Heidi Joensuu

Nettisivusto käyttöliittymä IoT laitteelle

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

1.11.2022

Tiivistelmä

Tekijä:	Heidi Joensuu
Otsikko:	Nettisivusto käyttöliittymä IoT-laitteelle
Sivumäärä:	39 sivua
Aika:	1.11.2022
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Osaamisaluepäälikkö Janne Salonen

Opinnäytetyössä tavoitteena oli luoda demonettisivusto IoT-laitteelle, eli kohdelaitteelle. Kohdelaite keräsi dataa, joka piti saada käyttäjille näkyviin. Pääpaino oli nettisovellukselle datan hankinta ja sen käsittely sekä käyttäjän tekemien parametrimuutoksien, tai ladattujen tiedostojen lähettäminen kohdelaitteelle. Lisäksi haluttiin luoda alustus erinäisille käyttäjätyypeille.

Tässä työssä käydään läpi, mikä on IoT ja samalla, mitä erinäisiä teknologioita internetyhteyksiin käytetään. Ilman internettiä, ei olisi IoT:ta ja erinäiset verkkoteknologiat vaikuttavat vahvasti, miten IoT-laitetta voidaan käyttää. Työssä selvitetään, miten perinteinen nettisivusto muodostuu; mikä on palvelin- tai selainpuoli tai api ja miten näitä voidaan kehittää tai käyttää. Lisäksi pohditaan, miten IoT-laitetta varten kehitetty nettisivusto eroaa perinteisestä nettisivustosta.

Kehitysprosessia käydään läpi erinäisiä keinoja tavoitteiden saavuttamiseen ja esimerkkeinä, miten niitä käytettiin. Projekti laittoi tutustumaan, miten kerättyä dataa kyettiin käsittelemään. Sekä pohtimaan, miten kommunikointia kohdelaitteen kanssa toteutettaisiin. Lopputuloksena saatiin toimiva demoversio nettisivustosta kohdelaitteelle.

Avainsanat: Nettisivusto, JavaScript, ReactJs, Node.js, IoT-nettikehitys

Abstract

Author:	Heidi Joensuu
Title:	A website interface for the IoT device
Number of Pages:	39 pages
Date:	1 November 2022
Degree:	Bachelor of Engineering
Degree Programme:	Information Tecnology
Professional Major:	Software Engineering
Supervisors:	Janne Salonen, Head of School

The goal in this thesis was to create a demo wesite for an IoT device, i.e. the target device.The target device collected data that was needed to be available and understandable for users. The main focus was to obtain the data and process it for website. There also were requirement for user to have availability to send parameter changes or downdload files to target device. In addittion, we wanted to create initialization for different types of users.

During this thesis, is reviewed what is Iot and what different technologies are used for Internet connections. Without the internet, there wouldn't be IoT. Various network technologies strongly influence how an IoT device can be used. The thesis explains how a traditional website is formed; what is server or client side or API and how these can be developed or used. In addition, this thesis considers how a website developed for an IoT device differs from traditional one.

The development process will be reviewed with various means to achieve the goals and how they were used with examples. The project introduced us to how the collected data could be processed. And made to consider how communication with the target device would be implemented. The result was a working demo version of the website for the target device.

Keywords: Website, JavaScript, ReactJs, Node.js, IoT Web development

Sisällys

Lyhenteet

1 Johdanto 1

2 Yhdistetty IoT 2

2.1 Yhteyskäytäntö 3
2.1.1 Bluetooth 3
2.1.2 Wi-Fi 4
2.1.3 Matkapuhelinverkko 5
2.2 Oikean yhteyskäytännön valinta 7

3 Mistä nettisivusto muodostuu 7

3.1 Palvelinpuoli 8
3.2 Ohjelmointirajapinta 10
3.3 Selainpuoli 12
3.3.1 Staattinen nettisivusto 13
3.3.2 Dynaaminen nettisivusto 14
3.3.3 Yksisivuinen web-arkkitehtuuri 16

4 IoT-verkkokehityksen eroavaisuus perinteiseen verkkokehitykseen 17

4.1 Haastava data 18
4.2 Mukautuva selainpuoli 19
4.3 Turvallisuus 20
4.4 Hybridikehitystiimit 21

5 Nettisivuston kehitys kohdelaitteelle 22

5.1 Tavoite 22
5.2 Käytetyt keinot 22
5.2.1 Sequelize 23
5.2.2 Datan muuntaminen 25
5.2.3 Kirjautuminen 27
5.2.4 Kommunikointi kohdelaitteen kanssa 29
5.2.5 Selainpuolen keinot 31
5.2.6 Redux 33

5.3 Testaus 35
5.4 Jatkokehitys 37
6 Yhteenveto 38
Lähteet 39

Lyhenteet

API *Application programming inteface*, ohjelmointirajapinta, joka sallii eri ohjelmien kommunikoinnin keskenään.

ASCII *American Standard Code for Information Interchange*, amerikanenglannissa tarvittavat kaikki merkit (kirjaimet, numerot, erikoismerkit, jne…) laitettuna tietokonemerkistöön.

BLE: *Bluetooth Low Energy*, tavallista Bluetoothia vähemmän sähköä kuluttavampi versio.

CA *Certificate Authority*, varmentaja, luotettu kolmasosapuoli, joka tarkistaa ja antaa nettisivustolle digitaalisen sertifikaatin, eli toisin sanoen antaa nettisivustolle salauksen.

CSS *Cascading Style Sheets,* porrastetut tyyliarkit, dokumentti, johon voi määritellä nettisivuston ulkonäköä koskevia tyyliohjeita, kuten kokoja, värejä, fontteja, jne…

DOM *Document Object Model*, dokumenttioliomalli, antaa keinot, miten luoda käyttäjän kanssa vuorovaikutteisia nettisivustoja.

GHz: Giga hertsi, tässä dokumentissa radiotaajuus.

HTML *Hypertext Markup Language*, hypertekstin merkintäkieli, hypertektillä luotu dokumentti, johon nettisivusto rakennetaan.

HTTP *Hypertext Transfer Protokol*, hyperteksin siirtoprotokolla, jota käytetään tiedonsiirtoon selaimen ja palvelintietokoneen välillä.

HTTPS *Hypertext Transfer Protocol Secure*, salausprotokollan avulla suojattu hypertekstinsiirtoprotokolla.

IoT: *Internet of things*, esineiden internetti, järjestelmiä, jotka on rakennettu toimimaan internetin välityksellä.

JSON *JavaScript Object Notation*, tiedostomuoto, jonka avulla tallennetaan ja välitetään tietoa ja dataa.

JWT *JSON Web Token*, standardin menetelmä, joka käyttää JSON:ia pohjanaan ja antaa keinoj käyttöoikeistietueiden hallinnoiseen.

LTE *Long Term Evolution*, 4G, matkapuhelinverkon tiedonsiirtotekniikka

LTE-M: *Long Term Evolution for Machines*, vähemmän sähköä kuluttavampi, kuin LTE. Suunniteltu IoT tai muun kaltaisia laitteteita varten.

NB-IoT: *Narrow Band-IoT*, IoT-laitteille suunniteltu matkapuhelinverkko. Kuluttaa vähän virtaa ja kuljettää vähän data.

ORM *Object Relational Mapping*, Olio-relaatiokuvaus, Ohjelmointitekniikka välittää data olio- ja relaatiokielten välillä.

REST *Representational State Transfer*, arkkitehtuurityyli, jonka avulla luodaan ohjelmointirajapintoja.

SOAP *Simple Object Access Protol*, tietoliikenneprotokolla, jonka avulla luodaan ohjelmointirajapintoja.

SPA *Single-page application*, yksisivuinen webarkkitehtuuri, keino luoda nettisivusto.

TLS *Transport layer Security*, salausprotokolla internetissä tapahtuvalle tiedonvälitykselle.

UI *User Interface*, käyttöliittymä, laitteistoissa osio, jonka kanssa ihminen kykenee kommunikoimaan

URL *Uniform Resource Locator*, määrittää, missä tietty tieto sijaitsee.

XML *Extensible Markup Language*, merkintäkielen standardi. Voidaan käyttää sekä tiedonsiirtoon, että tiedostomuotona tallentamaan dokumentteja.

1 Johdanto

Esineiden internet (eng. Internet of Things), eli lyhennetysti IoT, on yksiä kuumimpia kehittyviä teknologioita 2000-luvulla. IoT:lla tarkoitetaan laitteita, jotka ovat yhdistettyinä internettiin ja joihin on sisäänrakennettu sulautettuja järjestelmiä, kuten esim. erinäisiä sensoreita. IoT:n pääpainoisena määritteenä on, että ”esineet”, eli laitteet lähettävät ja vastaanottavat dataa internetistä. Ne kykenevät kommunikoimaan keskenään ja/tai ihmisen kanssa käyttöliittymän avulla. Ohjaavaa ihmiskättä IoT laitteet eivät tarvitse. [1]

Teollisuudessa yksiä yleisiä käyttöjä IoT laitteelle on monitoroiminen, joka on myös tämän insinöörityön käyttötarkoitus. Esim., sen sijaan, että joku työntekijä fyysisesti paikan päällä kävisi välillä tarkistamassa, vaikka veden tai jonkun raaka-aineen kulutusta, voidaan nykyään korvata IoT-laitteella, joka hälyttää, mikäli on tapahtunut ei-haluttuja muutoksia. Tiedot muutoksista saadaan reaaliajassa. Tämä säästää yrityksiltä paljon aikaa ja rahaa.

Tämän insinöörityön aiheena oli luoda kevyt verkkosivustopohjainen käyttöliittymä Comatec Oy:n tekemälle IoT-laitteelle, jonka yrityksen asiakas oli pyytänyt. Comatec Oy on n. 600 hengen yritys, jolla on IoT Solutions niminen osasto, joka nimensä mukaan on painottunut tekemään IoT laitteita. Projektiin osallistui muitakin henkilöitä, jotka tekivät IoT laitteeseen sulautetut järjestelmät, sekä verkkoyhteyden. Tehtäväalueekseni jäi puhtaasti nettisivuston tekeminen.

Määritteissä sivuston tulisi olla sisällytetty rakennettuun IoT-laitteeseen ja kommunikoida muiden prosessien kanssa. Kommunikoinnissa pääpainona oli laitteen lähettämän datan analysointi ja muuntaminen ihmisen luettavaksi, sekä kyetä lähettämään laitteelle parametreja sekä tiedostoja, joita laite käyttää prosesseissaan. Lisäksi nettisivustolla tuli olla keino kirjautua sisään ja pääkäyttäjän, eli Adminin tuli kyetä hallitsemaan muiden käyttäjien oikeuksia sivustolla.

IoT-laite, johon nettisivusto asennettiin, kutsutaan tässä tekstissä nimeltä kohdelaite. Asiakkaan nimeä taikka kohdelaitteen tarkkaa tarkoitusta ei kerrota tässä tekstissä. Kohdelaitteen suurpiirteinen tarkoitus oli analysoida saamaansa dataa ja muuntaa ne talletettavaan muotoon. Nettisivuston tarkoitus puolestaan oli muuntaa tämä talletettu data luettavaan numeraaliseen muotoon. Toteutuksessa käytettiin palvelinpuolella Javasriptin NodeJS:ää sekä ReactJS:ää selainpuolen luomiseen.

2 Yhdistetty IoT

Nettisivusto on se osa IoT-kehittämisessä, joka on vuorovaikutuksessa käyttäjän kanssa. Nettisivustolle vaihtoehtoisia muotoja ovat myös kännykkäapplikaatio tai laitteeseen rakennettu käyttöliittymä. Kaikki muodot on mahdollista kehittää kommunikoimaan sujuvasti keskenään.

Hyvänä esimerkkinä IoT-laitteesta kuluttajapuolella on nykyään tarjolla olevat erilaiset puettavat IoT-laitteet. Näistä osa kommunikoi käyttäjän kanssa, kuten esim. älykellot. Osa voi olla sellaisia, jotka eivät kommunikoi käyttäjän kanssa ollenkaan, esim. Polar sykevyöt tai Oura-sormukset. Käyttäjä itse ei voi suoraan kommunikoida sykevöiden tai Oura-sormusten kanssa, vaan hän kykenee saamaan tietonsa kännykkäsovelluksien avulla. Lisäksi samat laitteen kerätyt tiedot on mahdollista saada nettisivustolta.

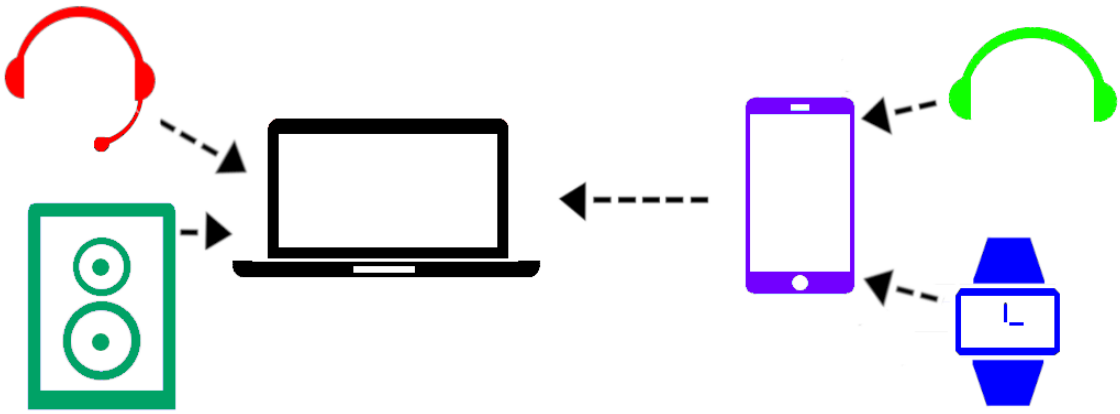
Kännykkäsovellus, nettisivusto ja puettavat laitteet ovat erilisiä osioita IoT-laitteessa. Niillä ei aina ole yhtenäistä fyysistä kontaktia toisiinsa, vaan ne kommunikoivat internetin välityksellä. Ilman kännykkäsovellusta käyttäjä ei kykenisi olemaan minkäänlaisessa yhteydessä Oura-sormukseen, vaan sormus olisi vain laite, joka keräisi dataa. Sykevyössä voisi olla jokin pieni pääte, eli näyttö, josta voisi käyttäjä nähdä tietojansa, mutta voi olla tilanne, että käyttäjä tarvitsee puhelimen, jotta hän näkisi saman datan kattavammin tai vaikka historiaa.

