviivaisuuden. Myös julkaisu pitäisi olla yksinkertaista GitHub-palvelimen kanssa. Kehitysputken pystyttäminen olisi hyvä olla mahdollisimman mutkatonta. Palvelujen vertailussa Herokuapp tarjosi yksinkertaisen ja suoraan Githubista julkaisun (Watts, Raza 2019). Herokuapp tarjosi samalla valmiin verkkotunnuksen. Kyseinen palvelu antaa ilmaisille käyttäjille oman aliverkkotunnuksen, jotka kaikki ovat HTTPS-protokollan alla. (MOZ 2020).

4.2 Työn toteutus

Asetin työlle tarkentavat vaatimukset, jossa otin mukaan teknisiä puolia. Nämä olivat SoC-periaatteen mukainen kehittäminen, tehokkaasti käytettävä versionhallinta ja laadukas dokumentaatio. Laitetukea en ottanut vaatimukseksi, sillä työ ei ollut visuaalisuuteen kohdistuvaa ja projekti toimii kaikilla Javascriptiä pyörittävillä selaimilla. Työn turvallisuusvaatimuksina olivat turvallinen teknologioiden välinen tiedonsiirto ja tietokannalla suojatut käyttäjätiedot. Dokumentaationa toimii opinnäytetyö ja koodiin tehdyt muistiinpanot.

Projektin kulku tehtiin vaiheittain, käyttäen SoC-periaatetta, jotta kokonaisuus pysyisi yksinkertaisina pieninä palasina. Tämän myötä projektin läpikäyminen olisi helpommin avattavissa ulkopuoliselle. Näin myös kokonaisuus pysyy järjestyksessä. Palaset eivät ole täysin sidoksissa keskenään, mikä mahdollistaa eri osien vaihtamisen toiseen helpoksi. Kokonaisuuksiin ei tarvitse tehdä suuria muutoksia ja tuotekehitystä ei tarvitse aloittaa puhtaalta pöydältä. (Naumov 2020.)

Projekti eteni vaiheittain ja otin käyttöön useita eri teknologioita, jotka mahdollistivat lopputuloksen. Teknologiat olivat ilmaisia ja vapaasti käytettäviä. Työn eteneminen on asetettu käsittelemään teknologioita niiden kehittämis- ja käyttöönottojärjestyksessä.

4.2.1 Github-versionhallinta

Projektissa piti ensin vertailla, minkälaisia teknologiapinoja olisi hyvä käyttää verkkopalvelimen luonnissa. Palvelimen ei tarvinnut olla täydellinen, vaan yksinkertainen prototyyppi ja helposti omaksuttava. Itsellä oli kokemusta monesta kokoonpanosta, mutta päädyin Node.js- ja Express-pohjaiseen kokoonpanoon, jolla loin ensimmäisen palvelimen lokaaliin verkkoon. Tämän projektin asetin Microsoftin ylläpitämään Github-palvelimen versionhallintaan, jossa toimeksiantaja voi seurata ja kokeilla valmista koodia. (Github 2020.)

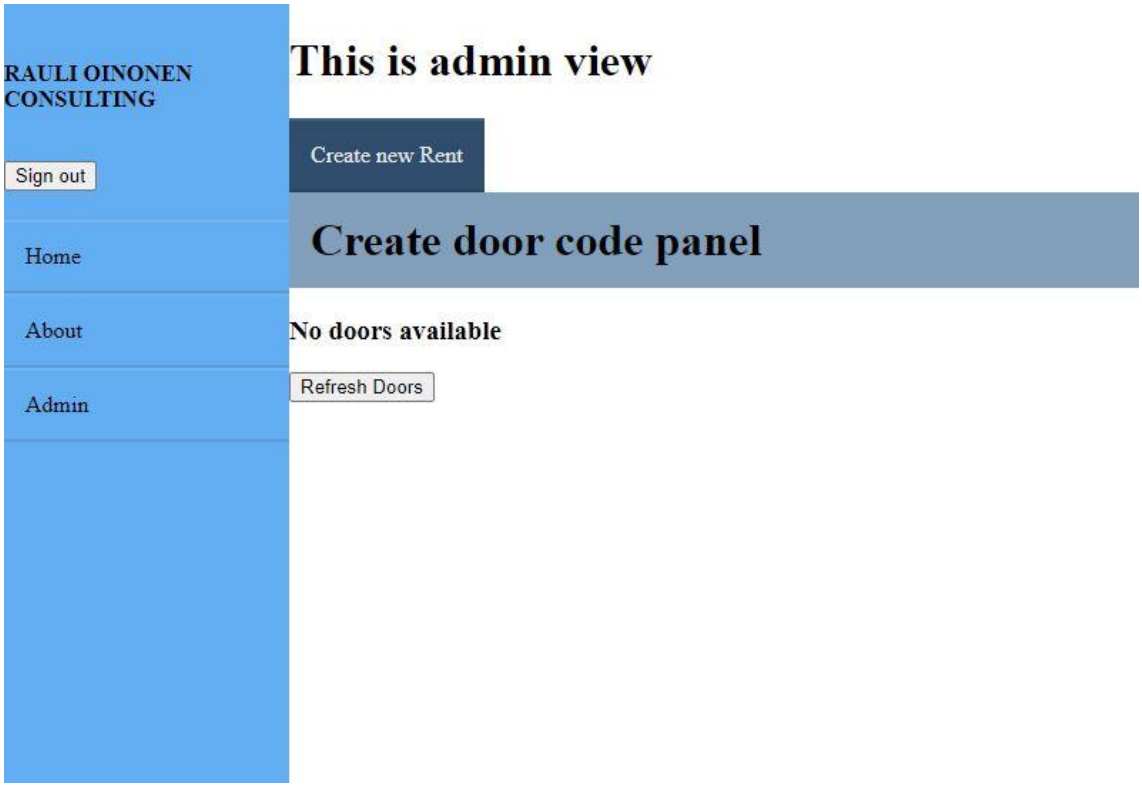
Etsin verkosta laajimman materiaalin ja dokumentaation omaavan paketin. Tämä mahdollisti toimeksiantajan tutustua itse tarkemmin palvelimen sisältöön. Projekti Github-palvelussa asetettiin yksityiseksi, jotta projektin salaiset avaimet eivät joutuisi vääriin käsiin helposti. Nämä avaimet voidaan aina tarvittaessa generoida uudelleen. (Github 2020.)

