



Front end -sovelluskehitys teollisuusalalla

Jussi Junnila

Haaga-Helia ammattikorkeakoulu

Tradenomin tutkinto

Amk-opinnäytetyö

2023

Tiivistelmä

Tekijä(t) Jussi Junnila
Tutkinto Tradenomi
Raportin/Opinnäytetyön nimi Front end -sovelluskehitys teollisuuslalla
Sivu- ja liitesivumäärä 91 sivua + 6 sivua liitteet
<p>Opinnäytetyö kuvaa front end -puolen sovelluskehittäjän kohtaamia haasteita ohjelmoidessa ja kehittäessä sovellusta teollisuudenalan tarpeisiin. Tässä opinnäytetyössä analysoidaan koodia teknisellä tasolla, sekä pohditaan sovelluskehitystyöhön liittyviä aiheita kuten ketterä kehitys ja modernit tekniikat. Yksi keskeinen teema läpi tämän opinnäytetyön on ChatGPT-tekoäly, josta puhutaan työn seurantajaksojen aikana useaan kertaan.</p> <p>Aluksi tekijä kertoo lähtötilanteensa ja kuvailee työtehtävänsä. Tämän jälkeen tekijä raportoi 40 työpäivän pituisen jakson, jonka aikana hän kehittää teollisuuden sovellukseen haastaviakin ominaisuuksia ja kohtaa joka viikko erilaisia ongelmia. Raportointi on tehty päiväkirjatyyppisesti ja jokainen työviikko päättyy pohdintaan, jossa tarkastellaan kulunutta viikkoa kriittisesti vaihtelevalla katselukulmalla. Näitä katselukulmia on muun muassa sovelluksen suorituskyvyn optimointi, ChatGPT-tekoälyn hyödyntäminen, ketterä kehitys ja itsensä kehittäminen sovelluskehittäjänä.</p> <p>Lopulta tekijä pohtii omaa kehitystä sekä ChatGPT-tekoälyn merkitystä omassa työssään, mutta myös nousevien tekoälytyökalujen merkitystä lähitulevaisuudessa. Loppupohdinnassa tekijä kertoo ChatGPT-tekoälyn hyvistä ja huonoista puolista.</p>
Asiasanat Sovelluskehitys, ChatGPT, Front end, Teknologia, Tekoäly, Vektorigrafiikka.

Sisällys

1	Johdanto.....	1
2	Lähtötilanteen kuvaus.....	4
2.1	Oman nykyisen työ analysointi	4
2.2	Sidosryhmien esittely	6
2.3	Työpaikan vuorovaikutustilanteet.....	7
3	Seurantajakson raportointi viikkoanalyyseineen	9
3.1	Seurantaviikko 1	9
3.2	Seurantaviikko 2.....	18
3.3	Seurantaviikko 3.....	25
3.4	Seurantaviikko 4.....	35
3.5	Seurantaviikko 5.....	47
3.6	Seurantaviikko 6.....	54
3.7	Seurantaviikko 7	60
3.8	Seurantaviikko 8.....	69
3.9	Seurantaviikko 9.....	81
4	Pohdinta	83
	Lähteet.....	87
	Liitteet	92
	Liite 1. Terminologia	92
	Liite 2. Viikkoanalyysi 1 Testaus.....	97

1 Johdanto

Opinnäytetyössäni kerron työstäni ohjelmistokehittäjänä teollisuusallalla ja niistä haasteista, joita aloitteleva ohjelmistokehittäjä kokee hypätessään mukaan suuren sovelluksen kehittämiseen. Työskentelen yhdelle Suomen suurimmista teollisuuden automaatio- ja mittausratkaisuiden tarjoajista. Heillä on omaa tuotekehitystään oleva reunalaskentaa hyödyntävä digitaalinen tuote, joka kerää, analysoi ja seuraa teollisuuden laitteilta saatua dataa. Tuote mahdollistaa laitekokonaisuuksien älykkään järjestelmäohjauksen ja tilavalvonnan. Haluan syventää ymmärrystäni teollisuuden sovellusten kehittämisestä ja tarkastella tekemääni koodia kriittisesti.