2.1 Yhteyskäytäntö

Yhteys internettiin on IoT:lle elintärkeä asia. Ilman sitä, esineiden internettiä ei olisi olemassakaan. Riippuen datatyypeistä, sovelluksista tai käyttötarkoituksesta, erinäiset yhteyskäytännöt, eli protokollat, ovat tarpeen. Protokollaa valitessa pitää ottaa huomioon erinäisiä ominaisuuksia, kuten esim. protokollan nopeus ja mahdolliset välimatkat. Lisäksi, etenkin sähköverkosta irrallisissa laitteissa, esim. akulla toimivissa, on hyvä ottaa huomioon valitun protokollan virran kulutus. Vuosien saatossa verkkoprotokollia on muodostunut, kehittynyt ja vakiintunut useita erilaisia. [2] Tässä dokumentissa käsittelemme kevyesti Bluetoohtin, Wi-fin sekä matkapuhelinverkon.

2.1.1 Bluetooth

Esim. yleensä puettavien laitteiden kohdalla laite itse on Bluetooth yhteydessä valittuun laiteeseen, kuten kännykkään. Bluetooth on kuluttajapuolella yleisesti käytettävä lyhyen kantaman protokolla, jonka avulla voidaan lähettää tietoja digitaalisten laitteiden välillä. Bluetooth käyttää vähän virtaa ja pystyy käsittelemään suhteellisen paljon dataa. Bluetoothista on myös uudempi versio: Bluetooth Low Energy (BLE), joka puolestaan kuluttaa paljon vähemmän virtaa, mutta myös käsittelee vähemmän dataa. [3] Bluetoothin yksiä huonoja puolia on sen lyhyt kantama. Yleinen kantama Bluetoothille on n. 10 metriä. [4] Näin ollen kommunikoivien laitteiden tulee olla lähietäisyydellä toisistaan.

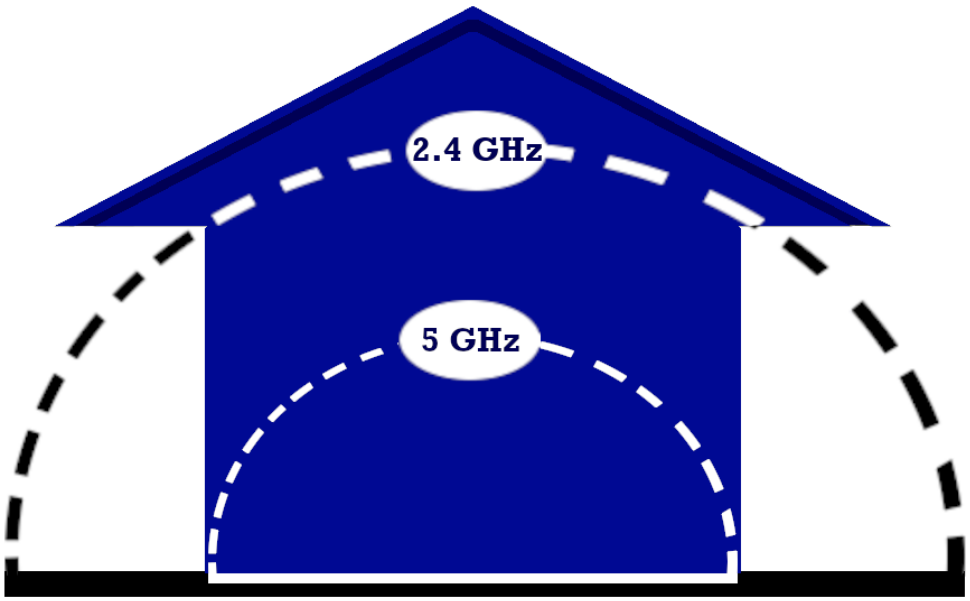


Kuva 1 Bluetooth yhteyksissä olevia laitteita. Yksi laite voi olla yhteen laitteeseen aktiivisessa yhteydessä.

2.1.2 Wi-Fi

Wi-Fi:ä yleensä käytetään tietokoneissa, tableteissa sekä kännyköissä. Suomen kielessä Wi-Fiä myös kutsutaan langattomaksi lähiverkoksi, tai paikallisverkoksi. Wi-Fi kuluttaa paljon virtaa toimiakseen ja sen yleinen kantama on n. 10-35 metriä., mutta ilman fyysisiä esteitä se voi myös kantaa 100 metrinkin päähän. [5] Mahdolliseen välimatkaan vaikuttaa myös gigahertsin (GHz) koko. Verkkoyhteyksien kohdalla gigahertsiyksikköä käytetään määrittelemään radiotaajuuksien taajuuksia [6]. Wi-Fi:ssä käytetään kahta taajuutta: 2,4 GHz sekä 5 GHz. Wi-fi:n kattavuus välimatkoissa on suurempi 2,4 GHz kuin 5 GHz, ja se kykenee läpäisemään joitakin kiinteitä kohteita, mutta myös samalla pystyy käsittelemään vähemmän ja hitaammin dataa. 5 GHz puolestaan toimii parhaiten lyhyissä välimatkoissa ja isoissa datamäärissä. [7]

Kumpikaan taajuus ei ole toista parempi, vaan ne sopivat omiin käyttötarkoituksiin hyvin. Esim. pelatessa tai verkon kautta televisio-ohjelmia katsoessa 5 GHz on parempi vaihtoehto, kun puolestaan 2,4 GHz riittää hyvin puhelimella viestittelyyn, tai yksinkertaisen älykkään kodinkoneen, kuten etänä ohjattava pyykkikoneen tai jääkaapin, toimintaan.

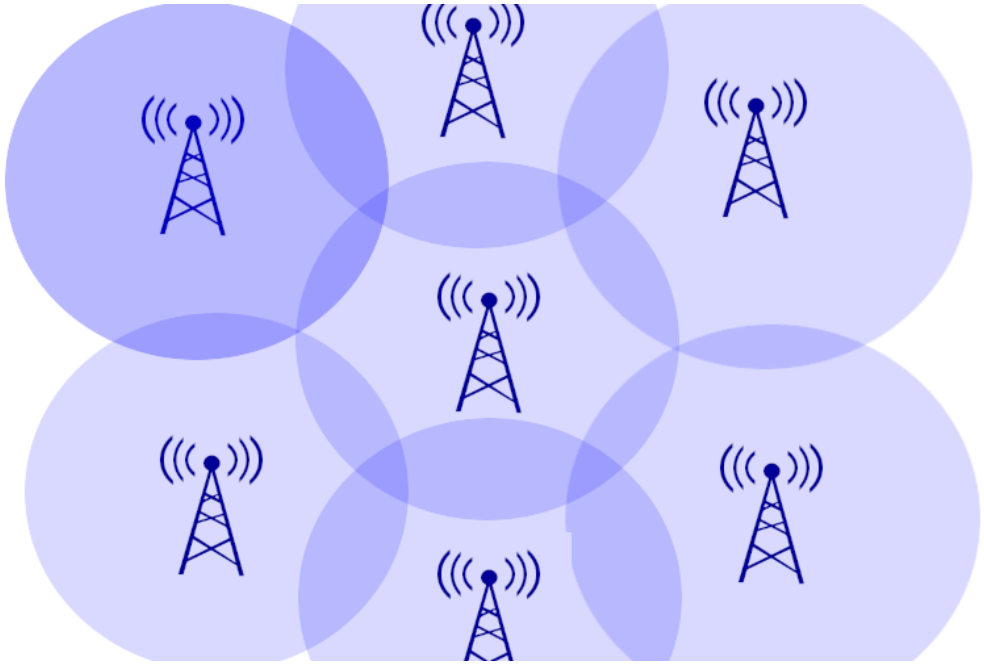


Kuva 2 Wi-fin 5 GHz kattaa lyhyemmän kantaman kuin 2,4 GHz, joka voi läpäistä seiniä ja toimia ongelmitta.

2.1.3 Matkapuhelinverkko

Matkapuhelinverkkoa voi käyttää missä tahansa IoT-laitteessa, jossa tahdotaan saavuttaa hyvä langaton internet-yhteys pitkien välimatkojen päästä. Aikaisemmin matkapuhelinverkon käyttö IoT-laitteissa oli paljon virtaa kuluttavaa, mutta nykyään on saatavilla hyviä vähävirtaisia vaihtoehtoja [8].

Matkapuhelinverkon suurin rahallinen hyöty yksittäiselle IoT-laitteen kehittäjälle on, että siihen tarpeelliset tukiasemat on jo rakennettu valmiiksi. Muilla verkkoprotokollilla ei ole käytettävänä yhtä laajaa saatavuutta. Ne ovat myös paljon tietoturvallisempia käyttää, kuin muut verkkoprotokollat. Samaan verkkoon voidaan yhdistää tuhansia eri IoT-laitteita, ja niillä ei tarvitse olla yhtä kattavaa virrankulutusvalmiutta toimiakseen, kuin esim. puhelimilla. Toisin kuin Bluetooth ja Wi-Fi, matkapuhelinverkkoon yhdistetyt laitteet toimivat ongelmitta liikkeellä ollessa. Yleisimmin kehittyneissä maissa matkapuhelinverkon tukiasemat on rakennettu tiheään, jolloin yhteys internettiin on katkeamaton. Näin ollen IoT-laitteita voi huoletta käyttää esim. autossa ajaessa kaupungista toiseen. [9,10]



Kuva 3 Matkapuhelinverkkojen tolppia pyritään rakentamaan luomalla aukottoman verkoston.

Matkapuhelinverkossa on erinäisiä teknologioita, joita voi käyttää internetyhteyden ottoon. Jokaisella on omat vahvuutensa. Parhaiten matkapuhelinverkko tunnetaan kännykkäteknologian kautta mainostetuilla 4G tai uudemmalla 5G sanoilla. G-lyhenne tulee englannin kielen sanasta generation, eli sukupolvi. Numero puolestaan kertoo, kuinka mones sukupolvi on kyseessä, eli ensimmäinen sukupolvi (1G), toinen sukupolvi (2G) ja niin edelleen, kunnes ollaan 2022 vuoden uusimmassa 5G sukupolvessa. [11]

2G ja 3G ovat poistuneet puhelinyhteyksistä, mutta niitä edelleen käytetään IoT-laitteissa. 4G puolestaan IoT-puolella mainitaan myös LTE-nimityksellä (Long Term Evolution). LTE-teknologiaa käytetään enemmälti mobiililaitteissa. Uusimmasta 5G:sta odotetaan suurta menestystä. Se kykenee käsittelemään huomattavasti enemmän dataa ja on nopeampi kuin aikaisemmat yhteydet. 5G on suunniteltu ottaen huomioon IoT-laitteet ollen muita sukupolvia joustavampi, eli sitä voi käyttää sekä isojen datojen siirtoon, että pienien. [12]

Nykyisin suuressa suosiossa ovat kuitenkin LTE-M sekä NB-IoT.

LTE-M (Long Term Evolution for Machines) on läheistä sukua 4G:n LTE teknologialle. Erona on, että kyseinen teknologia on suunniteltu enemmälti IoT laitteille. LTE-M käyttää samoja yhteyksiä kuin LTE, joten se sopii sellaisiin IoT-laitteisiin, joiden halutaan kommunikoivan mobiililaitteiden kanssa. Se kykenee välittämään sekä ääntä että dataa ilman suuria virran kulutuksia. LTE-M sopii parhaiten sellaisiin IoT-laitteisiin, joiden kohdalla on kriittistä, että ne välittävät täsmällistä dataa, jota saattaa olla hyvinkin paljon. Lisäksi LTE-M toimii reaaliaikaisesti. LTE-M on kuitenkin muista IoT-verkkoyhteyksistä yksi kalleimmista. [9, 10]

NB-IoT (Narrow Band-IoT) suunniteltu IoT-laitteille, joiden halutaan kuluttavan mahdollisimman vähän dataa. Se kuitenkin pystyy käsittelemään vain pieniä datamääriä. NB-IoT:n isoin etu on sen kykeneväisyys erittäin pitkille välimatkoille, luoden samalla erittäin luotettavan yhteyden. Lisäksi isoin etu NB-IoT:ssa on

sen äärimmäisen vähäinen virrankulutus. Hyvin suunniteltuna IoT-laitteessa oleva patteri voi kestää jopa kymmenen vuotta. [9, 10]

2.2 Oikean yhteyskäytännön valinta

Pääpainotteisesti kehitettävä laite itse määrittää, mitä yhteyskäytäntö vaihtoehtoja on käytettävissä. Aivan kaikkia protokollia ei pystytä asentamaan haluttuun laitteeseen. [13] Myös laitteessa käsiteltävän datan muoto vaikuttaa tähän. Virrankulutus tulee myös huomioida etenkin, jos laite toimii akulla tai pattereilla. Väärin valittu yhteyskäytäntö voi hetkessä kuluttaa laitteen virrat loppuun.

Joissain tapauksissa välimatkoihin tulee kiinnittää huomiota. Mikäli laite halutaan olevan lähiyhteydessä toiseen laitteeseen ja liikkuvia Bluetooth, on silloin edeltä edellä mainituista menetelmistä paras. Jos puolestaan laitteita on useita ja paikallaan olevia, Wi-Fi riittää. Pitkissä kantamissa kannattaa käyttää matkapuhelinverkkoja. Datan määrällä, nopeusvaatimuksilla on myös merkitystä sekä etenkin budjetilla.