4.2.2 Käyttöliittymän kehittäminen

Käyttöliittymän luonnissa oli tärkeää valita mahdollisimman yksinkertainen runko eli framework, joka on samalla helppo käyttää ja mutkaton jatkaa. Itsellä oli aikaisempaa kokemusta VueJs:n käytöstä ja sen hyvästä dokumentaatiosta. Itse sivuston luontiin en paljoa käyttänyt aikaa, sillä visuaalisuus ei ollut tärkeä osa projektia. Visuaalisuuteen käytin vain toiminnallisuuksien mahdollistamiseen tarvittavan ajan. (TechTerms 2013.)

Jaoin sivuston omiin yksinkertaisiin komponentteihin, jotka edustivat kukin omaa osiota. Sivuston layout jakautui navigaatioon, header-osioon, footer-osioon ja main content-osioon. Näin sivuston ulkoasu pysyy kasassa, vaikka komponentteja muuttaisi jälkikäteen. Projektissa ei otettu huomioon puhelimen käyttöä, sillä sitä ei nähty näin alkuvaiheessa tarpeelliseksi lopputulokselle. Myös esteettömyyden asetin jatkokehitykseen. (Dolan 2016.)

VueJs toimi tehokkaasti komponenttien käyttämisessä, mikä antaa SoC-periaatteelle lisää arvoa. Jokainen komponentti hallitsee oman osan frameworkista ja nämä voivat sisältää omia pieniä komponentteja. Jokainen osa pysyy omana kokonaisuutena ja nämä huolehtivat omasta toiminnallisuudestaan.



Kuva 5: Käyttäjän käyttöliittymä

Kuva verkkosivujen käyttäjän käyttöliittymästä (kuva 5). Liittymän ulkoasu on yksinkertainen ja ei täten ole responsiivinen mobiilikäyttöön, mutta riittävä demonstroimaan tarvittavat toiminnallisuudet.

4.2.3 Tietokannan käyttöönotto kirjautumisessa

Projekti julkaistiin verkkoon. Palvelun kautta käyttäjät voivat hallita palveluun yhdistettyjä laitteita, minkä takia verkkosivuille piti luoda käyttäjän tunnistautuminen. Kirjautuminen mahdollisti laitteiden turvaamisen väärinkäytöltä.

Tietokantaa käytettiin käyttäjätunnistautumistietojen tallentamiseen ja kirjautumis-tokenien hallitsemiseen. Myös IoT-laitteen tunnistautumiset asetettiin tietokantaan, jotta laitteet voidaan turvata ja niiden väärinkäyttöä ehkäistä.

Tietokanta ja kirjautuminen luotiin yksinkertaisesti lokaalisti, kun käyttäjätunnus aluksi kovakoodattiin suoraan palvelimeen. Tietokannan lisäyksen jälkeen saatiin käyttäjätunnukset tietokantaan, mikä mahdollistaa tunnusten pysymisen paremmassa suojassa. Tietokanta toimii projektissa vain käyttäjätunnuksien hallitsemisessa ja pysyy kryptattuna palvelusta erillään. Palvelin ottaa erikseen tietokantaan yhteyttä, jolloin data ei ole palvelimen kanssa samassa pilvessä. Näin käyttäjätunnukset eivät leviä ulkopuolisten nähtäväksi (MongoDB 2020). Testauksissa applikaation yhteydenotto ulkoiseen tietokantaan ei tuottanut suurta viivettä.

Tietokannan liittämisen jälkeen applikaation muistiin kovakoodattu käyttäjätunnus pystyttiin siirtämään tietokannan muistiin. Tämä mahdollisti kirjautumisen testaamisen lokaalissa verkossa ja viive pysyi lyhyenä.

Käyttäjän tunnistautumisessa otin käyttöön JWT-standardin, jota käytin jsonWebtoken-kirjaston avulla. Tämä mahdollistaa yhden kirjautumispalvelun käyttöönottamisen ja käytön applikaation eri verkkosivuilla. (Peyrott 2020.) Palvelin asetettiin kahdella varmenteella toimivaksi: kirjautumis-token ja pyyntökäyttö-token. Tämä mahdollisti henkilön kirjautumisen uudelleen ilman käyttäjätunnuksen ja salasanan tallentamista selaimen muistiin. Vain kirjautumis-token tallennetaan. Se luodaan käyttäjälle uudestaan tämän kirjautuessa palvelimeen. Tokeniin asetetaan käyttöaika, ennen kuin se tuhotaan palvelimen muistista. Käyttöajan loputtua käyttäjä joutuu kirjautumaan tunnuksillaan uudelleen. (Peyrott 2020.)

Kirjautumis-tokenilla käyttäjä voi hakea pyyntökäyttö-tokenin, jolla hän voi tehdä palveluun muutoksia ja käyttää palvelua asetettujen oikeuksien mukaan. Pyyntökäyttö-tokenia ei tallenneta selaimen muistiin, joten ulkopuoliset eivät voi käyttää sitä. Tälle tokenille on asetettu lyhyt tuhoutumisaika. Pyyntökäyttö-tokenin voi uusia kirjautumis-tokenin avulla. (Peyrott 2020.) Pyyntökäyttö-tokeniin asetetaan käyttäjän oikeudet, joiden avulla käyttäjä voi pyytää palvelimelta hänelle kuuluvan datan.

4.2.4 Palvelimen rakentaminen

Otin palvelimeen käyttöön Typescriptin, joka toimi projektissa kätevänä työkaluna. Se auttoi tekemään valmiit objektit ja määrittelemään ne. Tyypitin funktioiden argumentit ja niiden palautukset. Tämä helpotti koodin kirjoitusta ja vähensi mahdollisia virheitä. Tein IoT-laitetta varten oman luokan ja määritin sen sisällön. Kun laite ottaa yhteyttä palvelimeen, palvelin vastaanottaa merkkijonon, jonka se deserialisoi objektin muotoon. Kun laiteelle halutaan lähettää uusi tila, tila serialisoidaan merkkijonoksi ja lähetetään takaisin laitteelle.