Aloitin työt ohjelmistokehittäjänä huhtikuussa 2022. Työstän ohjelmiston käyttöliittymäpuolta, eli ohjelmistokehittäjänä työtehtäväni painottuvat ohjelmiston front end -puoleen. Front end -kehittäjältä vaadittavat taidot voivat vaihdella tehtävä- ja työpaikkakohtaisesti, mutta sanoisin, että on tiettyjä taitoja, joita voidaan tyypillisesti olettaa front end -sovelluskehittäjän hallitsevan. Näitä ovat muun muassa kielituntemus ohjelmointikielistä kuten JavaScript, CSS ja HTML, joita web-kehityksessä aktiivisesti käytetään. Lisäksi jokaisen sovelluskehittäjän tulisi osata hyödyntää ja käyttää versionhallintaa riippumatta siitä, onko hän back end -puolen vai front end -puolen kehittäjä. Nämä ovat hyvin lyhyesti ilmaistuna niitä tärkeimpiä taitoja.

Ohjelmistokehittäjän tärkein työkalu on tietokone ja tietokoneen käytön täytyisi olla ohjelmistokehittäjälle rutiininomaista. Ohjelmistokehityksessä hyödynnetään paljon erilaisia tietokoneelle asennettavia ohjelmapaketteja. Sovelluskehittäjän on osattava löytää tarvittavat asennettavat ohjelmat, valita niistä sopivat versiot ja saada kaikki asennukset toimimaan omassa työympäristössä.

Ohjelmoidessa hyödynnän päivittäin teknologioita kuten React, Redux-toolkit, Azure DevOps ja Git. React on JavaScript-sovelluskehys, jota käytetään rakentamaan käyttöliittymiä. Redux on tilanhallintakirjasto, jolla optimoidaan ja selkeytetään sovelluksen tilanhallintaa. Azure DevOps on Microsoftin sovelluskehitysprosessin seurantaan ja tueksi kehitetty palvelu. Git on koodin versionhallintajärjestelmä.

Minun tulee myös hallita työpaikallani käytetyt viestintäohjelmat, kuten Microsoftin Teams ja Outlook. Myös selkeä kirjallinen tuottaminen on tärkeää varsinkin sovelluskehityksessä tehtävän dokumentoinnin kannalta. Työpaikallani sovelluskehittäjän on osattava myös sujuvasti englantia. Tällä hetkellä puhuttuna työkielenämme on suomi, mutta tämä voi muuttua tulevaisuudessa mahdollisesti tapahtuvien rekrytointien myötä. Kirjoitamme koodia englanniksi ja suurin osa netistä löytyvästä alan oppimateriaalista on usein englanniksi, joten vaikka tiimin sisällä kommunikoimme tois-
taiseksi suomeksi, on englannin osaaminen välttämättömyys.

Kun tehdään töitä tiimissä, on tärkeää osata kommunikoida avoimesti ja tehokkaasti tiimin kesken. Vaikka sosiaaliset- ja kommunikaatiotaidot ovat myös muille ammattiryhmille ja erilaisissa tehtävissä työskenteleville tärkeitä taitoja, ovat ne tärkeitä taitoja myös tiimissä työskentelevälle sovel-luskehittäjälle.

Omia kehityksen kohteita on front end -osaamisen syventäminen sekä Jest-testaukseen tutustumi-nen. Kehittämäämme sovellukseen ollaan tuomassa automatisoidumpaa front end -testausta hyö-dyntäen Jest-testauskehystä. Front end -testauksen suunnittelu, toteutus ja ylläpito tulee mahdolli-sesti tulevaisuudessa kuulumaan minun työtehtäviini sovelluksen käyttöliittymän kehittämisen li-säksi. Front end -osaamisen syventämisen osalta toivon oppivani ymmärtämään ja rakentamaan monimutkaisempia koodikokonaisuuksia entistä selkeämmin ja luettavammin. Haluaisin, että kehit-tämäni käyttöliittymäkomponentit ovat mahdollisimman vähällä koodimäärällä logiikaltaan toimivia, sekä tyylittelyiltään selkeitä ja responsiivisia.