Vaihtoehtoja on monia muitakin kuin tässä tekstissä pienesti mainitut. IoT-laitteen kehittäjän kannattaa tutustua erinäisiin vaihtoehtoihin ennen valintansa lukitsemista.

3 Mistä nettisivusto muodostuu

Nettisivusto on yksi visuaalisista keinoista ihmiskäyttäjälle päästä sisälle internettiin. Nettisivusto on oikeastaan palvelintietokone, eli serveri, joka on yhdistettynä internettiin. Näin ollen se on laite, jonka pitää aina olla toiminnassa, jotta nettisivusto on olemassa ja vierailtavissa. [14]

Yleinen virhetilanne, missä ei saada ladatuksi nettisivustoa, on oikeastaan tilanne, jossa nettipalvelinlaite on joko esim. sammunut, estynyt tai kaatunut, eli tapahtunut virhetilanne, josta se ei selviä ilman ihmisapua.

Nettisivustoja pystytään selaamaan URL:in (Uniform Resource Locator) eli web-osoitteen, tai toisella nimellä verkkotunnuksen, avulla. Jokaisella verkkosivustolla on oma uniikki osoite. Myös internettiin tallennetuilla tiedostoilla on omat web-osoitteensa. [15]

Nettisivuston sivut puolestaan ovat yhdistettyinä toisiinsa hyperlinkkien avulla. Hyperlinkki on sivustolla oleva elementti, joka voi olla esim. teksti, ikoni tai kuva, joka sisältää linkin toiseen nettisivuston sivuun.

Nykyisin nettisivusto yleensä muodostuu niin kutsutuista palvelin- ja selainpuolista. Nämä yhdessä tai yksinään palauttavat käyttäjän selaimeen nettisivuston.

3.1 Palvelinpuoli

Palvelin tunnetaan myös muista nimityksistä, kuten serveripuoli ja englanniksi backend. Palvelinpuoli on yleisesti vastuussa raskaista laskennallisista ja ohjelmallisista toiminnoista. Se saattaa määrittää nettisivuston sisällön, kuten ladattavat tiedostot, ja/tai on yhteyksissä talletusohjelmiin, eli tietokantoihin. Nettisivuston kävijä itse ei kykene olemaan yhteyksissä palvelinpuoleen, vaan tapahtumat ovat käyttäjiltä piilossa. Palvelinpuolelle on siis hyvä siirtää kaikki tiedot ja asiat, joita ei halua nähtäville selainpuolelle.

Palvelinpuolella tieto säilytettään yleisimmin tietokannassa. Tietokanta on työkalu, johon voidaan tallentaa haluamaansa dataa. Kaikki data voidaan sisällyttää tietokantaan; tekstit, erityyppiset numerot, käyttäjät, jne. Tietokantojen avulla voidaan helposti etsiä, järjestellä ja käsitellä sinne tallennettua sisältöä. Sieltä on myös helppo ladata tallennetut tiedot. Tässä projektissa käytetään tietokannan tarjoamaa taulukko ominaisuutta, johon riveihin listataan tietoa. Rivejä kutsutaan tietueiksi, eli yksi rivi on yksi tietue. Tietueessa on useita sarakkeita, eli kenttiä. Ne voivat sisältää kaikki omaa tietoansa ja tietotyyppinsä: henkilön nimi (teksti), syntymä (päivämäärä), onnennumero (numero) tai muuta tietoa. [16]

Palvelinpuolta voidaan ohjelmoida useammalla eri kielellä. Stackoverflow –sivuston vuonna 2021 tekemän kyselyn mukaan viisi nettisivustokehityksessä käytetyintä ohjelmointikieltä ovat JavaScript, Python, Java, C# sekä PhP [17]. Tässä projektissa käytimme JavaScriptiä.

JavaScript on yleisin ohjelmointikieli verkkokehityksessä, sillä se on yksinkertainen, joustava ja sitä voidaan käyttää sekä selainpuolella että palvelinpuolella. JavaScript on alun perin suunniteltu selainpuolen ohjelmointiin. Sen avulla luodaan erinäisiin näkyviin elementteihin toimintaa, kuten nappia painamalla saa vaihdetuksi näkyvää tietoa. Palvelinpuolella sitä voidaan käyttää kuin mitä tahansa muuta ohjelmointikieltä, eli laskennallisissa toimituksissa sekä toiminnallisena ohjelmointina. Toimiakseen palvelinpuolella JavaScript tarvitsee ajoympäristökseen Node.js:n. Node.js on asynkroninen, eli sen kanssa kommunikoivan ohjelman ei tarvitse olla ajallisesti tahdissa Node.js:llä luodun sovelluksen kanssa. Kommunikoiva ohjelma lähettää pyynnön, ei jää odottamaan vastausta, eli jatkaa omaa toimintaansa, ja vastaanottaa vastauksen, kun se tulee NodeJS:ltä. Node.js:llä tehty sovellus puolestaan tekee työtä ainoastaan kutsuttaessa ja muuten se on lepotilassa. [18]

Nettikehityksessä ohjelmointikielet käyttävät yleensä ohjelmistokehystä (engl. Framework). Ohjelmistokehys on kirjasto, joka valittuna vaihtoehtona kertoo, miten sovellus rakennetaan, eli antaa sovellukselle yhteisen säännön, miten sitä kehitetään [19]. Tämä nopeuttaa ohjelmointia huomattavasti, koska valmiin rakenteen avulla sovelluksen pohjakehitystä ei tarvitse siten luoda joka kerta uudestaan. Samalla kehittäjä voi keskittyä itse oman projektinsa toteuttamiseen kuin niihin ongelmiin, joihin ohjelmistokehys antaa jo valmiin vastauksen. Koska ohjelmistokehykset ovat entuudestaan kehitettyjä ja monen eri kehittäjien käyttämiä, ne ovat siten hyvin testattuja ja vakaissa versioissa yleensä virheettömiä ja turvallisia. Yksi ohjelmistokehys on yleensä suunniteltu tiettyä ohjelmointikieltä varten. Tämä voi olla ohjelmointikieli palvelin- tai selainpuolella. Samalla ne ovat tiettyyn ohjelmoinnissa osa-alueeseen painottuneita, kuten palvelimen

pystyssä pitämiseen tai selaimen ulkonäön luontiin. [20] Samalla ohjelmistokehykset tarjoavat työkaluja nopeuttamaan ja helpottamaan ohjelmointia, sekä reitit ykset, jota käyttäjä voi navigoida eri osioihin sivustolla. [19]

JavaScriptin Node.js käyttää paketinhallinta järjestelmää (engl. Package manager). Se on työkalu, jonka avulla kehitetyssä projektissa olevia ohjelmia, ohjelmistokehyksiä sekä kirjastoja pidetään listalla, päivitetään ja tuodaan uusia tai poistetaan.

3.2 Ohjelmointirajapinta

Ohjelman käyttämä data yleensä halutaan säilyttää palvelinpuolella. Jotta käyttäjät tai joku muu ulkopuolinen taho ei pääsisi dataan käsiksi, palvelin- ja selainpuoli käyttävät yhteistä ohjelmointirajapintaa, lyhennetysti API (Application programming inteface). Rajapinnan avulla mahdollisestaan eri sovellusten kommunikointi keskenään. Nämä voivat olla palvelinpuoli ja selainpuoli, kahden eri palvelinpuolen välillä tai haluttu kommunikointi muun kaltaisiin sovelluksiin. API määrittelee, miten nämä eri ohjelmat kommunikoivat keskenään. Sovellusten itse ei tarvitse olla kovinkaan tietoisia toistensa toiminnallisuuksista, vaan voivat keskittyä, miten API tarjoaa yhteisen kommunikoinnin. Tämä helpottaa kehittämistä, sillä eri sovellusten kehittäjät voivat keskittyä puhtaasti omaan osuuteensa ja tavoitteisiinsa. API on myös turvallinen keino luoda yhteyksiä. [21]

API:ien yhteydenmuodostus on yleensä joku seuraavista kolmesta: yksityinen, yhteistyö(kumppani) tai avoin. Yksityistä API:a käytetään, kun halutaan ainoastaan itse päästä käsiksi yhteyteen. Tämä voi olla esim. yksityishenkilön oma tai yrityksen sisäinen projekti. Yhteistyö API on esim. kahden eri yrityksen jaettu yhteinen API yhteisissä projekteissa. Toinen toimii yhteyden tarjoajana ja toinen sen vastaanottajana. Avoin puolestaan tunnetaan myös toisella nimellä; avoin rajapinta. Avoin on internetissä julkisesti kaikille näkyvissä ja käytettävissä.

Nettipohjaiset API:t pohjautuvat HTTP-kutsuille (Hypertext Transfer Protokol, eli hyperteksin siirtoprotokolla), ja ovat siten etäyhteysohjelmointirajapintoja. Nettisivustokehityksessä tyypillisesti selainpuoli lähettää pyynnön (request) palvelinpuolelle, joka puolestaan lähettää vastauksen (response). API:ssa on erilaisia keinoja luoda yhteyksiä, joista käymme tässä dokumentissa kolme:

SOAP (Simple Object Access Protol) on protokolla muotoinen API, jonka avulla luodaan ohjelmointirajapintayhteyksiä. Se on alun perin suunniteltu toimimaan kommunikointi keinona eri kielillä ohjelmoitujen ohjelmien välillä. SOAP sisältää rakennesääntöjä, joiden mukaan kommunikoivat sovellukset kehitetään. Tämä kuitenkin luo hieman monimutkaisuutta ohjelmointiin. Itsekommunikointi tapahtuu XML-tiedostoilla (Extensible Markup Language). [22]

REST (Representational State Transfer) toisella nimityksellä RESTful puolestaan on arkkitehtuurityyli, jonka avulla REST-kehittäjät luovat omat REST API:insa. Sille ei ole luotu virallista standardia, kuten SOAP:ille, vaan on paljon joustavampi toteutustavoissaan. Samoten se on paljon yksinkertaisempi käyttää kuin SOAP:ia. REST taipuu paljon kattavamman tiedon käsittelyyn. Sen läpi voidaan kuljettaa esim. tekstitiedostoja, JSON (JavaScript Object Notation) tai HTML-dokumenttia (Hypertext Markup Language, suomeksi hypertekstin merkintäkieli). REST:ia käytetään enimmäkseen netti- ja kännykkäsovelluksissa. [23]

GrapQL on uudenlainen API, joka on ns. suunniteltu REST:in kilpailijaksi. GraphQL on kyselykielimuotoinen. Samalla se on erittäin kevyeksi suunniteltu; hakee ja palauttaa ainoastaan niitä tietoja, joita kysyjä on pyytänyt. Muissa ohjelmointirajapinnoissa saadaan yleensä iso rakennepohjainen vastaus, josta pitää poimi haluamansa vastaus. Kwvywksi luodun haun ansiosta GraphQl:ää voidaan käyttää laitteissa, joilla on vain vähän virtaa käytettävissä. [24]

3.3 Selainpuoli

Selainpuoli, toisella nimeltään käyttöliittymä, tunnetaan myös englanninkielisellä nimeltään frontend. Kun palvelinpuoli toimii käyttäjältä näkymättömissä, selainpuoli on puolestaan näyttää kaiken saamansa tiedon käyttäjälle. Mitään, mikä on selainpuolella, ei voida piilottaa käyttäjältä.

Selaimessa näkyvä näkymä on HTML-dokumentin tekemä piirros. Se määrittelee nettisivuston struktuurin, eli rakenteen ja sisällön. Erilaiset elementit erotellaan toisistaan tunnisteen < > sisälle laitetulla merkinnällä (engl. markup), eli esim. <img> (kuva) tai <p> (kappale-elementti). CSS (Cascading Style Sheets, suomeksi porrastetut tyyliarkit) puolestaan käytetään, kun halutaan muunnella nettisivuston näkymää, eli vaihtaa fonttia tai värejä tietyistä valituista HTML-elementeistä. JavaScriptiä, kuten aikaisemmin mainittu, käytetään, jotta saadaan HTML-elementteihin toimintoja.

Selainpuoli voidaan tehdä kokonaan JavaScriptillä, joka generoi käyttäjän selaimeen HTML-dokumentin. Jotta JavaScript kykenee luomaan muuntuvaisen HTML-dokumentin, se tarvitsee avukseen DOM:in (Document Object Model, suom. dokumenttioliomalli). DOM on ohjelmointirajapinta dokumenteille. Dokumentit voivat olla esim. HTML tai XML. Dom määrittelee, kuinka voidaan muuttaa näkyvän HTML-dokumentin rakennetta, tyyliä sekä sisältöä ilman, että koko dokumentti pitäisi luoda uusiksi. DOM:ia voidaan käyttää myös muiden kielten kanssa, eli se ei ole JavaScript sidonnainen. [25]

JavaScript kehityksessä apuna käytetään erinäisiä ohjelmistokehyksiä. Suosituimpia niistä on Angular, React, Vue ja JQuery. Nämä neljä ohjelmistokehystä, ja muut saman kaltaiset, sisällyttävät DOM:in itseensä, joten sovelluksen kehittäjän ei tarvitse erikseen keskittyä DOM-sääntöihin ja siten ohjelmointi on helpompaa. Toisin sanoen ohjelmistokehykset piilottavat DOM:in sisäänsä. [19] Myös CSS:n puolella on käytössä ohjelmistokehyksiä, kuten Bootstrap sekä Reactin kanssa käytettävä MUI. CSS:n ohjelmistokehykset tarjoavat valmiita keinoja kehittää nettisivuston ulkonäköä. Samalla ne saattavat sisältää koodia,

jolloin esim. sivun skaalautuvuuslogiikkaa eri tarvitse erikseen keksiä uudestaan.

Selainpuolella on pääosin kolme erilaista kehitystaktiikkaa; Staattinen, dynaaminen tai yksisivuinen web-arkkitehtuuri (Single-page application, lyh. SPA). Jokaisessa nettisivuston toteutustapa on erilainen, mutta tavoite sama: selainpuoli lähetetään tavalla tai toisella käyttäjän selaimeen.