Projekti haluttiin saada käyttämään salattua HTTPS-viestintää, joka vaatii SSL-salauksen asettamisen itse. Voi myös käyttää pilvipalvelun tarjoamaa alinimeä, joka yleensä toimii vahvan domainin alla. Domain on Herokuapp-palvelussa asetettu HTTPS-yhteyden alle. (MDN Web Docs 2020.)

Palvelimeen otetaan yhteys ESP32:n avulla HTTPS-yhteyttä käyttäen. Viestinnässä käytetään JSON-formaattia, joka mahdollistaa palvelimen ja IoT-laitteen välisen kommunikoinnin selkeyden. Koska ESP32 käyttää C++ -koodikieltä ja JSON on JavaScriptissä käytetty formaatti, tarvitsemme C++ -kirjastoa, joka hoitaa XML-formaatin muutoksen JSON-formaattiin. Onneksi Arduinon kirjastosta löytyi valmis kirjasto prosessoimaan merkkijonoja haluttuun JSON-muotoon. Tämän avulla pystyi keskustelemaan palvelimen kanssa ilman suurta määrää työtä

ja koodia. Tämän myötä on myös mahdollista tarkistaa, että laitteelle tullut data on peräisin palvelimelta. (DroneBot Workshop 2020.)

Node.js:n vahvuus on NPM-pakettien (node package managerin) hallintatyökalu, joka helpottaa pakettien lataamista ja niiden lisäämistä projektiin. Projekti ei kasva suureksi myöskään versionhallinnassa, sillä paketit voidaan asettaa ”gitignoren” avulla poistamaan ne. Nämä paketit voidaan ladata aina, kun projekti otetaan uudelleen käyttöön, paketteja päivitetään tai applikaatio rakennetaan. Myös ”gitignoren” saa luotua valmiina Github-palvelusta.

IoT-laitteen tunnistautuminen palvelimeen onnistuu pienen GUID-tunnuksen avulla. Tämä tunnus on kovakoodattu eli kirjoitettu käsin laitteen koodikantaan. Tämä GUID pitää huolen, että palvelin tunnistaa, mikä laite lähettää dataa ja toimii samalla tunnisteena IoT-laitteelle. IoT-laite vastanottaa palvelimelta saapuvan paluuviestin ja tarkistaa tunnisteesta, että uusi tila on tarkoitettu ko. laitteelle. (TechTerms 2008).

4.2.5 IoT-laitteen käyttäminen palvelimessa

ESP32-piiri sisältää kaikki tarpeelliset ominaisuudet prototyypin tekemiseen, kuten Wi-Fi-verkon käytön. Prototyyppiä voi kehittää kalliimpaan ESP32-versioon, joka pystyy luomaan oman SIM-korttiyhteyden mobiiliverkkoon. Tämä tosin on maksullinen lisätoimi, jonka takia pysyin omassa Wi-Fi-verkossa. (DroneBot Workshop 2020.)

ESP32:n kehittäminen onnistuu Arduinon tai yleisiä C++ -kirjastoja käyttäen. Tässä työssä pystyin käyttämään Arduinon kirjastoa ja löytämään sitä kautta tarpeelliset toiminnallisuuden ajurit. Myös Arduinon kokoelmasta löytyi kätevä JSON-kirjasto, joka helpotti huomattavasti datan siirtämistä ja käyttämistä, sillä data pysyi objektimuodossa. (MDN contributors 2021.)

Tein IoT-laitteelle kolme erilaista kansiota, joissa jokaisessa oli yksi Arduino-ominaisuus. Ominaisuuksia olivat näytön käyttäminen, Wlan-verkkoon yhdistyminen sekä tilan lähettäminen palvelimelle ja sen vastaanottaminen. Lopuksi otin käyttöön HTTPS-yhteyden.

Näytön käyttöönottaminen oli aluksi haasteellista, sillä laitteen käynnistyessä se joutuu odottamaan näytön käynnistymistä. Tässä oli tärkeää huomioida, että laite tarvitsee sekunnin aikaa käynnistyä ja näyttää tuotemerkkiä. Kun laite oli suorittanut tämän vaiheen, näyttö piti tyhjentää ja kirjoittaa ruudulle haluttu ilmoitus. Tein tästä pienen funktion, jota kutsumalla voidaan asettaa näytölle uusi teksti. Arduinon kirjasto sisälsi lähes kaiken tarvittavan näytön käyttöönoton saamiseksi. Tämän jälkeen siirryin uuteen tiedostoon, jossa halusin kokeilla laitteen yhdistymistä Wlan-verkkoon.

Yhteyden luominen palvelimeen tapahtui ensin Wlan-verkkoon pääsemisen testaamisella. Tämän yhteyden luominen vaati reitittimen nimen ja myös salasanan asettamisen IoT-

laitteelle. Yhteyden luonti onnistui yksinkertaisesti Arduinon kirjastossa olevan esimerkkikoodin avulla. Testasin yhteyttä pinging-funktiota käyttäen kohdistaen tämän kutsun Google-palvelimelle. Näin tarkistin, että laite oli onnistuneesti päässyt verkkoon. Tämä testi ei jäänyt IoT-laitteeseen pysyväksi muutokseksi. (DroneBot Workshop 2020.)