Haluan tässä opinnäytetyössä keskittyä nimenomaan ammatilliseen kehittymiseen front end -ohjel-mistokehityksen kannalta ja en tule raportoimaan syvällisesti työskentelemäni yrityksen asiakkuuk-sista saatikka palavereista asiakkaisiin liittyen. En myöskään raportoi yrityksen tarjoamien teolli-suuden laitekokonaisuuksien tarkemmasta toiminnasta tai teknisistä toteutuksista liittyen laitteis-toon (eng. hardware) tai ohjelmiston back end -puoleen, vaikka ymmärrystä ja oppia näihin teemoi-hin minulla kertyisi tämän opinnäytetyön aikana. Front end -puolen koodista tulen näyttämään esi-merkkejä havainnollistaakseni selkeämmin, miten olen ratkonut erilaisia ongelmia. Huomioitavaa on kuitenkin, että näissä esimerkkikoodissa muuttujanimiä ja dataa on voitu muokata salassa pi-dettävän tai arkaluonteisen tiedon suojaamiseksi.

Ohjelmistokehittäjät käyttävät jatkuvasti työssään erilaisia tietolähteitä, joista tyypillisiä ovat doku-mentaatiot, blogikirjoitukset ja foorumikeskustelut ohjelmistokehittäjien hyödyntämistä ohjelmointi-kielistä, -tekniikoista ja erilaisista ohjelmistokehityksessä hyödynnettävistä teknologioista. Itse käy-tän aktiivisesti esimerkiksi ”MDN Web Docs” -verkkosivustoa, joka on ohjelmistokehittäjäyhteisön ylläpitämä tietopankki kaikkeen web-kehitykseen liittyen. ”MDN Web Docs” -verkkosivut ovat olleet vuodesta 2005 asti olemassa ja sivuston sisältämän tiedon ajankohtaisuus ja hyvä laatu ovat seu-rausta 45 000 ohjelmistokehittäjän tarjoamasta panoksesta sivuston sisältöön. (MDN Web Docs 2023L.)

Toinen yleisessä käytössä oleva tietolähde, jota myös itse käytän, on ”stack overflow”-niminen verkkosivusto. Tämä on julkinen keskustelufoorumi, jossa tyypillisesti ohjelmistokehittäjät kysyvät toisiltaan kysymyksiä ja saavat niihin muilta ohjelmistokehittäjiltä vastauksia. ”Stack overflow”-si-vustolla vierailee yli 100 miljoonaa ihmistä kuukausittain, siellä on esitetty yli 21 miljoonaa kysy-mystä ja ohjelmistokehittäjät ovat raportoineet saaneensa apua ongelmiinsa ”stack overflowin”

kautta yli 50,6 miljardia (eli 50 600 miljoonaa) kertaa. (Stackoverflow.) Luvun suuruuden hahmottamiseksi kerrottakoon, että 15.4.2023 maapallon asukasluku oli noin 8 miljardia worldometer-sivuston mukaan, joka siis pyrkii aktiivisesti arvioimaan maapallon väkilukua (worldometer).

Uudeksi työkaluksi ohjelmoinnin tueksi olen ottanut ChatGPT-tekoälyn, joka on siis OpenAI nimisen yrityksen lanseeraama älykäs chatbotti. ChatGPT-tekoäly vastaa useisiin kysymyksiin hyvinkin informatiivisella ja järkevällä tasolla. Esimerkkejä ChatGPT-tekoälyn vastauksista erilaisiin kysymyksiin ja on nähtävillä OpenAI yrityksen ChatGPT-tekoälyn -julkaisublogissa. (Openai 30.11.2022.)

ChatGPT-tekoäly on loistava työkalu nopeaan tiedon etsimiseen, koodiongelmien ratkomiseen sekä myös koodin kirjoittamiseen. ChatGPT-tekoäly voi tosiaan kirjoittaa koodia puolestasi, kunhan osaat kertoa sille tarpeelliset seikat ja antaa täsmälliset ohjeet. (Openai 30.11.2022.) Löysin ChatGPT-tekoälyn ja sen potentiaalin samoihin aikoihin, kun aloitin kirjoittamaan tämä työtä. Työn seurantaviikkojen aikana voimme tarkastella ChatGPT-tekoälyn hyötyä ohjelmistokehityksen tukena.