3.3.1 Staattinen nettisivusto

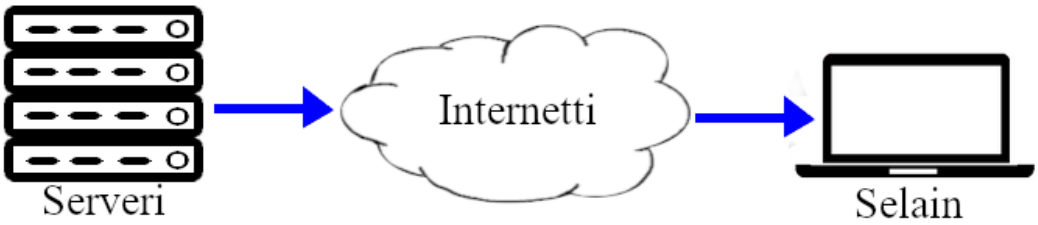
Kun nettisivustot kehitettiin, staattinen nettisivusto oli alkuperäinen tapa luoda niitä. Staattisen nettisivusto on erittäin kevyesti ja yksinkertaisella koodilla rakennettu sivusto. Nimensä mukaisesti staattisen sivusto on muuttumaton, eli jokainen käyttäjä näkee yhden sivuston sivun täsmälleen samanlaisena. Jokainen kokonainen sivu on oma lähdekooditiedosto. Eli suhteellisen samannäköiset sivut ovat eri tiedostoja, jotka sisältävät samaa koodia pienten erojen kera. Esim. mikäli sivustolla olisi kaksi sivua, jotka ovat täysin samanlaisia, mutta tekstin väri on eri, nämä kaksi sivua ovat kaksi toisiaan kopioivaa tiedostoa, joissa ainoastaan yksi koodipätkän, tässä tapauksessa yksi virke CSS:ää, on erilainen.

Staattista sivustoa kannattaa käyttää pienissä sivustoissa, joiden on tarkoitus pysyä muuttumattomina pitkän aikaa. Tai vaikka vain demoesityksen ajan. Ne ovat erittäin nopeita sekä helppoja kehittää ja siten kustanteisesti halpoja. Koska sivut lähetetään käyttäjän selaimeen esirenderöityinä, eli esipiirrettynä, ne eivät kärsi ajallisista tai muista yhteys ongelmista, esim. puuttuvasta tai hitaasti latautuvasta kuva. Ne eivät tarvitse palvelinpuolta toimiakseen, vaan voivat toimia myös täysin itsenäisesti. Huomioitavaa on, etteivät niitä myöskään ole suunniteltu kommunikoimaan tietokantojen kanssa.

Staattisen nettisivustot ovat ainoastaan katselmointia varten. Sivustolla kävijä pystyy huonosti itse kommunikoimaan sivuston kanssa. Staattisia nettisivustoja on myös vaivalloista päivittää. Edes yhtä uutta sivua tai artikkelia varten pitää

olla henkilö, joka osaa ohjelmoida kyseistä nettisivustoa. Myös valmiiden sivujen muuntaminen on aikaa vievää. Mikäli haluaa esim. Muuttaa navigoivaa yläpalkkia sivustolla, sama näkyvä koodin kohta pitää muokata kaikissa tiedostoissa samalla tavalla.

Staattiset nettisivustot ovat huono vaihtoehto, mikäli nettisivustosta haluaakin saada suuren sekä raskasrakenteisen. On erittäin aikaa vievää ja epäkäytännöllistä luoda isoa kasaa samankaltaisia tiedostoja, jolloin lopulta saadaankin vain tiedostojen viidakko aikaan. Suurin osa nykyajan sivustoista on isoja ja niiden halutaan olevan muuntuvia. Sen takia staattisen nettisivusto menetti suosiota pitkällä aikavälillä, mutta nyt sanotaan, että niiden suosio on viime aikoina jälleen noussut ja olisi nousussa tulevaisuudessakin. [26, 27, 29]



Kuva 4 Staattinen nettisivusto lähetetään kokonaisena selaimeen internetin välityksellä.

3.3.2 Dynaaminen nettisivusto

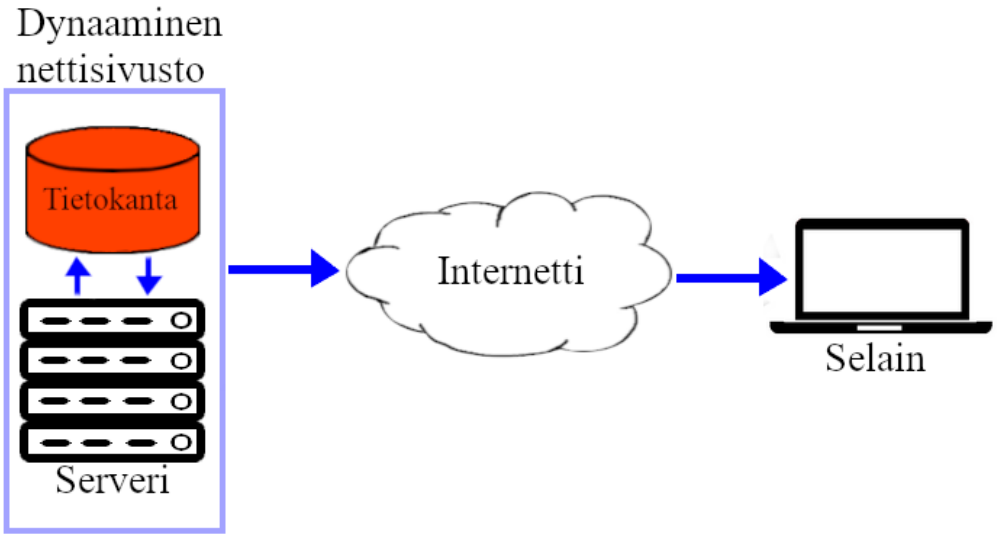
Dynaaminen nettisivusto, toisin kuin staattinen, ei ole koskaan niin sanotusti valmis pakkaus tai lopullinen versio itsestään. Se rakentuu ja muuttuu jatkuvasti sitä käyttäessä. Kokonaiset sivut eivät staattisen nettisivuston tavoin ole erillisiä omia tiedostoja, vaan ne rakennetaan erinäisistä pienemmistä tiedostoista tai komponenteista. Dynaamiset sivut luodaan reaaliaikaisesti ja monista sivustolle tallennetuista ulkoisista muuttujista riippuen. Kaikki voidaan laittaa vaikuttamaan dynaamiseen sivustoon: Kuka sitä katselee? Milloin? Missä? Ja vaikka mikä kellonaika ja millaista säätä on luvattu tai mitä nappeja käyttäjä on painellut sivustolla.

Dynaamisen nettisivusto saattaa tehdä erittäin paljon yhteistyötä tietokantojen kanssa, joten sitä kehittäessä pitää osata sekä nettisivusto-ohjelmointia että vahvaa tietokanta suunnittelua.

Koska dynaaminen nettisivusto toimii reaaliaikaisesti, sen kanssa käyttäjä voi kommunikoida. Toisin kuin staattiseen, dynaamista varten ei aina tarvitse osata ohjelmoida. Esim., mikäli sivusto perustuu uusien artikkelien luomiseen ja muokkaamiseen, sitä varten on voitu tehdä sivu, johon voi käyttäjä mennä kirjoittamaan uuden haluamansa artikkelin ja julkaista se, eli toisin sanoen luoda uusi sivu sivustolle. Käyttäjät voivat myös muokkailla dynaamisten sivustojen värejä, tekstejä ja kuvia. Tämä on mukautuvan datan ansiota, joka helpottaa pääsyä käsiksi jo valmiiseen dataan sivustolla. Tietenkin kaikki mukautuvuus rajoittuu siihen, mitä dynaamisen nettisivuston kehittäjä on suunnitellut muunneltavaksi.

Nettisivuston ohjelmoinnillinen muokkaaminen on myös paljon helpompaa kuin staattisessa lähestymistavassa. Koska kaikki osiot sivustolla ovat komponentteja, voidaan tehdä yksinään niihin muutoksia ja nämä muutokset vaikuttavat kaikkiin sivuston sivuihin.

Dynaamisen sivuston yhtenä huonona puolena staattiseen verrattuna on sen hitaus. Koska muuttuvat osiot palvelimen puolella rakennetaan valmiiseen HTML dokumenttimuotoon ja lähetetään seilainpuolelle, saattaa ilmetä kankeita viiveitä tiedon haussa. Lisäksi dynaamisen sivuston kehittäminen on hitaampaa ja kalliimpaa sekä itse ylläpitokin on hintavampaa kuin staattisen kanssa. Dynaaminen on kuitenkin kannattavampaa, sillä se mukautuu ihmisten tarpeisiin paljon paremmin kuin staattinen sivusto. [27, 28, 29]



Kuva 5 Dynaaminen nettisivusto käyttää aktiivisesti tietokantaa ja lähettää valmiin nettisivuston selaimeen.

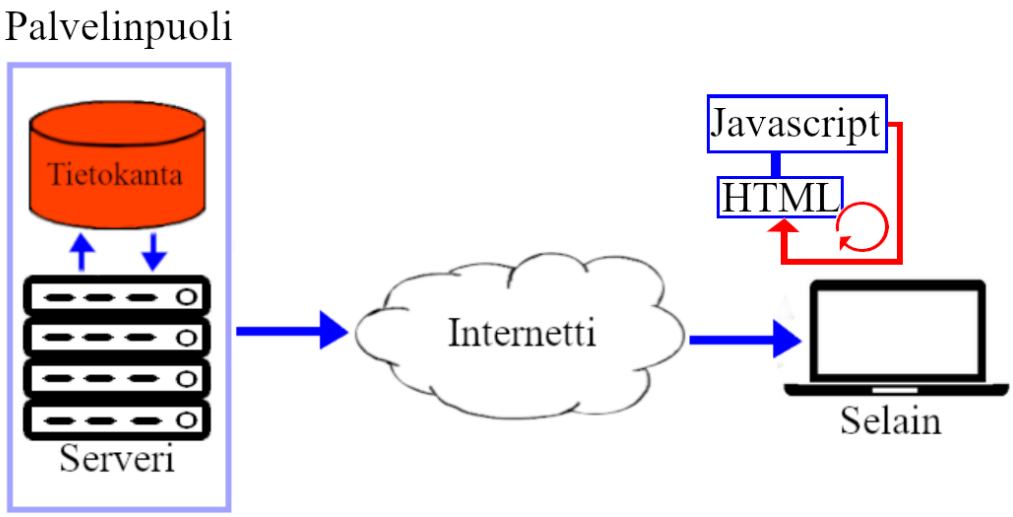
3.3.3 Yksisivuinen web-arkkitehtuuri

Yksisivuinen webarkkitehtuuri (engl. Single-Page Application, SPA) on dynaamisen nettisivuston kaltainen lähestymistapa, mutta nettisivuston HTML-dokumentti generoidaankin selaimen puolella. Tämä toimii siten, että selaimeen lähetetään koko selainpuolen koodi, joka on kehitetty JavaScriptillä ja se generoi HTML-dokumentin. Näin ollen selainpuolella dynaamiseen tapaan nettisivusto ei ole koskaan valmis vaan alati kehittyvä ja sen kanssa voidaan kommunikoida. Erona on, että nettisivuston luonti on siirretty selaimen vastuulle. Palvelinpuolelle otetaan yhteyttä, kun halutaan esim. tietokannallisia, laskennallisia tai muita logistillisia toimintoja toteutettaviksi.

SPA-taktiikka yleisimmin on paljon nopeampaa kuin dynaamisen nettisivusto. Sanotaan jopa välittömästä toiminnasta, sillä nettisivusto kyetään generoimaan ilman varmistettuja vastauksia palvelinpuolelta. Puuttuvien tietojen kohta voidaan esim. korvata latausta kuvastavalla symbolilla, kuten pyörivällä ympyrällä. Yhtenä isona etuna on, että nettisivusto voidaan generoida selaimeen, vaikka palvelinpuolella olisi ongelmia. Kyseisessä tilanteessa voidaan ilmoittaa selain-

puolen käyttäjälle ilmenevistä ongelmista palvelinpuolella. Kätevyydestä ja nopeudesta huolimatta SPA ei ole ongelmaton. Yhtenä huonona puolena on, että SPA saattaa olla erittäin raskas prosessi itse selaimelle, jolloin saattaa tulla latausviiveitä.

Koska kaikki tiedot selainpuolella on käyttäjän selaimessa ja muunneltavissa, voi siinä käytettävä JavaScript olla erittäin raskas ja monimutkainen kehittää. SPA:ssa on suosittua käyttää ohjelmointikehyksiä sekä kirjastoja, jotka helpottavat taakkaa ja nopeuttavat ohjelmointia. Lisäksi, koska SPA:ssa sivu on aina vakio, eli URL ei muutu, on suosittua käyttää selaimen omaa säilytystilaa, kuten Localstoragea. Selaimen localstoragessa tiedot ei katoa, vaikka sivun päivittäisi. [29]



Kuva 6 SPA toimii dynaamisen tavoin tietokannan kanssa ja lähettää puolestaan JavaScriptin selaimeen, jolloin selain luo nettisivuston.

4 IoT-verkkokehityksen eroavaisuus perinteiseen verkkokehitykseen

Nettisivusto on yksi IoT-laitteen keinoista näyttää ihmiskäyttäjälleen keräämäänsä ja prosessoimaansa dataa. Samalla sen pitää olla kykeneväinen vastaanottamaan tietoa käyttäjältä ja prosessoida sitä sekä mahdollisesti lähettämään tiedot eteenpäin laitteelle käsiteltäväksi.

IoT-nettisivustoa on hieman monimutkaisempaa kehittää, kuin ns. tavallista nettisivustoa. Perinteiset nettisivustot ovat yleensä suhteellisen kevyitä verrattuna IoT-nettisivustoihin. IoT-nettisivustoa kehittäessä pitää hallita monimuotoista nettisivusto ohjelmointia ja syvempää ymmärrystä tietokannan hallinnasta sekä ohjelmoinnista. Kovat vaatimukset luovat haasteita aloitteleville ohjelmoijille.