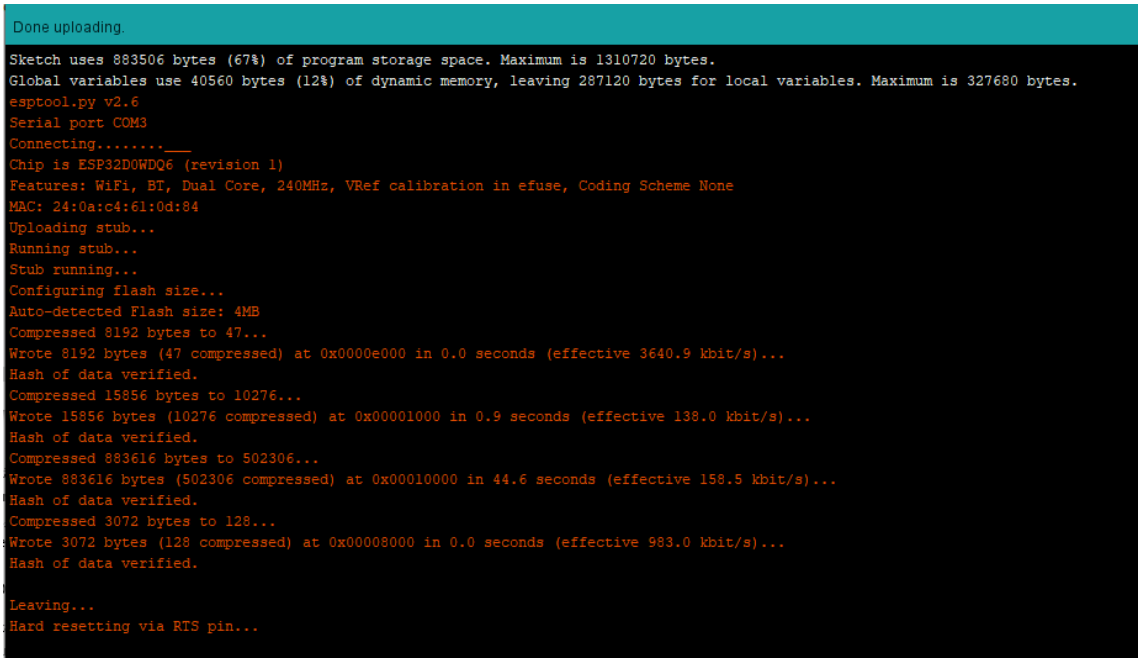
Kun olin saavuttanut laitteelle onnistuneen verkkoyhteyden, päätin yhdistää nämä kaksi ominaisuutta. Tämän jälkeen pystyin printtaamaan kaikki tarvittavat ilmoitukset laitteen näytölle.

```
String GUID = "4512-SDPW-5SDES-RSEQS";
```

Kuva 6: IoT-laite GUID

Onnistuneen yhteydenluonnin jälkeen aloin luomaan yhteyttä omaan palvelimeen. Palvelimeen luodaan oma rajapinta IoT-laitteelle, jossa laite ensin rekisteröityy. Rekisteröinnin yhteydessä laite saa tunnuksen, jolla sen voi tunnistaa palvelussa. Laitteiden tunnistautumista ei tallenneta tietokantaan, sillä tämän muistin pitäminen palvelimen kaatuessa ei ole välttämätöntä. (DroneBot Workshop 2020.)

Kuvan mukainen GUID asetettiin IoT-laitteelle kovakoodattuna ja tallennettiin palvelimelle ensimmäisellä yhteydellä (kuva 6). Palvelin käyttää GUID:a laitteen tunnistamisessa ja lähettää tämän GUID:n laitteelle takaisin paluuviestissä. Tätä laite käyttää varmistamaan, että paluuviesti on tullut samalta lähteeltä.



Kuva 7: Arduino IDE

Kuvan mukaisesti IoT-laitteen koodin puskeminen laitteeseen tapahtui Arduinon tarjoaman ympäristön avulla (kuva 7). Tämä ympäristö kääntää koodin ja pakkaa sen laitteelle sopivaksi koodiksi.

Palvelimeen yhdistymisessä käytin aluksi Postman-sovellusta, jolla pystyin testaamaan lokaalisti palvelimen Rest API -rajapintoja. Tietokoneen lokaalin IP-osoitteen onnistuin löytämään Windows-käyttöjärjestelmässä komentorivin kautta komennolla ”ipconfig”. IPv4 Address antaa koneen IP-osoitteen reitittimen antamassa aliverkossa. Tämä mahdollistaa reitittimen sisäverkossa olevien laitteiden keskinäisen yhteydenoton. Tämän lisäksi pitää antaa osoitteelle portti, jossa palvelin tarjoaa rajapintaa. Palvelimeen asetettiin portiksi 8080. (Postman 2020.) IoT-laitteelle asetettiin näytölle eri virhetilailmoituksia, jotta laitetta voidaan testata eri olosuhteissa ilman tietokoneeseen yhdistymistä.

4.3 Julkaisuprosessi

Projektin materiaalit julkaistaan Github-palvelussa. Tiedotteeseen kerätään projektin eteneminen ja liitetään projektin siihen asti luotu sisältö dokumentoituna. Tämä sisältää lyhyesti ja selkeästi, mitä päivitys tuo edelliseen päivitykseen nähden. Koska projekti luotiin vesiputousmenetelmää hyväksi käyttäen, projektista tehdään yksi julkaisu, joka sisältää kaikki Spike-suunnitelman määritellyt vaatimukset. (DeLuca 2016.)

Julkaisun tärkeä osa on sen testaaminen. Testaamisessa pyritään ottamaan huomioon mahdollisia tilanteita ja etsiä virhetiloja. Julkiseen verkkoon julkaistu palvelu toimii eri tavalla kuin lokaalissa verkossa toimiva. Tässä ympäristön muutoksessa voi tulla virheitä tai odottamattomia muuttujia. Työn virheet dokumentoidaan ja korjataan riippuen siitä, onko virhe palvelun toimivuutta rikkova tekijä. Dokumentaatiossa ei tarvitse antaa ratkaisua vaan esitetään ongelman ilmeneminen. Jatkokehitystilanteessa virhe voidaan korjata, vaikka tuote ei olisi enää sen kehittäjän vastuulla.

4.3.1 Versionhallinta ja palvelimen versioiminen

Versionhallinta suoritetaan Github-palvelussa. Palveluun tehdään jokaisessa valmiissa vaiheessa oma release-ilmoitus käyttäen ”git tags”-ominaisuutta. Tämän tagin avulla luodaan ilmoitus, mitä uutta projektissa on saavutettu, mitä korjattu ja mitä parannettu. Näin kuka tahansa saa tiedon yksinkertaisesti selitettynä.