Tämä opinnäytetyö sisältää paljon sovelluskehittäjän ammattisanastoa. Informaatioteknologian alalla lainasanat ja englanninkieliset termit ovat aktiivisessa käytössä työelämässä, mutta sovelluskehityksen kategoriassa tilanne on entistä vaikeampi. Kaikille termeille ole vakiintuneita suomalaisia vastineita ja sanojen suorat käännökset voisivat ajaa lukijaa enemmän harhaan. Olen kuitenkin pyrkinyt käyttämään ammattitermien suomenkielisiä vastineita ja tarvittaessa tarjoan myös termin englanninkielisen version suluissa. Termien selitykset löytyvät liitteestä 1.

Alla olevassa taulukossa 1 on jäsennetty, missä luvuissa tai seurantaviikkojen raporteissa käydään läpi työn tärkeimpien teemojen mukaisia asioita. Tärkeimmät teemat ovat kehittyminen front end -sovelluskehityksessä, sekä ChatGPT-tekoälyn hyödyntäminen työtehtävissä ja itsensä kehittämisessä.

Taulukko 1. Peittomatriisi päiväkirjaopinnäytetyön tekstinsisäisistä kytköksistä

<i>Oman ammatillisen kehittymisen tavoitteet</i>	<i>Tietoperustan luku raportissa</i>	<i>Seurantaviikko</i>	<i>Oman ammatillisen kehittymisen tulokset</i>
<i>Front end -sovelluskehitys</i>	<i>Luku 1, Luku 2.1</i>	<i>Luku 3 kaikki seurantaviikot</i>	<i>Luku 4 pohdinta sekä luku 3 kaikki seurantaviikot</i>
<i>ChatGPT-tekoälyn hyödyntäminen sovelluskehityksessä sekä itsensä kehittämisessä</i>	<i>Luku 1</i>	<i>Luku 3 seurantaviikot 1, 3, 4, 6, 7, 8</i>	<i>Luku 4 pohdinta</i>

2 Lähtötilanteen kuvaus

Minulla on alle vuoden työkokemus ohjelmistokehityksestä, eli olen alalla nuori. Kehitykseni on kuitenkin ollut nopeaa ja minulla on ohjelmointiin liittyvää harrastuspohjaa, kuten peliohjelmointi ja -kehitys, joka on edistänyt nopeaa oppimistani. Tässä luvussa käyn läpi nykyiset työtehtäväni ja niihin liittyvät seikat läpi tarkemmalla tasolla.

2.1 Oman nykyisen työ analysointi

Tehtävänimikkeeni on sovelluskehittäjä (eng. software developer). Työtehtäväni on kuitenkin pitkälti rajattu front end -puolen kehitykseen. Minun tulee aktiivisesti pohtia keinoja parantaa käyttöliittymää, käyttäjäkokemusta ja sovelluksen front end -koodin rakennetta ja selkeyttä. Tähän sisältyy niin käyttöliittymän visuaalinen suunnittelu, kuin itse sovelluksen front end -logiikan ohjelmointi. Otan aktiivisesti vastaan palautetta käyttöliittymästä ja tarkastelen palautetta kriittisesti pohtien, onko palautteen myötä syytä ryhtyä toimiin ja millaisella aikataululla. Keskustelen tiiminvetäjän kanssa kehitettävistä ominaisuuksista ja pohdimme yhdessä niiden priorisointia. Tosin tiiminvetäjällä on minua parempi ymmärrys, mitkä ominaisuudet ovat tärkeä saada tiettyyn aikamääreeseen mennessä valmiiksi, johtuen siitä, että hän on tekemisissä asiakasrajapinnassa.

Sovelluskehittäjän on hallittava käyttämänsä työvälineet, eli tietokoneen sujuva käyttö on sovelluskehittäjälle välttämättömyys. Työtietokoneeni käyttöjärjestelmänä on Windows, eli minun tulee pystyä toimimaan sujuvasti Windows-pohjaisella käyttöjärjestelmällä. Tietokoneen ja sen käyttöjärjestelmän asennus ovat myös hyviä taitoja, mutta en tarvitse niitä aktiivisesti työssäni. Voi kuitenkin tulla tilanne, että minun täytyy asentaa Linux-käyttöjärjestelmä teollisuuden laitekokonaisuuteen liitettävään tietokoneeseen ja tähän olen saanut perehdytyksen. Muutoin työtehtävissäni pitkälti riittää, että teollisuuden laite- ja komponenttiymmärrys on pintapuolista.