IoT-verkkokehityksen isoimmat haasteet ja huomioitavat asiat ovat

- haastava data
- toimiva käyttöliittymä
- turvallisuus
- hybridikehitystiimit.

Lisäksi haasteita voi tulla jopa laitetasolla, sillä IoT-nettisovelluksia voidaan myös kehittää sulautettuihin järjestelmiin, joissa voi olla tila- tai muistirajoitteita. Tämä luo lisähaastetta, kun kehitetään nettisivustoa, joka olisi samanaikaisesti kattavasti toimiva, turvallinen ja kevyt.

4.1 Haastava data

IoT-laitteet toimiessaan keräävät ja käsittelevät massiivisen määrän dataa reaaliaikaisesti. Samalla ne lähettävät tämän suuren määrän dataa nettisivuston palvelimelle. Yksi tärkeä hallittava ominaisuus IoT-nettisovelluskehityksessä on, että datan vastaanottaminen ja käsittelyn pitää olla nopeaa, suorastaan reaaliaikaista. Saatu data pitäisi olla täsmälleen samaa, mitä IoT-laite on lähettänyt. Samanaikaisesti palvelimen pitäisi kyetä prosessoimaan vastaanottamaansa dataa. Data pitää esim. saada toiseen muotoon, sillä aina IoT-laitteen lähettämä data ei ole ihmiselle luettavassa muodossa tai se on pakattuna raakadata versiona. Näissäkään tapauksissa data ei saisi muuttua vääristyneeksi esim. virheellisen muuntelun tuloksena. Isot datat saattavat aiheuttaa viiveitä tai toiminnallisia katkoksia palvelimen toimivuuteen, mitä pitäisi kyetä välttämään. Data ei myöskään saisi kadota missään vaiheessa, vaan se tulisi olla takaisin saatavissa ja myös uusiopäivitettävänä. [30]

Palvelinpuoli ei tiedä, minkälainen selainpuoli tai kuinka monta on vastaanottamassa tietoa, joten sekin pitää ottaa huomioon. Sama data pitäisi saada lähetetyksi nettisivustolle, puhelimelle tai jollekin muunkaltaiselle laitteelle. Jokaisella näillä on oma toimivuutensa ja kielensä.

4.2 Mukautuva selainpuoli

IoT-kehityksessä halutaan toimivia ja hienoja nettisivustoja. Selainpuolen kehittäminen ei ole helppo tehtävä, sillä siinä käytettävät tekniikat, kuten ohjelmistokehykset ja kirjastot tai ihan vain harvinaisemmassa tapauksessa ohjelmointikieli, kehittyvät alituisesti. [31] Alle puolessa vuodessa voi jokin tekniikka vanhentua ja vaatia päivitystä, joten nettisivustot tarvitsevat alituista ylläpitoa. Myös pitää olla ajan tasalla käytettävistä selaimista. Chrome on tämänhetkinen suosituin nettiselain, mutta myös löytyy muitakin selaimia, kuten safari, Firefox sekä DuckDuckGo. Käyttäjille on epämieluisa kokemus, mikäli nettisivusto ei olekaan suunniteltu toimimaan heidän käyttämässään selaimessa.

Välillä on hankala ennustaa, minkälainen selainpuoli toimii milläkin alustalla. Nykyään helposti näkyvänä esimerkkinä selainpuolelta on, että nettisivustojen pitää kyetä mukautumaan isosta ja leveästä television kokoisesta näytöstä pienen kännykän pystymuotoiseen näyttöön. Koska kyseessä on IoT-laitteen tuottamaa dataa, tämä data saatetaan haluta nähdä graafisessa muodossa, eli kuvaajina. Haasteeksi tulee, miten esittää sama kuvaaja isossa että pienessä näytössä ilman, että näytettävä informaatio katoaisi tai näyttäisi erilaiselta. Kuvaajien tulisi samalla mukautua erikokoisiin datoihin. Data voi olla erittäin suuriarvoista tai pienarvoista. Sivustojen tulee myös uusiorenderöityä, kun uutta dataa tulee palvelimelle. Tämän tulisi tapahtua viiveettä ja reaaliaikaisesti. Selainpuolen pitää olla itse aktiivinen uuden datan vastaanottamisesta palvelinpuolelta, sillä teknisesti palvelinpuoli itse ei pysty lähettämään vastausta ilman pyyntöä.

4.3 Turvallisuus

IoT-nettisivustot saattavat käsitellä arkaa dataa. Mikä tahansa, mikä on monitoroitu IoT-laitteella, päätyy internettiin, missä huonon suojauksen seurauksena se voi vaarantua – esim. päätyä ei-tahdottujen osapuolien nähtäville. Näin ollen selaimen turvallisuus tulee ottaa huomioon kehittäessä, ja kehittäjillä tulee olla hyviä salaustaitoja

Tavanomaisesti IoT-laite on rakennettu siten, että käyttäjä pääsee siihen käsiksi vain erinäisen käyttöliittymän kautta. Puhtaasti paperilla tämä luo turvallisuutta, koska käyttöliittymässä pystytään rajaamaan käyttäjälle vain valittuja mahdollisuuksia hallita ja vaikuttaa laitteeseen. Käytännössä rajauksen pitää olla erittäin hallittua. Etenkin, mikäli sivusto sisältää tekstillisiä osioita, tulisi tekstin parsiminen olla huollella tehty. Vapaiden tekstikenttien tulisi ottaa huomioon SQL-injektiot, sekä muut käskymuotoiset lauseet, jotka voivat esim. olla menomatkalla IoT-laitteen muihin koodillisiin osioihin.

SQL-injektiolla pyritään päästä käsiksi tietokantaan joko hakien tietoa tai muuttaen sitä. Siinä tekstikentän kohtaan tavallinen tekstivastaus yliajetaan ja tilalle laitetaan sen sijaan SQL-kyselymuotoista tekstiä. Mikäli nettisivuston palvelin ei ole laitettu ottamaan huomioon SQL-injektiota, se lukee syötetyn tiedon SQL kyselynä ja toteuttaa siinä olevan käskyn. Tekstikenttään voidaan myös laittaa itse IoT-laitetta haittaavia käskyjä, etenkin jos laite on suunniteltu kuuntelemaan käyttäjän käskyjä. Palvelinpuolella tai vähintään IoT-laitepuolella tulee ottaa huomioon haitalliset käskyt sekä / tai rajata käskyt vain tiettyihin sallittuihin lukuihin/sanoihin. Mikäli käyttöliittymä ja palvelinpuoli on hyvin rakennettu, käyttäjä ei pysty lähettämään omia mielivaltaisia komentoja laitteelle.

Kuten kaikki muutkin nettisivustot, IoT-nettisivustot ovat riskialttiita verkkohyökkäyksille. Verkkohyökkäykset voivat ilmetä yhdessä tai kahdessa tilanteessa: selaimen ja palvelimen välillä sekä myös palvelimen ja IoT laitteen välillä, mikäli ne kommunikoivat internetin välityksellä. Internetyhteyksiä varten pitää lähetet-

tävä data olla salattu, eli kryptattu. Salauksella estetään ulkopuolisen mahdollisuus lukea internetyhteydessä kulkevaa dataa muuntamalla data ei-luettavaan muotoon. Vain vastanottaja ja lähettäjä osaavat lukea kryptatun viestin. Tätä kutsutaan TLS (Transport layer Security) salausprotokollaksi.

Nettiselain kehityksessä yleinen TLS keino on käyttää HTTPS (Hypertext Transfer Protocol Secure) yhteyttä [32]. Kyseinen yhteys luodaan luotetun kolmannen osapuolen avulla, eli varmentajalla. Englanniksi tämä nimitys on Certificate Authority (CA). Varmentaja on yritys tai organisaatio, jonka toiminta perustuu siihen, että se antaa esim. tarkastamalleen nettisivustoille omat yksityiset salausavaimet. Varmentaja luo nettisivustolle sertifikaatin, eli varmenteen, joka sisältää julkisen avaimen, sekä erillisen yksityisen avaimen. [33] Varmenteen ja avainten avulla nettisivustolta ja nettisivustolle kulkevasta datasta tehdään kryptattu, eli suojattu ulkopuolisilta. Kyseisen varmennuksen voit tehdä myös itse. Silloin varmennusta kutsutaan itse allekirjoitetuksi sertifikaatiksi, englanniksi self signed certificate. Tässä tapauksessa kuitenkin, koska todentaja ei ole ollut luotettava ulkopuolinen taho, nettiselaimet eivät luota sivustoon vaan varoittavat käyttäjäänsä.

Toinen keino varmistaa nettisivuston turvallisuus, etenkin, jos sivusto sisältää kirjautuneita käyttäjiä, on käyttää todennusta. Yksi keino on käyttää JSON Web Token:ia (JWT), eli satunnaisesti luotua merkkiä, joka lähetetään kirjautuvalle käyttäjälle. Aina, kun kirjautunut käyttäjä tekee jotain nettisivustolla, tämä merkki tarkistetaan aidoksi samalla. JWT:hen tutustumme tarkemmin Kirjautuminen-osiossa.

4.4 Hybridikehitystiimit

Koska kyseessä on IoT-laitetta varten luotu nettisivusto, kehittäminen tapahtuu hybridikehitystiimissä. Hybridikehitystiimit eivät ole yhtä tuottavia tai edes helppoja toteuttaa kuin yksittäistiimit. Hybridikehitystiimin voi sisältyä nettisivustokehittäjiä, sulautettujen järjestelmien kehittäjiä sekä kyseiseen projektiaiheeseen perehtyneitä asiantuntijoita. Viimeisin tiimi voi olla myös ihan vain asiakas, jolla

on tietoa, miten kyseinen rakennettava tai jatkokehitettävä tuote toimii. Kommunikointi on avainasemassa. Jokaisella tiimillä on tietoa omasta osa-alueestaan, mutta voi olla, ettei yksi tiini tiedä toisen tiimin asioista yhtään mitään. Lisäksi yhteisistä parametreista tai ohjelmien kommunikoinneista ja niiden toimivuuksista tulee keskustella huolella, sillä kaikki osapuolet saattavat käyttää niitä. Hybridikehityksessä voi olla tilanteita, joissa yhden osapuolen tekemät muutokset vaikuttavat toisen osapuolien ohjelmiin.

5 Nettisivuston kehitys kohdelaitteelle

5.1 Tavoite

Tämän lopputyön tavoitteena oli kehittää nettisivusto, joka pyörisi kohdelaitteessa, joka on samalla sulautettu järjestelmä. Tavanomaisen nettisivuston kaltaisesti sivustolle pitäisi pystyä kirjautumaan ja käyttäjätyyppejä olisi erilaisia kuten esim. user ja admin. Nettisivuston pitäisi kyetä suhteellisen reaaliaikaisesti vastaanottamaan dataa, muuntamaan se ihmisen luettavaan muotoon, sekä näyttämään saamaansa graafisena muotona. Nettisivuston tulisi myös kyetä lähettämään parametreja sekä tiedostoja kohdelaitteelle. Kyseessä on demoversio, jota olisi tarkoitus jatkokehittää myöhemmin. Tarkkoja vaatimuksia nettisivuston kehitykselle ei annettu, vaan keinot ja lopputulos oli suhteellisen vapaita päättää itse. Kun demoversio on tehty ja dokumentoitu, asiakas jatkokehitettäisi sitä lopulliseen versioon.

5.2 Käytetyt keinot

Käytettävät keinot ja lopputulos olivat suhteellisen vapaavalintaisia ja oman harkinnan mukaan valittuja. Ainoana ehtona oli, että käytetään NodeJS:ää ja ReactJS:ää kehittämisessä. NodeJS käyttää ohjelmistokehyksenä Express:iä, jonka tarjoaa mahdollisuuden luoda palvelimelle REST API:n. Expressin tarjoaa mahdollisuuden pyyntöjen reititykselle, eli jokaiselle pyynnölle voi olla oma

URL-osoite. Samalla pyyntöjä voidaan käsitellä väliohjelmistossa (engl. Middleware), eli esim. tarkistaa kysyjän tiedot, kuten, mikä käyttäjätyyppi tai todenmukaisuus, sekä samalla mahdollisuuden hallinnoida virhetilanteita.

Vaikka aikaisemmin tässä projektissa on sanottu, että palvelinpuolella tapahtuvat asiat ovat kulisseissa, myös sielläkin halutaan salata joitakin asioita ulkopuolisilta. Tätä varten projektissa on otettu käyttöön dotenv-moduuli. Dotenv mahdollistaa ympäristömuuttujien tietojen tallentamisen .env-tiedostoon, joka yleensä piilotetaan tai ei oteta mukaan, mikäli koodia siirretään paikasta toiseen. .env-tiedostossa olevia muuttujia pystytään kutsua nodeJS:n oliolla process.