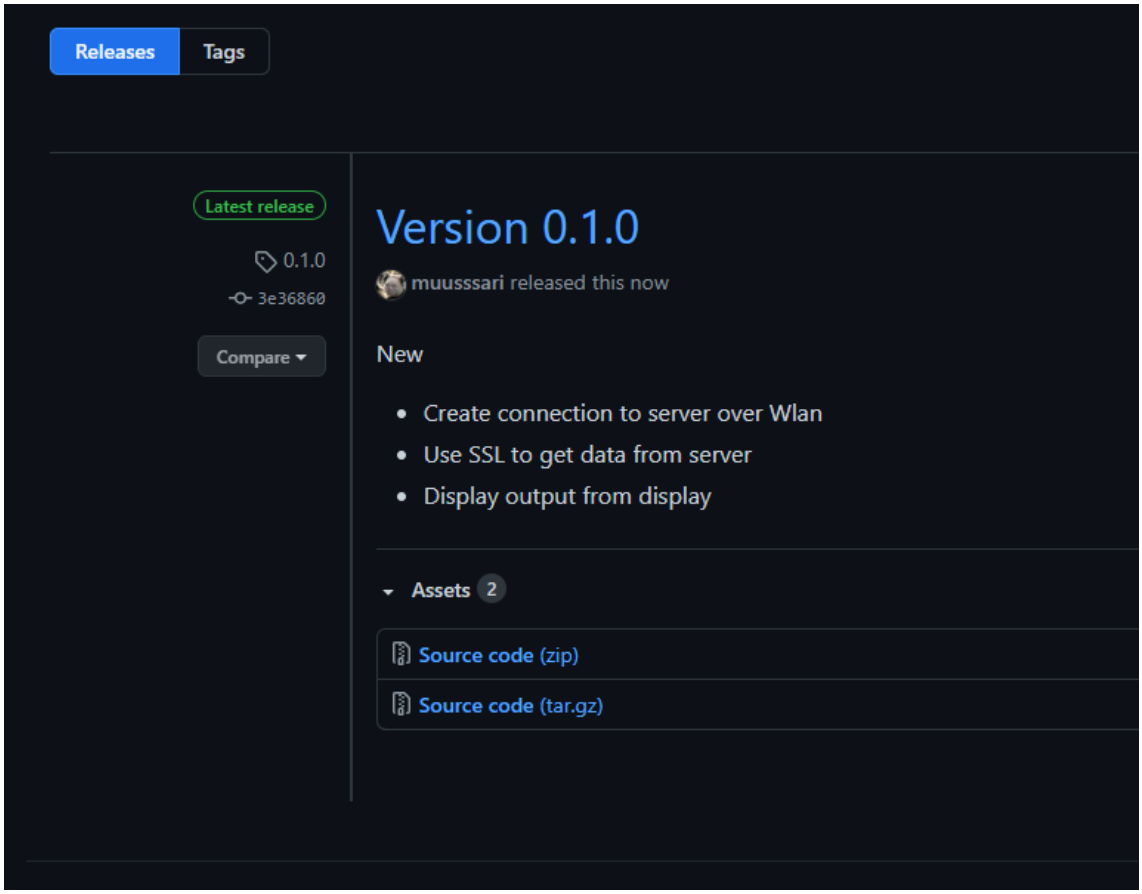
Versionhallinta auttaa myös uusia projektiin liittyviä henkilöitä nopeasti selvittämään, mitä on jo tehty. Tähän listaan voidaan lisätä löydettyjä bugeja tai ongelmia. Itse suosittelen näiden lisäämistä ”Issues”-osioon, jossa ongelma voidaan ohjata ko. vastuuhenkilölle. Tänne voivat myös muut projektin kanssa toimivat lisätä mahdollisia parannusideoita.

Projektiin voidaan tarvittaessa lisätä CI (continuous integration) -ominaisuuksia tarkastelemaan koodia ja käymään testit läpi automaattisesti. Tämä prosessi tapahtuu ennen Deploy-vaihetta, jolla voidaan turvata, että projekti on pysynyt kunnossa (Github 2021).

```
Muusssari@DESKTOP-5EE6HE4 MINGW64 /e/WebPages/Arduino_BoxClient (prod)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

Muusssari@DESKTOP-5EE6HE4 MINGW64 /e/WebPages/Arduino_BoxClient (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
g
Muusssari@DESKTOP-5EE6HE4 MINGW64 /e/WebPages/Arduino_BoxClient (master)
$ git add .

Muusssari@DESKTOP-5EE6HE4 MINGW64 /e/WebPages/Arduino_BoxClient (master)
$ git commit -m "init version control 0.1.0"
[master 3e36860] init version control 0.1.0
 1 file changed, 2 insertions(+)

Muusssari@DESKTOP-5EE6HE4 MINGW64 /e/WebPages/Arduino_BoxClient (master)
$ git tag 0.1.0

Muusssari@DESKTOP-5EE6HE4 MINGW64 /e/WebPages/Arduino_BoxClient (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

Muusssari@DESKTOP-5EE6HE4 MINGW64 /e/WebPages/Arduino_BoxClient (master)
$ git push origin master --tags
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 314 bytes | 314.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:WebDevelopmentJustCoala/Arduino_BoxClient.git
   dba3aa4..3e36860  master -> master
 * [new tag]         0.1.0 -> 0.1.0
```

Kuva 8: Gitbash tag-attribuutin käyttäminen

Julkaisussa asetetaan versionhallintaan kuvan mukaisesti Tag-merkintä (kuva 8). Tämä merkintä näkyy versionhallinnassa ja auttaa näkemään, milloin projektin julkaisu on suoritettu viimeksi. Tag-merkintä auttaa julkaisutiedotteen luomisessa siten, että oikea julkaisu tulee suoraan viitatuksi Githubin julkaisutiedoteosioon.



Kuva 8: Version julkaisuilmoitus

Kuvan mukaisesti projektista luodaan Github-palveluun release-ilmoitus (kuva 8). Tämä ilmoitus sisältää, mitä uutta, parannuksia ja korjauksia projektiin on lisätty edellisen version jälkeen. Versionhallintaan käytin semanttista versiointia, joka antaa periaatteen projektin numeroinnille.

4.3.2 Palvelun julkaisu PaaS-pilvipalvelimessa

Applikaatioon kirjautumista testattiin myös muilla laitteilla, jotta ongelmia ei tule myöhemmin. Koko tähän asti saatu projekti julkaistiin Herokuapp-palvelimen palvelussa, josta saa ilmaiseksi tarpeeksi muistia ja prosessointitehoa suorittaa tarpeelliset toimenpiteet. Se myös tukee kahden Node.js-projektin julkaisua samaan aikaan. Tämä mahdollistaa myöhemmän kehittämisen pitäen taustalla saman kirjautumislogiikan. (Heroku Dev Center 2020.)