5.2.1 Sequelize

Palvelinpuoli käyttää kahta erilaista tietokantaa: kohdelaitteeseen kanssa jaettavaa sekä omaa tietokantaa. Oma tietokanta sisältää puhtaasti täysin nettisivuston liittyvää dataa. Jaettava tietokanta puolestaan sisältää kohdelaitteeseen keräämää dataa. Kohdelaitteeseen kanssa palvelin kommunikoi seuraavanlaisesti: kohdelaite kerää saamaansa dataa tietokantaan. Palvelin puolestaan hakee datansa yhteisestä tietokannasta käyttäen sequelize kirjastoa. Sequelize on ORM (Object Relational Mapping, suom. olio-relaatiokuvaus) työkalu. ORM on ohjelmoinnissa keino olla yhteydessä tietokantaan, ja korvata perinteiset SQL-kyselyt kehittäjille helpommiksi muodoiksi [34]. Sequelize on suunniteltu toimivaksi nodeJS:n apuna, jolloin saadaan yhteyksiä erinäisiin tietokantoihin, kuten MariaDB, MySQL tai Postgress. Sequelize on lupauspohjainen ORM, eli se lähettää lupauksen, kun siltä pyydetään jotain, esim. hakea kaikki tieto jostain osiosta tietokantaa. Kun lupauksen kohde on suoritettu, se lähetetään kutsujalle. Toimii hyvin tilanteissa, joissa joudutaan esim. odottamaan vastausta. Sillä on myös kyky estää sql injektioita. [35] Sequelizea käytetään myös nettisivuston oman tietokannan hauissa.

```
const sequelize = new Sequelize(process.env.DATABASE, pro-
cess.env.DB_USER, process.env.DB_PASSWORD, {
     host: process.env.HOST,
     dialect: process.env.DB_DIALECT,
     DialectOptions: {
          socketPath: process.env.SOCKED_PATH
     },
     Logging: false
})

const db = {}
db.sequelize = sequelize
db.example = require(./example.model.js)(sequelize,Sequelize)
```

Esimerkkikoodi 1. Uuden sequalize-konstruktion luonti, sen ja example-mallin lisääminen db-olioon.

```
module.exports= (sequelize, Sequelize) => {
     const Example = sequqlize.define('example', {
          name: {
               type: Sequelize.STRING,
               allowNull: false,
               unique: true
          }
     }
}
```

Esimerkkikoodi 2. Example-mallin koodi.

Esimerkkikoodissa 1 luodaan sequelizella yhteys tietokantaan. Sequalizesta luodaan uusi konstruktio, johon asetetaan yhteysparametreja. Db_user ja db_password ovat nettiselainta varten tietokantaan luodun käyttäjän tiedot. Host määrittää, kuka isännöitsee tietokantaa. Voi olla oma tietokanta (localhost) tai linkki jonnekin muualle tietokoneessa. Dialec kertoo, mikä tietokanta on kyseessä. DialectOptions ja siitä socketPath kertoo reitityksen tietokantaan. Tämä siksi, koska tietokannalla ei ollut porttia kohdelaitteessa. Logging puolestaan on, ettei sequelize nettiselaimen pyöriessä raportoi tapahtumia. Konstruktion luonnin jälkeen se asetetaan db –olioon, jota kutsutaan aina, kun halutaan kutsua ja käsitellä tietokantaa. Esimerkkikoodissa 2 luodaan malli (engl. model), joka siis mallintaa sequelizelle, minkälainen Example-taulukko on. Esimerkkitaulukossa on vain yksi kenttä: nimi. Sille kerrotaan, että sen tyyppi (type) on merkkijono (Sequelize.STRING), se ei koksaan voi olla tyhjä (allowNull: false) ja pitää olla uniikki (unique: true). Esimerkkikoodissa 2 example-malli lisätään db-olioon kohdassa db.example = require(./example.model.js)(sequelize,Sequelize).

5.2.2 Datan muuntaminen

Kun data on haettu tietokannasta, seuraavana prosessissa on datan kääntäminen. Lyhyesti sanottuna data on tallennettu isoon blobiin. Tarkastellaan paremmin, mitä tämä käytännössä tarkoittaa:

Blob on objekti, joka sisältää äärimmäisen paljon dataa sisällään. Se voi olla mitä tahansa. Se voi olla tekstiä, numeroita tai vaikka kuva, video tai äänitiedosto. Tämä data on binaari-merkkijono (binary string) datatyyppi muodossa. Binaari-merkkijono ns. “katkotaan” yhden tavun välein. Yksi tavu on 8 bittiä, jolloin binaari-merkkijono on kasa 8-bittisiä bittisarjoja. Nämä bittisarjat voidaan muuntaa heksadesimaaliksi, desimaaliksi tai ASCII (American Standard Code for Information Interchange) muodoksi. Tässä projektissa ASCII:ta ei käytetty, mutta käytämme sitä taulukko 1:ssa vertailun vuoksi ja, koska on myös hyvä tietää, että ASCII-merkistö voidaan luoda binaari-merkkijonosta.

Taulukko 1 Eri arvoja käännettynä. ASCII-merkistö on 7-bittinen, jonka takia se loppuu desimaali 128 kohdalla.

ASCII merkistö	**Heksadesimaali**	**Binaari**	**Desimaali**
1	31	00110001	49
2	32	00110010	50
3	33	00110011	51
>	3E	00111110	62
?	3F	00111111	63
@	40	01000000	64
H	48	01001000	72
I	49	01001001	73
J	4A	01001010	74
h	68	01101000	104
i	69	01101001	105

j	6A	01101010	106
DEL	7F	01111111	127
	80	10000000	128
	81	10000001	129

Tämän projektin tapauksessa tiedämme, että saatu blobi sisältää ison kasan numeroita. Tietokannasta haetut numerot eivät kuitenkaan ole luettavassa muodossa eikä binaari muodossa, vaan ne ovat heksadesimaalimuodossa ja ne pitää kääntää ihmisen ymmärrettävään muotoon. Lisämausteena haasteelle on, että saatu data saattaa olla integer, (kokonaisluku) tai float (liukuluku). Datan koko, eli kuinka monta erinäistä arvoa siinä on, on vaihteleva. Arvoja voi olla pari tai tuhansia. Blob ei itse kerro, mitä se sisältää, vaan tieto pitää kerätä muualta. Kun kaikki tarpeellinen tieto on kerätty, blob kyetään muuntamaan listaksi erinäisiä arvoja ja lähettämään selainpuolelle. Tekemässäni demoversiossa ei tullut vielä tarpeelliseksi määrittää, kuinka monta arvoa blobissa pitäisi olla, joten jätin sen huomiotta ja muunsin kaikki saamani arvot.

```
const sliceBuffer = (data, buffer, sliceLength) => {
     for (let i = 0; i <= buffer.length; i += sliceLength) {
          if (if<= buffer.length-sliceLength) {
               data.push(buffer.slice(i, i+sliceLength))
          }
     }
}

let data = []
sliceBuffer(data, bolb, 4)
data = data.map(buf => {
     return buf.readFloatLE(0)
})
```

Esimerkkikoodi 3. Datan muuntaminen float-liukuluvuksi.

Blob, kun se haetaan tietokannasta, saadaan Buffer muodossa yhtenä pötkönä esim. <Buffer 40 5B 5F 3F 63 3G …> Jokainen kahden kirjaimen ja numeron yhdistelmä (5B) on yksi tavu. Jotta voidaan alkaa muuntaa saatua blobia listaksi eri numero arvoja, pitää ensin tietää, minkä tyyppistä numeroa (integer, float) halutaan saada. Esimerkkikoodi 3 tapauksessa haluamme saada float arvon ulos. Float arvosta on yleisessä tiedossa, että se on 4 tavua [36]. Tein funktion

sliceBuffer blobin muuttamiseksi listaksi. Se tarvitsee listan (data), mihin se tallentaa arvonsa, käsiteltävän blobin (buffer) sekä leikkauspituuden (sliceLength), eli tavujen määrän. Esimerkkikoodissa 3. luodaan tyhjä data niminen lista ja blob tietokannasta saatu. Koska haluamme saada float arvoja asetamme sliceBuffer muuttujan arvoksi 4, eli saatu Blobi pätkitään jokaisen neljän tavun jälkeen. sliceBuffer käy blobin läpi ja lisää datalistaan uudet pätkityt bufferit. Sen jälkeen käymme data-listan läpi ja muunnamme float-arvoiksi pätkityt bufferit buf.readFloatLE([offset]), joka on NodeJS:n tarjoama metodi. Lopputuloksena on lista float numeroita, jonka voi lähettää eteenpäin selainpuolelle.

5.2.3 Kirjautuminen

Kirjautumisesta tahdottiin demovaiheessa saada mahdollisimman yksinkertainen: on käyttäjiä, jotka voivat kirjautua sisään. Käyttäjä tyyppejä oli kolmenlaisia: admin pääkäyttäjä, asiakkaan käyttäjä user; tavallinen käyttäjä, sekä advanced; käyttäjä, jolla on user:ia enemmän oikeuksia sivustolla. Koska kyseessä on demoversio, uusien käyttäjien luonti tehtiin adminilla, eli admin ainoastaan kykeni luomaan uusia käyttäjiä.

Käyttäjät ja heidän salasanansa tallennettiin nettisivustoa varten luotuun tietokantaan. Salasanat ei tallennettu suorana tekstinä, vaan ne kryptattiin, eli salattiin. Kryptaamista varten käytin bcryptjs-kirjastoa. Sen avulla voidaan salata tekstejä bcrypt.hashSync(salasana, suolausnumero) -funktiolla. Salausnumerolla tarkoitetaan jotain syötettyä numeroa, jonka mukaan salasana hajautetaan algoritmilla. Se määrittelee samalla, kuinka kauan salasanaan salaaminen kestää sekä samalla sen hakkeroiminen auki. Yleensä mitä isompi numero, sen parempi, mutta pitää ottaa huomioon käyttäjän oma kärsivällisyys. Käyttäjällä ei saata olla intoa odottaa montaa minuuttia (tai edes sekunteja), että bcryptjs-kirjasto kryptaa salansanan tai tarkistaa salauksena. Koska bcryptjs on tehnyt itse salauksen, se pystyy myös avaamaan sen. Tämä tapahtuu yksinkertaisella bcrypt.compare(syötettySalasana, tallennettuSalasana) -funktiolla.

Kirjautuessa sisään käyttäjä saa itselleen yksityisen JSON Web Token:in, eli aikaisemmin mainittu JWT. JWT on JSON-pohjainen avoimen RFC 7519 standardin menetelmä, joka antaa käyttöoikeustietueita, eli tässä projektissa se vahtii ja sallii selain- ja palvelinpuolen kommunikoida keskenään. RFC 7519 on JWT standardi tunnus. JWT luo käyttäjälle allekirjoitetun merkin, joka luodaan käyttäen salaista merkkijonoa, eli ei koodiin näkyviin kovakoodattua merkkijonoa, tai yksityisavainta.

```
const token = jwt.sign({ user: 'name', role: ‘admin’ }, pro-
cess.env.SECRED, { expiresIn: '6h'})

router.get(‘/’, [checkJwt, checkRole([‘admin’, ‘advanced’])],
files.getAllFiles)

const checkJwt = (req, res, next) => {
     try {
          let jwtPayload = jwt.verify(req.headers[‘auth’], pro-
          cess.env.SECRED)
          Res.locals.jwtPayload = jwtPayload
     } catch (error) {
          next(error)
     }
}
```

Esimerkkikoodi 4. Esimerkki, miten merkkiä käsitellään. Ensin uuden merkin allekirjoittaminen, sitten reititys/tiedonsiirto vaiheessa laitetaan checkJwt-funktio tarkistamaan merkki.

Esimerkkikoodissa 4 on ensin yksinkertaistettu esimerkki JWT merkin allekirjoittamisesta. Merkin (token) allekirjoitukseen laitetaan ensiksi tietosisältönä olio, joka sisältää tietoa. Esimerkkikoodissa 4 olio on kahden kentän mittainen: käyttäjän nimi (user: ‘name’), ja rooli (role: ‘admin). Sen jälkeen funktiolle annetaan salattu merkkijono. Lisäksi allekirjoitukseen laitetaan tieto, että merkki menee vanhaksi kuuden tunnin kuluttua. Allekirjoitettu merkki on pitkä merkkijono täynnä satunnaisia numeroita ja kirjaimia.

Kun käyttäjä tekee jotain nettisivustolla, mikä tarvitsee palvelinpuolelta hakuja, JWT merkki tarkistetaan siinä vaiheessa. Koodiesimerkissä u ensimmäisellä rivillä näytetään, miten ja missä vaiheessa JWT merkin tarkistusta pyydetään, eli kun selainpuolen pyyntö (request) välitetään eteenpäin expressin tarjoaman rei-

tittäjän (router) avulla sitä käsittelevälle moduulille, se pyydetään ensinnä tarkistettavaksi checkJWT-funktiossa. Sen jälkeen on yksinkertaistettu esimerkki, miten merkki tarkistetaan. Jwt.verify()-funktio onnistuessaan tallentaa jwtPayload-olioon sille allekirjoitusvaiheessa annetun olion. Epäonnistuessaan funktio tulostaa virhetilanteen (error).

Koodiesimerkkissä 4 nähdään myös, että JWT merkin tarkistamisen lisäksi tarkistetaan käyttäjän rooli. Esimerkkitapauksessa admin ja andvanced ovat sallittuja käyttämään files.getAllfiles -moduulia, mutta user ei ole.

```
const checkRole = (roles) => {
     return async(req,res,next) => {
          try {
               if (roles.includes(res.locals.jwtPayload.role)) next()
               else {
                 res.status(401).send({error: 'Unauthorized action'})
                 next()
               }
          } catch (error) {
               next(error)
          }
     }
}
```

Esimerkkikoodi 5. Käyttäjän roolin tarkistaminen.

Esimerkkikoodissa 5 käyttäjän rooli tarkistetaan if (roles.includes(res.locals.jwtPayload.role)) osiossa. Mikäli käyttäjän rooli on sille annetussa listassa, mikä tällä hetkellä on admin ja advanced, se pääsee tarkistuksen läpi ja mennään eteenpäin, eli poistutaan funktiosta (next()). Mikäli vertailu ei ole tosi, lähetetään virheviesti.

5.2.4 Kommunikointi kohdelaitteen kanssa

Nettisivuston kommunikointi kohdelaitteelle tehtiin puolestaan eri tavalla. Nettisivusto ei tee muutoksia yhteiseen tietokantaan, sillä se on vain yksisuuntainen säilytyspaikka datalle. Sen sijaan kohdelaitteelle piti kyetä lähettämään käskyparametreja sekä tiedostoja. Tiedostojen kohdalla piti ensin saada satunnainen määrä tiedostoja selainpuolelta palvelinpuolelle. Tätä varten on kehitetty erinäisiä väliohjelmistoja, joista valitsin käytettäväksi multer:in. Multer on NodeJs:ään

kehitetty väliohjelmisto, joka kykenee käsittelemään multipart/form-data –dataa [37]. Tässä tapauksessa multer käsittelee tiedostoja, jotka lähetetään lomakkeen mukana.

```
const upload = multer({ storage: multer.diskStorage({
     destination: pathToFiles,
     filename: ((req, file, callback) => {
          callback(null, file.originalname)
     })
})}).array(`files`)

exports.uploadfiles = (req,res,next)=> {
     upload (req, res, async (err) =>{
          if (!req.files) return next({data: `Content can not be
          empty`, name:`serverError`})
          if (err instanceof multer.MulterError) {
               return res.status(500).json(err)
          } else if (err) {
               return res.status(500).json(err)
          }
     })
}
```

Esimerkkikoodi 6. Multerin konfiqurointi ja sen jälkeen sen käyttäminen.

Koodiesimerkki 6 näyttää meille, miten multer yksinkertaisuudessaan toimii. Aluksi multer konfiquroidaan. Sille asetetaan, mihin se tallentaa käsiteltävät tiedostot. DiskStorage moottori kykenee tallentamaan tiedostot mihin tahansa tietokoneen tiedostoihin. Destination-kohtaan laitetaan, mihin haluamme saada tiedostot. Turvallisuus syistä multer tallentaa tiedostot satunnaisella nimellä. Mikäli tämän ominaisuuden haluaa poistaa, pitää erikseen lisätä filename-kohtaan, millä nimellä kyseinen tiedosto halutaan tallentaa. Myöhemmin konfiguroitua multeria (upload) tarvitsee vain kutsua ja se toimii virheettömissä tilanteissa. Multer tarjoaa virhetilanteita varten multer.MulterError-luokan, joka käsittelee multerin osa-alueisiin kuuluvat virheet. Lisäsin varmuuden vuoksi tilanteet, joissa tiedostoja ei sittenkään ole, jolloin lähetetään virheviesti, sekä yleisen virhetilanteen hallinnan.

Nettisivuston kommunikointi kohdelaitteelle päin lähetetään sisäisten yhteyksien, eli porttien kautta. Porttien avulla käsitellään yksittäisiä fyysisiä yhteydenmuodostuksia, eli juuri tässä tapauksessa palvelinpuolen pyyntöä käsitellä tiettyä lähetettyä parametria tai käydä läpi ladattuja tiedostoja. Käytettävien porttien

numerot ovat 0:sta 65535 numeroon saakka. Joillekin porteille on ennalta määritelty, mitä varten ne ovat, kuten 80 portti on HTTP-yhteyksille ja 443-portti puolestaan HTTPS-yhteyksille. Jotta nettisivuston palvelin saa yhteyttä kohdelaitteeseen, tai ylipäätänsä, miten palvelinpuoli saa yhteyttä muihin portteihin, käytetään NodeJS:ään kehitettyä Net moduulia, joka tarjoaa asynkronisen verkkosovellusliittymän muihin portteihin [38].

```
const data = {name: example}

const client = net.connect({port: 3002}, () => {
     client.write(JSON.stringify(data))
})

client.on(`data`, (resData) => {
     const response = JSON.parse(new Buffer.from(resData).toString())
     client.end()
     if (response.response !== `error`) {
          return res.status(200).send(response.message)
     } else {
          return next({data: response.message, name:`serverError`})
     }
})

client.on(`end`, () => {
     client.destroy()
})

client.on(`error`, (errorMes) => {
     return next({data: errorMes.code, name: errorMes.code})
})
```

Esimerkkikoodi 7. Net-moduuli esimerkki. Client.on() takaisinkutsu funktiot erotellaan toisistaan ja ne toimivat itsenäisesti.

Koodiesimerkki 7:ssa net moduuli otetaan käyttöön luomalla yhteys valittuun porttiin. Otamme yhteyttä 3002-porttiin ja lähetetään objekti. Client.on() ovat takaisinkutsu funktioita, jotka odottavat ja vastaanottavat tapahtuman (data, end, error). Data-tapahtumassa välitämme saadun vastauksen selainpuolelle. End puolestaan sulkee yhteyden ja error käsittelee virhetilanteen.

5.2.5 Selainpuolen keinot

UI:n puolella lähdin kehittämään mahdollisimman yksinkertaista käyttöliittymää. UI tulee englannin kielen sanasta User Interface, eli käyttöliittymä. Käyttöliittymä

toteutettiin käyttämällä yksisivuista webarkkitehtuuria ja pyrin pitämään sen mahdollisimman yksinkertaisena. Mikäli käyttäjä ei ole kirjautuneena sisään sivustolla näkyy ainoastaan kirjautumislomake. Käyttäjän kirjauduttua sisään nettisivusto koostuu kolmesta osiosta. Sivuston yläreunassa on navigaatiopalkki, jossa näkyy sivut, joita kyseinen käyttäjätyyppi kykenee näkemään. Sivuston sisältö on navigaatiopalkin alapuolella. Sisältö vaihtuu sen mukaan, millä sivulla käyttäjä on. Vasemmalla on toiminnallisuuksiin perustunut sivupalkki ja sen oikealla puolella itse näkyvä sisältö. Yksinkertaisuuden vuoksi, jokainen sivu toteuttaa samaa formaattia. Esimerkki kuvassa 7.

UI:n kehittämisessä käytettiin Material UI, joka on avoin lähdekoodikirjasto suunniteltu ReactJS:ää varten. Se sisältää valmiita elementtejä, joita voi laittaa sivustoille kuten virheilmoituksia, sivupalkkeja ja keinoja navigoida sivustoilla. Material UI:n käyttö tässä projektissa oli hyvä valinta, sillä se tarjoaa tähän projektiin haluttuja yksinkertaisia keinoja käsitellä sivustoa.

Kuva 7 Sininen: navigaatiopalkki sivustolla eri sivujen selaamiseen. Vaalensininen: sivupalkki eri sivuilla vaihtuville toiminnoille, joilla voi vaikuttaa sisältöön. Vihreä: sivuston sisältö. Esimerkissä esimerkkikuva React-vis:n luomasta kuvaajasta

Toinen käyttäjälle näkyvä ominaisuus on React-vis kirjasto. Se tarjoaa yksinkertaisen ja keinon esittää kuvaajia. Se tarjoaa useanlaisia erilaisia kuvaajatyyppejä erinäisiin tarkoituksiin. Tässä projektissa halusimme saada yksinkertaisen lineaarisen kuvaajan, minkä React-vis tarjoaa. Yksinkertainen React-vis:llä tehty kuvaaja löytyy kuvasta 7.

Koska sovellus on kehitetty itse rakennettuun sulautettuun järjestelmään, siinä oleva käytettävä tila oli rajallinen. Sekä palvelinpuoli, että etenkin selainpuoli ei mahtunut laitteeseen, joten jouduin tekemään pienen tilakikkailun. Selainpuolen valmis versio pakattiin build-kansioon ja lähetettiin kopiona palvelin puolelle. Build-kansio on tiivistetty pakkaus selainpuolen koodista, jolloin se vie huomattavasti vähemmän tilaa. Kun palvelin laitetaan tuotannossa toimimaan, sillä on tiedossa, että sillä itsellään on tarjottavana selainpuoli, joten se lähettää sen käyttäjän selaimeen. Expressissä on tehty mahdollisuus luoda reitti erinäisiin kansioihin kutsumalla funktio express.static(). Jotta noudettu kansio saadaan vielä toimimaan selaimessa, se pitää lähettää res.sendFile() funktiolla. Yksinkertainen esimerkki koodiesimerkissä 8.

```
app.use(express.static(path.join(__dirname, ´/build´)))
app.get(´*´, (req, res) => {
     res.sendFile(path.join(__dirname, ´build´, ´index.html´))
}
```

Esimerkkikoodi 8. Selainpuolelta kopioidun build-kansion ottaminen käyttöön palvelinpuolella.

5.2.6 Redux

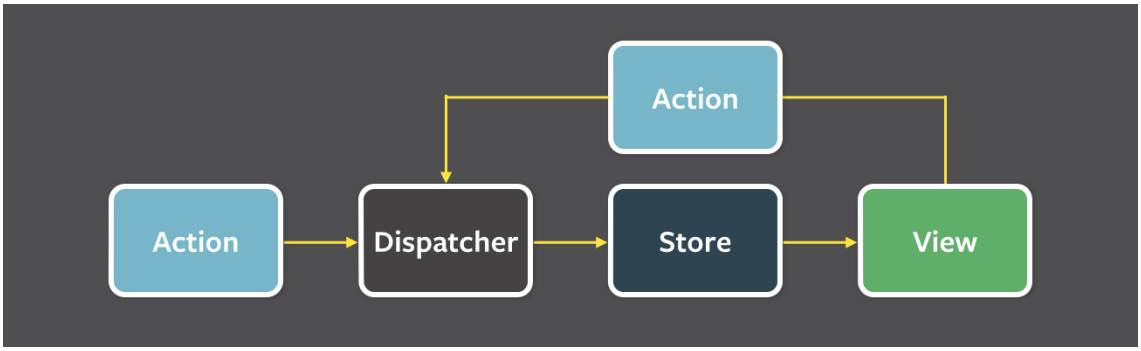
Selaimen puolella erinäisten datojen säilyttämiseen käytin React:iin suunniteltua avoimen lähdekoodin kirjastoa nimeltä Redux. Redux:n avulla voidaan tallentaa kaikki selainpuoleen liittyvä tieto. Se toimii myös muuallakin kuin selainpuolella. Redux:n toiminta perustuu Facebookin kehittämään Flux-arkkitehtuuriin. Flux vaikuttaa koko selainpuolen arkkitehtuuriosioon, joka käsittelee sivuston dataa ja tietoa, joten siihen on kannattanut hyvin perehtyä ennen käyttöönottoa.

Fluxin toiminta perustuu kolmesta eri osasta: lähettäjästä (engl. dispatcher), varastosta (engl. store) ja näkymästä (view) [39]. Lähettäjä käsittelee kaikkea dataa, mitä Flux-sovelluksessa liikkuu. Sen kautta lähetetään, haetaan ja päivitetään dataa. Mitä isompi sovellus on, sitä käytännöllisemmäksi käsittelijöiden käyttö on, sillä niitä voidaan käyttää monessa eri osiossa, eikä sovelluksessa oleva datan käsittely mene ns. solmuun. Eli Flux-sovellus osaa järjestellä lähettäjille tulevat takaisinkutsut (callback), jolloin jokainen odottaa vuoroansa ennen toimintaansa.

Varastot puolestaan nimensä mukaisesti säilyttävät niille annetut datat. Puhutan myös sovelluksen tilanhallinnasta. Yksittäinen varasto rekisteröi itsensä tietylle valitulle lähettäjälle ja lähettää sille takaisinkutsun, eli ns. yhdistää itsensä lähettäjään. Takaisinkutsun avulla varasto vastaanottaa tapahtuvat muutokset, jotka sen pitää tehdä. Kun muutokset on tehty varasto ilmoittaa eteenpäin näkymille, että sen tila on muuttunut, jotta näkymät voivat reagoida siihen.

Näkymät hakevat datan varastoista ja piirtävät sen, eli näyttävät sen käyttäjälle. Reactissa erinäiset näkymät voivat yhdessä luoda yhden sivustonäkymän. Ne ovat vapaasti uusiopiirrettäviä, eli voivat reagoida käyttäjään tai varastoissa oleviin datoihin. Flux-sovelluksessa olevaa näkymää, joka tarkkailee varastoja, kutsutaan myös ohjainnäkymäksi. (engl. controller-view) Se, kuinka ohjainnäkymiä käsitellään ohjelmoinnissa, on itse päätettävissä. Ohjainnäkymä voi olla ns. hierarkiassa päänäkymä, joka kerää kaiken datan ja antaa sen eteenpäin jälkeläisnäkymille. Ohjainnäkymiä voidaan laittaa myös syvemmälle hierarkiassa, jolloin yksittäinen ohjainnäkymä kerää vain itselleen tarpeellisen datan. Tässä projektissa käytin jälkeistä taktiikkaa. Näin jokainen sivu ja ohjainnäkymä pääsi keskittymään vain itselleen tärkeään toimintaan ja dataan välittämättä muista.

Jotta lähettäjän saa toimimaan, se tarjoaa toimintoja (engl. Action), jota käyttäjä voi käyttää. Tekstikenttiin tai nappuloihin voidaan lisätä lähettäjä tarjoamia funktioita, jotka lähettävät käyttäjän toivomat muutokset lähettäjälle käsittelyyn.



Kuva 8 Facebookin kuvaelma Flux-arkkitehtuurin toiminnasta [40]

5.3 Testaus

Tavanomaisesti sovelluksen komponentit testataan yksikkötestien avulla. Yksikkötesti nimensä mukaan testaa yhtä yksikköä, eli tässä tapauksessa yhtä komponenttia tai funktiota kerrallaan. Yksikkötestaamista suositellaan tehtäväksi jokaiseen sovellukseen, sillä se löytää ja minimoi yksittäisten komponenttien tai funktioiden virhetilanteita. [40] Tavallisesta poikkeavasti, koska palvelinpuoli oli tässä demokehityksen aikana suhteellisen pieni, päätin tehdä integraatiotestit palvelinpuolelle. Integraatiotestauksen päätavoite on testata toimivatko eri komponentit yhteen ja antavatko ne toivotun lopputuloksen. Integraatiotestausta käytetään, kun halutaan esim. varmistaa sovelluksen ja tietokannan yhteistyön toimivuuden, mikä etenkin tässä projektissa oli erityisen tärkeää. Lisäksi integraatiotestaus testaa, että onhan erinäiset poikkeustilanteet otettu huomioon ohjelmoinnissa [41].