Projekti julkaistaan suoraan GitHub-arkiston kautta, mikä helpottaa julkaisuputkea huomattavasti. Yksinkertainen putki auttaa toimeksiantajaa julkaisemaan omia projekteja itsenäisesti. (Heroku Dev Center 2020.)

5 Yhteenveto ja jatkokehitys

Opeteltavat asiat projektin aloittamiselle olivat IoT-laitteen käyttäminen ja toimintojen luominen. Niihin tutustuttuani projekti eteni harmittomasti. Työssä käytettiin paljon periaatteita ja eri teknologioita, jotka toimivat hyvässä harmoniassa keskenään. Tämän ansiosta oletan projektin etenemisen onnistuneen hyvin.

Vesiputousmallin valinta sopi projektiin, sillä sen periaatteet ja rakenne oli alusta asti oikeat tuotteen lopputulosta ajatellen. Vesiputousmalli sopii silloin, kun kyseessä on prototyypin luominen ja suunnitelma sen luomiselle on selkeä ja mahdollinen toteuttaa. Teknologiat olivat suurimmalta osalta tuttuja, joten työn rakenteen suunnittelu oli selkeä. Vesiputousmallissa ei tarvinnut tehdä muutoksia. Testaaminen olisi voinut olla suuremmin esillä työssä, mutta tähän projektissa ei ollut aikaa.

Teknologian dokumentaatio rakentui selkeiksi kokonaisuuksiksi SoC-periaatteen avulla. Tämän avulla jokaisen osan esittely helpottui, kun ominaisuudet pysyivät omassa paketissaan.

IoT-laite sopi projektin tarpeeseen luoda yhteys ja tehdä tarvittava visuaalinen toimenpide. Piirissä oli paljon muita mahdollisia ominaisuuksia, joita projektissa ei tarvittu. Ne ovat hyödyllisiä mahdolliselle jatkokehitykselle. Työ IoT-laitteella oli haasteellista C++ -koodikielen vuoksi, jota en ollut erityisen paljoa käyttänyt. Laitteen ja palvelimen välinen viestinnän salaus oli tuttua, mikä nopeutti projektin etenemistä. Tarvittavat toimenpiteet olivat valmiiksi dokumentoitu Arduinon opastuksessa.

Projektin vaikein osa oli keksiä keino, miten viestintä voitaisiin toteuttaa palvelimelle. Tähän löytyi kuitenkin nopeasti vastaus. IoT-laitteelle olisi voinut antaa enemmän ominaisuuksia. Arduino-ympäristön käyttöönottaminen oli hankalaa, sillä ESP32 vaati oikean alustan. Alustan etsiminen monesta kymmenestä eri mallista vei aikaa. Toimeksiantajalta sain apua oikean mallin löytämiseen.

Projektissa otettiin mukaan tietokanta, joka mahdollisti käyttäjätunnuksien tallentamisen kryptatusti kolmannen osapuolen ylläpitämään palveluun. Tämä helpotti palvelimessa hallittavien tietojen käyttöä ja turvallisuutta, koska palvelimen tietokanta on erillään muusta toiminnasta. Tietokannan jättäminen projektista suurentaisi riskiä käyttäjätunnusten joutumisesta vääriin käsiin. Projekti ei kuitenkaan käytä IoT-laitetta paljon, joten tietokannan tarpeellisuus ei ollut suuri. Tietokannan lisääminen oli pieni tehtävä ja tarpeellinen jatkokehitykselle.

Tekniset kriteerit saavutettiin hyvin. Projektin omaksuminen onnistui mielestäni hyvin. Toimeksiantaja sai hyvän kuvan, kuinka projekti on tehty, sekä miten sitä voi ja kannattaa jatkokehittää. Tarvittava tietoturva saavutettiin tiedonsiirron salaamisella ja dokumenttien

kryptaamisella. Ylläpitämisen mahdollistaa kehitysympäristön ilmaisten palvelujen käyttäminen. Kehittäminen ympäristössä ei tuota kustannuksia.

Tuotteen kaupallinen kehittäminen on vasta alussa. Nyt hyvän kehitysputken ja versionhallinnan avulla pystyy projektia kehittämään kuka tahansa projektiin lisätty henkilö. Työn jatkokehitykseen olisi hyvä miettiä CI-ominaisuutta, jolloin tuote pysyy ehjänä eikä riko verkkosivuja.

Github-palvelimeen voisi lisätä ”production”-haaran, joka olisi automaattisesti julkaistava. Tämä vähentäisi Herokuapp-palvelun käyttöä ja testausta voisi tehdä masterissa.

Käyttöliittymän kehittäminen on tärkeä osa, mikäli projekti tuotteistetaan. Tässä voidaan ottaa huomioon sivuston responsiivisuus eri laitteilla. Erilainen värikokonaisuus tekisi tuotetta kuvaavammaksi. Visuaalisointiin voi käyttää paljon aikaa, jota projektissa ei ollut riittävästi, eikä se ollut projektin pääkohde.

6 Arviointi

Projekti eteni hyvin eikä ilmennyt suuria ongelmia. Projekti pystyi saavuttamaan sille asetetut määritelmät. Projekti oli iso ja käsitteli monta eri osa-aluetta verkkoapplikaation kehittämisestä. Tässä vesiputousmalli toimi hyvin ja mahdollisti projektin rakenteen suoraviivaisen toteutuksen. Projektin laajuuden vuoksi joitakin asioita piti jättää pois tai kertoa muutamalla sanalla. Monet asiat olivat minulle itsestäänselvyyksiä, minkä vuoksi tarpeellisten tietojen dokumentoiminen tuotti vaikeuksia.

Projektin lopputulos oli tavoitteen mukainen prototyyppi. Projektia olisi voinut supistaa pienempään kokonaisuuteen jättämällä tietokannan käyttöönotto-osa pois.

Projektissa oli toiveena luoda kehitystä edistäviä aspekteja, jolloin tuotteen kehittämistä on turvallista ja mukava jatkaa. Projektiin voisi lisätä oman dokumentaation, jossa käydään läpi projektin rakenne niin päällisin puolin kuin myös itse koodin rakenteen kautta. Tässä dokumentaatiossa olisi hyvä esitellä koodausperiaatteet. Opinnäytetyö toimii hyvänä dokumentaationa projektille ja mahdollisesti sen lukijalle, mikäli henkilö itse haluaa luoda samankaltaisen projektin.