Integraatiotestaamisessa on monta eri keinoa, joista otin käyttöön Big Bang-keinon. Big Bang testityyppi saattaa sekaantua systeemitestaamisen kanssa. Isona eroavaisuutena on, että systeemitestaamisessa testataan koko järjestelmää / sovellusta, kun puolestaan integraatiotestaamisen Big Bang testaa tiettyjä yhdistettyjä / integroituja moduuleja ja funktioita. Tätä testityyppiä varten pitää kaikki yhtenäiseen integraatioon kuuluvat komponentit olla valmiina, sillä kaikki integroidut komponentit testataan samanaikaisesti yhtenä yksikkönä. Big Bang

testaustyyppi sopii parhaiten juuri pieniin sovelluksiin. Virhetilanteiden syntysijoja voi olla vaikea löytää, joten isoissa systeemeissä niiden etsintä olisi paljon aikaa vievää.

Palvelinpuolen testit tein käyttäen Jest testausohjelmistokehystä. Jest on kehitetty testaamaan JavaScriptiä, etenkin yksikkötestejä, mutta se myös taipuu integraatiotestaamiseen. Lisäapuna Jest testaamiseen käytin Supertest kirjastoa. Se on suunniteltu auttamaan API, eli ohjelmistorajapinnoille kehitetyissä testeissä. Supertestin avulla kykenin luomaan testit, jotka jäljittelivät mahdollisimman realistista tilannetta. Testi tilanteessa testattavia komponenttia kutsuttiin reitittimen kautta, joka vastaa samalla tavalla tilannetta, kuin selainpuoli kyselisi kyseistä komponenttia. Yksinkertaistettu testausmenetelmä esimerkkikoodi 9. Testejä varten luotiin täysin identtiset tietokannat alkuperäisistä tietokannoista.

```
describe("get all files test", () => {
        test(`with success`, async () => {
            const expecting = [
                {fileName: 'donkydoe.txt'},
                {fileName: 'mickeymouse.txt'},
            ]
            let answer = await api.get('/api/files').set('Authoriza-
tion', token)
            answer = JSON.parse(answer.text)
            answer.sort((a,b) => (a.fileName > b.fileName) ? 1 : -1)
            expect(answer).toMatchObject(expecting)
        })
    })
```

Esimerkkikoodi 9. Esimerkkitesti, jossa pyritään tarkistamaan testattavan tapahtuman onnistuminen. Testissä luodaan ensin odotettu lopputulos (expecting). Moduuli, joka hakee tiedostoja, kutsutaan sen reitityksen avulla. Vastauksen saatua, vastaus parsitaan, laitetaan järjestykseen ja tarkistetaan, onko se sama odotetun lopputuloksen kanssa.

Teoriassa selainpuolelle olisi voitu tehdä testejä, kuten yksikkötestejä eri toiminnallisten, kuten nappien ja tekstikenttien, testaamiseksi. Käytännössä, mutta koska oli tiedossa, että demoversio tullaan todennäköisesti tekemään uusiksi selainpuolella, testaukset olisi olleet turhia lopulta ja aikaa projektilta vievää. Näin ollen selainpuolella ei ollut erillisiä testejä.

5.4 Jatkokehitys

Nettisivusto on kehitetty ja dokumentoitu siten, että siitä pystytään lähtemään jatkokehittämään virallista ja mahdollisesti monimutkaisempaa versiota. Tässä demovaiheessa tuli jo mieleen mahdollisia muutoksia, joita sovellukseen voisi tehdä.

Kirjautuminen on hyvin pintaraapaisulla tehtyä. Kun lähdetään kehittämään lopullista versiota, olisi siihen hyvä sisällyttää sähköpostien kanssa kanssakäyminen. Sähköposti loisi käyttäjien hallintaan uusia ominaisuuksia. Tärkeimpänä ominaisuutena olisi, että mikäli käyttäjä unohtaa salasanansa, hän voisi sähköpostin avulla pyytää sen päivittämistä. Kun käyttäjän kanssa haluttaisiin kommunikoida, voitaisiin käyttää tallennettua sähköpostia. Tällä hetkellä kaikki valta käyttäjissä on adminilla, myös salasanan uusinta, mikä ei tietoturvallisesti ole paras vaihtoehto.

React-vis alkujaan tuntui hyvältä vaihtoehdolta yksinkertaisen graafisen piirroksen tekemiseen, mutta myöhemmin kehitysvaiheessa selvisi, että React-vis saattoi olla juuri vähän aikaa sitten vanhentunut, eikä sitä ylläpidetä kunnolla tällä hetkellä. Näin ollen React-vis voitaisiin korvata toisella graafisia ominaisuuksia tarjoavalla kirjastolla.

Testauspuolella voisi myös jatkokehittää. Olisi hieno ominaisuus, jos palvelinpuolelle lähetetyn build-kansion avulla pystyttäisiin End-To-End testata. End-To-End testaaminen testaa koko sovelluksen kuin todellinen käyttäjä olisi naputtelemassa sivustolla. Se kuitenkin voi olla erittäin hidas ajaa, sillä se simuloi reaaliaikaista testaamista, joten se olisi hyvä vaihtoehto lisätä lopputestiksi ennen kuin sovellus laitetaan tuotantoon.

Lisäksi, mikäli datan määrä lisääntyisikin virallisessa versiossa voisi Sequelize ORM:in poistaa ja korvata SQL kyselyillä. ORM:eilla sanotaan olevan huono tapa hieman hidastaa tietokantahakuja, etenkin mikäli dataa on erittäin paljon, mikä IoT-kehittämisessä ei olisi suotavaa. Näin ollen nopeuden puolesta itse

tehdyt SQL-kyselyt olisi parempi vaihtoehto. Tietenkin itse tekeminen on aikaa vievämpää kuin valmiin ORM:in käyttäminen.

6 Yhteenveto

Projektin tavoitteena oli luoda demoversio nettisivustosta asiakkaalle rakennetulle kohdelaitteelle. Päätavoitteena oli saada kohdelaitteen keräämä data nettisivustolle, prosessoida se ja näyttää se nettisivustolla. Kohdelaitteelle piti kyetä lähettämään tiedostoja sekä parametreja sekä sivustolle piti kyetä kirjautumaan. Sivustosta saatiin tehtyä demotuotantovaihetta varten pakattu ja tiivistetty, eli vähän tilaa vievä versio. Samalla se täytti demon vaatimukset, eli data saatiin prosessoitua ja lopputulos oli oikeanlaista. Viiveitä ei prosessoinnin aikana tullut. Myös kommunikointi kohdelaiteelle saatiin toimimaan. Demo-versio osoittautui onnistuneeksi ja siitä voidaan hyvin lähteä jatkokehittämään lopullista virallista versiota.

Lähteet

1 Alexander S. Gillis, 2022, What is the interdet of things (IoT)?,Verkkodokumentti, <https://www.techtarget.com/iotagenda/definition/Internet-of-Things-IoT> Luettu 20.7.-22

2 Ignacio de Mendizábal, 2022, IoT Communication Protocols – Network Protocols, Verkkodokumentti, <https://www.allaboutcircuits.com/technical-articles/internet-of-communication-communication-protocols-network-protocols/> Luettu 25.8

3 Akash Kandhare, 2019, Bluetooth Vs. Bluetooth Low Energy: What's The Difference?, Verkkodokumentti, <https://medium.com/@akash.kandhare/bluetooth-vs-bluetooth-low-energy-whats-the-difference-74687afcedb1/> Luettu 25.8

4 Melanie Pinola, 2019, Bluetooth-perusteet, Verkkodokumentti, <https://fi.eyewated.com/bluetooth-perusteet/> Luettu 9.8

5 Jon Martindale, 2021, What is Wi-Fi?, Verkkodokumentti, <https://www.digitaltrends.com/computing/what-is-wi-fi/> Luettu 26.8.2022

6 Damir Wallener, 2022, what is GHx?, Verkkodokumentti, <https://www.easytechjunkie.com/what-is-ghz.html> Luettu 26.8.2022

7 CenturyLink, The difference between 2.4GHz and 5 GHz WiFi, Verkkodokumentti, <https://www.centurylink.com/home/help/internet/wireless/which-frequency-should-you-use.html#:~:text=A%202.4%20GHz%20connection%20travels,use%20your%20WiFi%20connection%20most> Luettu 26.8.2022

8 Eclipse, 2019, What Is Cellular IoT?, Verkkodokumentti, <https://eclipstek.com/what-is-cellular-iot/> Luettu 30.8.2022

9 Tata Communications, A Guide to Cellular IoT, Verkkodokumentti, <https://www.tatacommunications.com/solutions/mobility-iot/cellular-iot-enablement/> Luettu 30.8.2022

10 Rob Lauer, 2021, What is Cellular IoT?, Verkkodokumentti, <https://blues.io/blog/what-is-cellular-iot/> Luettu 30.8.2022

11 Net-informations.com, 1G Vs. 2G Vs. 3G Vs. 4G Vs. 5GVerkkodokumentti, <http://net-informations.com/q/diff/generations.html> Luettu 30.8.2022

12 Pelion team, 5G for Iot – An Introduction, Verkkodokumentti, <https://pelion.com/blog/education/5g-for-iot/> Luettu 30.8.2022

13 Roman Lapa, 2022, IoT Communcation Protocols: Complete Guide for Startups, Verkkodokumentti, <https://dataflog.com/read/iot-communication-protocols-complete-guide-startups/> Luettu 30.8.33

14 GeekforGeeks, 2021, What is a Website?, Verkkodokumentti, <https://www.geeksforgeeks.org/what-is-a-website/> Luettu 25.9

15 Google, URL, Verkkodokumentti, <https://support.google.com/google-ads/answer/14095?hl=fi/> Luettu 8.9.2022

16 Microsoft, 2021, Perustiedot tietokannasta, Verkkodokumentti, <https://support.microsoft.com/fi-fi/office/perustiedot-tietokannasta-a849ac16-07c7-4a31-9948-3c8c94a7c204> Luettu 31.10

17 StackOverFlow, 2021, Developer Survey, Verkkodokumentti, <https://insights.stackoverflow.com/survey/2021#most-popular-technologies-language> Luettu 25.9

18 Node.js, About Node.js®, Verkkodokumentti, <https://nodejs.org/en/about/> Luettu 19.9

19 Mozilla, 2022, Introduction to client-side frameworks, Verkkodokumentti, <https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Introduction/> Luettu 20.10

20 Codeavademy Team, 2022, Verkkodokumentti, <https://www.codecademy.com/resources/blog/what-is-a-framework/> Luettu 21.9

21 Red Hat, 2022, What is an API?, Verkkodokumentti, <https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces/> Luettu 23.9

22 Red Hat, 2019, REST vs. SOAP, Verkkodokumentti, <https://www.redhat.com/en/topics/integration/whats-the-difference-between-soap-rest> Luettu 23.9

23 Red Hat, 2020, What is REST API?, Verkkodokumentti, <https://www.redhat.com/en/topics/api/what-is-a-rest-api> Luettu 15.9

24 How To GraphQL, Basics Tutorial – Introduction, Verkkodokumentti, <https://www.howtographql.com/basics/0-introduction/> Luettu 25.9

25 Mozilla, 2022, Introduction to the DOM, Verkkodokumentti, <https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction> Luettu 11.10

26 Joe Holmes, 2022, What is a static website?, Verkkodokumentti, <https://www.sanity.io/what-is-a-static-site> Luettu 25.9

27 Ben Kalkman, 2021, Static vs. Dynamic websites: What are They and Which Is Better? Verkkodokumentti, <https://rocketmedia.com/blog/static-vs-dynamic-websites> Luettu 25.9

28 Hughes&Co, The Difference Between Dynamic and Static Websites, Verkkodokumentti, <https://www.hughesandco.ca/blog/the-difference-between-dynamic-and-static-websites> Luettu 25.9

29 Charuka Herath, 2021, Web Development in 2021: Dynamic vs. Static. vs. Single-Page Architecture, Verkkodokumentti, <https://betterprogramming.pub/web-development-in-2021-dynamic-vs-static-vs-single-page-architecture-399d0c3defe6> Luettu 25.9

30 Ramotion, 2022, How Will The Internet Of Things Impact Web App Development?, Verkkodokumentti, <https://www.ramotion.com/blog/web-application-development-for-iot/> Luettu 9.10

31 Anjaneyulu Naini, 2022, Role of internet of Things(IoT) in Web Development, Verkkodokumentti, <https://geekflare.com/iot-on-web-development/#:~:text=IoT%20will%20create%20advanced%20communication,build%20up%20the%20correct%20directions> Luettu 18.8

32 Electronic Frontier Foundation, 2018, What Should I Know About Encryption?, Verkkodokumentti, <https://ssd.eff.org/en/module/what-should-i-know-about-encryption> Luettu 9.10

33 SSL.com Support Team, 2021, What Is a Certificate Authority (CA)?, Verkkodokumentti, <https://www.ssl.com/faqs/what-is-a-certificate-authority/> Luettu 11.10

34 Janne Ruuska, 2019, Olioista tietokantaan – ORM, Verkkodokumentti, <http://janneruuska.weebly.com/bloki/olioista-tietokantaan-orm> Luettu 12.10

35 Sequelize, 2022, Sequelize v6, Verkkodokumentti, <https://sequelize.org/docs/v6/> Luettu 12.10

36 Oracle, 2010, Data Types and Sizes, Verkkodokumentti, <https://docs.oracle.com/cd/E19253-01/817-6223/chp-typeopexpr-2/index.html> Luettu 26.10

37 npm, Verkkodokumentti, < https://www.npmjs.com/package/multer> Luettu 13.10

38 NodeJS, Net, Verkkodokumentti, <https://nodejs.org/api/net.html> Luettu 27.10

39 Yandshun Tay, 2022, In-Depth Overview, Verkkodokumentti, <https://facebook.github.io/flux/docs/in-depth-overview/> Luettu 25.10

40 Thomas Hamilton, 2022, Unit Testing Tutorial – What is, Types & Test Example, Verkkodokumentti, <https://www.guru99.com/unit-testing-guide.html> Luettu 25.10

41 Thomas Hamilton, 2022, Integration Testing: What is, Types with Example, Verkkodokumentti, <https://www.guru99.com/integration-testing.html> Luettu 25.10