Projektiin syntyi hiukan teknistä velkaa. Kirjautumisjärjestelmä pidettiin muun toiminnallisuuden kanssa samassa palvelimessa, vaikka sen olisi voinut siirtää omaksi palvelimeksi. Käyttöliittymä on kuitenkin irrotettu muusta logiikasta ja asetettu omaksi kokonaisuudeksi.

6.1 Oma oppiminen

Opin erityisesti, kuinka helposti pienen laitteen voi liittää verkkoon ja kuinka paljon tämä mahdollistaa asioiden kehittämistä. Omaan kotiin voi näin luoda oman älykäskoti -toiminnallisuuden. Tämä tosin vie aikaa, mutta silloin rajana on vain oma ajankäyttö ja halukkuus toteuttaa itseään.

Projekti oli ensimmäinen lyhytmuotoinen prototyyppi, jossa käytin vesiputousmallia. Olen aikaisemmin käyttänyt kaikissa projekteissa ketteriä menetelmiä. Vesiputousmallin käyttö toi uutta kokemusta ja antoi eri näkökulman käyttää eri kehitysmenetelmiä.

Verkkopalvelimien kohdalla tuli tutuksi eri palveluntarjoajien hinnasto sekä selvisi, kuinka asioita voi toteuttaa ilman kustannuksia. Pienellä lisäkustannuksella palvelun voi helposti siirtää myös asiakaskäyttöön. Projektin jälkeen osaan pystyttää pienen SaaS-palvelun tai IoT-laite -pohjaisen harrastussivuston. Tehtyä tuotetta voin käyttää myös omassa portfoliossa. Tämän palvelimen voi muuttaa mm. oman pelin palvelimeksi pienillä API-muutoksilla.

Opin tarkemmin jo tuntemieni periaatteiden terminologiaa, joka liittyy projektien kehittämiseen. Opin kehittämisen menetelmien käyttöä sekä kuinka suunnitella hyvin projektissa tarvittavia osia ja vertailla eri teknologioita. Otin projektiin monta itselleni jo ennestään tuttua teknologiaa, ettei projektin kehittäminen yksinkertaisena prototyyppinä venyisi liikaa. Tässä olisin voinut kokeilla uusia tekniikoita oppiakseni uutta.

Opin erityisesti, että hyvällä suunnittelulla ja teknologiatutkimuksilla voidaan saavuttaa lyhyitä ja toimivia kokonaisuuksia.

Lähteet

Sähköiset

Wyliodrin. 2015. Introduction to the Internet of Things. Viitattu 20.9.2020. https://www.youtube.com/watch?v=G4-CtKkrOmc&list=PL38GvXivffy44qSMiqJlMFoYkBlgd_iK6&index=1

Ilmatieteenlaitos. 2020. Ilmatieteenlaitos ja ITM Finland selvittävät IoT-teknologian hyödyntämistä tiesään ennustamisessa. Viitattu 20.12.2020. https://www.ilmatieteenlaitos.fi/uutinen/7EElIOl71FTb9ftVRzWU1O

ESPRESSIF. 2020. ESP32. Viitattu 20.12.2020. https://www.espressif.com/en/products/socs/esp32

JWT. 2020. Introduction to JSON Web Tokens. Viitattu 20.12.2020. https://jwt.io/introduction

AuthO. 2016. Token Based Authentication Made Easy. Viitattu 20.12.2020. https://auth0.com/learn/token-based-authentication-made-easy/

Sobers, R. 2018. What is OAuth? Definition and How it Works. Viitattu 19.12.2020. https://www.varonis.com/blog/what-is-oauth/

Tutorialspoint. 2020. Node.js – Introduction. Viitattu 10.11.2020. https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm

Arduino. 2020. What is Arduino? Viitattu 10.11.2020. https://www.arduino.cc/en/guide/introduction

MDN Web Docs. 2020. An overview of HTTP. Viitattu 20.12.2020. https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

TechTerms. 2020. WLAN. Viitattu 20.12.2020. https://techterms.com/definition/wlan

TechTerms. 2021. WLAN. Viitattu 28.2.2021. https://techterms.com/definition/guid

Uy, M. 2020. Bluetooth Basics. Viitattu 20.12.2020. https://www.lifewire.com/what-is-bluetooth-2377412

Red Hat. 2020. What is a REST API? Viitattu 20.12.2020. https://www.redhat.com/en/topics/api/what-is-a-rest-api

SquareSpace. 2020. What is JSON? Viitattu 21.12.2020. https://developers.squarespace.com/what-is-json

Mistry, A. 2019. What Is Spike In Agile Software Development. Viitattu 10.12.2020. https://www.c-sharpcorner.com/article/what-is-spike-in-agile-software-development2/

Danicic, D. 2020. What is DevOps Pipeline & How to Build One. Viitattu 10.12.2020. https://phoenixnap.com/blog/devops-pipeline

Korpela, J. 2004. Mitä RFC:t ovat. Viitattu 10.12.2020. http://jkorpela.fi/rfct.html

Wpbeginner. 2020. Beginner's Guide: What is a Domain Name and How Do Domains Work? Viitattu 10.12.2020. https://www.wpbeginner.com/beginners-guide/beginners-guide-what-is-a-domain-name-and-how-do-domains-work/

MOZ. 2020. Domains. Viitattu 12.12.2020. https://moz.com/learn/seo/domain

Microsoft. 2020. Typescript. Viitattu 12.12.2020 https://www.typescriptlang.org/

Hertog, B. 2019. Different Ways to Connect IoT Devices to Transmit and Receive Data. Viitattu 12.8.2020. https://embeddedams.nl/different-ways-to-connect-iot-devices-to-transmit-and-receive-data/

Github. 2020. Where the world builds software. Viitattu 12.8.2020. https://github.com/

TechTerms. 2013. Framework. Viitattu 12.8.2020. https://techterms.com/definition/framework

Dolan, R. 2016. What is a UI template and why use one? Viitattu 12.8.2020. https://medium.com/claritydesignsystem/what-is-a-ui-template-and-why-use-one-e5d455ad64c5

Korpela, J. 2007. Standardi, mikä se on?. Viitattu 27.12.2020. https://jkorpela.fi/stand.html

MongoDB. 2020. Why MongoDB Atlas?. Viitattu 12.8.2020. https://www.mongodb.com/cloud/atlas

Peyrott, S. 2020. Refresh Tokens: When to Use Them and How They Interact with JWTs. Viitattu 7.12.2020. https://auth0.com/blog/refresh-tokens-what-are-they-and-when-to-use-them/

Heroku Dev Center. 2020. Getting Started on Heroku with Node.js. Viitattu 27.12.2020. https://devcenter.heroku.com/articles/getting-started-with-nodejs

DroneBot Workshop. 2020. Getting started with ESP32. Viitattu 12.8.2020. https://dronebotworkshop.com/esp32-intro/

Postman. 2020. What is Postman? Viitattu 20.10.2020. https://www.postman.com/

TechTerms. 2008. GUID. Viitattu 20.12.2020. https://techterms.com/definition/guid

Github. 2021. About continuous integration. Viitattu 10.1.2021. https://docs.github.com/en/actions/guides/about-continuous-integration

Gazar. 2015. How Version Control Numbering Works?. Viitattu 10.1.2020. https://ehsangazar.com/how-version-control-numbering-works-ea5c72a8bb1f

Digi- ja väestötietovirasto. 2014. JHS 190 Julkisten verkkopalvelujen suunnittelu ja kehittäminen. Tulostettu 27.2.2021. https://www.suomidigi.fi/sites/default/files/2020-07/JHS190_0.doc

Blomvist, H. 2018. Onko ketterän ja vesiputousmallin vertailu harhaanjohtavaa? Luettu 27.2.2021. https://blog.oppia.fi/2018/02/21/onko-ketteran-ja-vesiputousmallin-vertailu-harhaanjohtavaa/

DeLuca, J. 2016. 5 excellent product release note examples and how to write your own. Luettu 28.2.2021. https://www.appcues.com/blog/release-notes-examples

Koski, J. N.d. Ketterät menetelmät, agile, LEAN ja scrum. Luettu 27.2.2021. https://www.itewiki.fi/opas/ketterat-menetelmat-agile-lean-ja-scrum/

Lehtinen, V. N.d. Verkkosivuston uudistaminen. Luettu 27.2.2021. https://www.valu.fi/wp-content/uploads/2019/10/verkkosivuston-uudistamisen-pikaopas-2019.pdf

Behrtech. 2020. 6 Leading Types of IoT Wireless Tech and Their Best Use Cases. Luettu 27.2.2021. https://behrtech.com/blog/6-leading-types-of-iot-wireless-tech-and-their-best-use-cases/

Watts, S & Raza, M. 2019. SaaS vs PaaS vs IaaS: What's The Difference & How To Choose. Luettu 26.2.2021. https://www.bmc.com/blogs/saas-vs-paas-vs-iaas-whats-the-difference-and-how-to-choose/

Ahonen, T. 2014. Mikä ihmeen PaaS? Luettu 26.2.2021. https://www.cybercom.com/fi/Suomi/Yritys/Blogit/Blogit/Mika-ihmeen-PaaS/

Itewiki. N.d. Pilvipalvelut. Luettu 26.2.2021. https://www.itewiki.fi/opas/pilvipalvelut/

Sharma, A. 2018. Why You Should Use TypeScript for Developing Web Applications. Luettu 26.2.2021. https://dzone.com/articles/what-is-typescript-and-why-use-it

Naumov, A. 2020. Separation of Concerns in Software Design. Luettu 29.2.2021. https://nalexn.github.io/separation-of-concerns/

Perälä, J. 2016. Asiakaslähtöisen testausautomaation toteutus. Luettu 1.3.2021. https://bitfactor.fi/fi/2016/04/08/asiakaslaehtoeisen-testausautomaation-toteutus

ProtectedIT. N.d. Runtime Application Memory Protection. Luettu 29.2.2021. https://www.protectedit.net/runtime-application-memory-protection

SolarWinds MSP. 2019. Types of Database Encryption Methods. Luettu 29.2.2021. https://www.solarwindsmsp.com/blog/types-database-encryption-methods

MDN contributors. 2021. JSON. Luettu 27.2.2021. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON

Preston-Werner, T. 2020. Semantic Versioning 2.0.0. Luettu 29.2.2021. https://semver.org/

Random Nerd Tutorials. 2020. ESP32 HTTP GET and HTTP POST with Arduino IDE (JSON, URL Encoded, Text). Luettu 23.12.2020. https://randomnerdtutorials.com/esp32-http-get-post-arduino/

Helsingin Yliopisto. 2009. Ohjelmistoprosessit ja ohjelmistojen laatu. Luettu 28.2.2020. https://www.cs.helsinki.fi/u/taina/opol/k-2009/pdf/luku-6_2.pdf

Turotialspoint. 2021. SDLC - Waterfall Model. Luettu 28.2.2021. https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm

Santos, J. 2020. Agile vs. Waterfall: Differences in Software Development Methodologies? Luettu 2.3.2021. https://project-management.com/agile-vs-waterfall/

Tuovinen, A. 2018. Staattinen testaus Dynaaminen testaus 1: luento 3. Ladattu 8.3.2021 https://docplayer.fi/storage/91/104807413/1615228611/umP4A_0ZFPMBlNOwoKFmuQ/104807413.pdf

Cloud66. 2016. 3 Tips for Selecting the Right Database for your App. luettu 6.3.2021. https://blog.cloud66.com/3-tips-for-selecting-the-right-database-for-your-app/

Long, L. 2015. What RESTful actually means. Luettu 7.3.2021. https://codewords.recurse.com/issues/five/what-restful-actually-means

Martin Fowler. N.d. Data Transfer Object. Luettu 7.3.2021. https://martinfowler.com/eaaCatalog/dataTransferObject.html

Divine, P. 2019. Serialization and Deserialization. Luettu 7.3.2021. https://betterprogramming.pub/serialization-and-deserialization-ba12fc3fbe23

Laakso, M. 2010. PK-yrityksen tietoturvasuunnitelman laatiminen. Luettu 13.3.2021. http://urn.fi/URN:NBN:fi:amk-2010100813528

Skinner, J. 2020. Rapid learning and risk reduction with Feasibility Prototyping. Luettu 13.3.2020. https://www.thrivewearables.com/rapid-learning-and-risk-reduction-with-feasibility-prototyping/

Kuviot