Ohjelmoidessa käytän Visual Studio Code -nimistä avoimen lähdekoodin koodieditoria. Olen käyttänyt erilaisia editoreja, mutta koen Visual Studio Coden miellyttävimmäksi vaihtoehdoksi. Tässäkin tapauksessa tärkeintä on, että pystyt työskentelemään sujuvasti valitsemallasi koodieditorilla.

Ohjelmointiosaamiseni on sellaisella tasolla, että pystyn itsenäisesti ratkojaan tavanomaisia ongelmia ja löytämään niihin toimivia ratkaisuja. Arkisia koodipulmia on mielestäni muun muassa objektien pyöryttäminen taulukoksi tai päinvastoin, tietyn arvon etsiminen monimutkaisesta JSON-objektista tai taulukosta tai mikä tahansa muu toiminto, joka irrallaan ohjelmakokonaisuudesta on melko yksinkertainen toteuttaa. Kehittäjän ei tarvitse ulkomuistista osata ohjelmoida kaikkia yksinkertaisiakaan asioita, mutta kehittäjän täytyy pystyä itsenäisesti etsiä keinot ratkaista nämä ongelmat. Mitä nopeammin, sen parempi.

Kyky kehittyä itsenäisesti ammatillisessa mielessä on myös tärkeää. Tämä on totta varsinkin minun kohdallani, sillä minulla ei ole työyhteisössäni kokenutta front end -kehittäjää minua ohjaamaan ja opastamaan. Koska koodiani ei jatkuvasti arvioi kokenut front end -kehittäjä, täytyy minun tarkastella omaa koodiani sitäkin kriittisemmin. Kaikki nämä taidot ovat kehittyneet suuresti siitä pisteestä, kun aloitin työskentelyn sovelluskehittäjänä.

Ohjelmointikieliä, joita työssäni tarvitaan päivittäin, ovat HTML, CSS ja JavaScript. Jokaisen näiden syntaksi tulee olla tuttua. Kehittämäni sovelluksen käyttöliittymä on rakennettu käyttäen React-kehystä, jonka syntaksissa yhdistyy kaikki edellä mainitut kolme kieltä. Työssäni React-osaaminen on siis ensisijaisen tärkeää. Jos osaa ennestään jo JavaScriptia ja HTML-kieltä, ei Reactin opettelu ole kuitenkaan vaikeaa. Itse opettelin Reactia ja JavaScriptia samanaikaisesti.

Tärkeää osaamista työssäni on myös Redux-osaaminen ja sen käyttö yhdessä Reactin kanssa. Redux on avoimen lähdekoodin tilanhallintakirjasto. Tilanhallinnalla tarkoitetaan, että sovelluksessa seurataan ja tiedetään, mitä asioita käyttäjä on painanut, mitä syötteitä tai komentoja hän on antanut ja mitä ylipäättänsä tehnyt sovelluksessa. Olennaiset tiedot tallennetaan tilanhallintaan, jotta sovellus osaa käyttäytyä toivotun mukaisesti riippuen näistä käyttäjän syötteistä, asetuksista ja toimista. Yksinkertaisena esimerkkinä kehittämässäni sovelluksessa on tilanhallinnassa tieto siitä, millä kielellä käyttäjä sovellusta käyttää. Tämä tieto pysyy tallessa, vaikka sovellus sammutettaisiin ja käynnistettäisiin uudelleen.

Kun aloitin työt, en ollut vielä toiminut lainkaan Reduxin kanssa, joten jouduin opettelemaan sitä itsenäisesti työn ohessa. Osaan tällä hetkellä toimia Reduxin kanssa perustasolla, mutta koen, että ymmärrykseni joistain toiminnoista tai yksityiskohdista ovat vajavaiset. Esimerkiksi middleware-toiminnot, thunk-funktiot sekä middleware-thunk-funktiot ovat minulle hieman epäselviä, eli en kykene selittämään näitä termejä auki. Olen tehnyt thunk-funktiota muun muassa hakemaan tietoa rajapinnasta Redux-storeen, mutta olen kopioinut koodin pohjan jostain toisesta funktiosta ja vain muokannut sitä uuteen tarkoitukseen miettimättä sen enempää, miksi tämä on niin kutsuttu thunk-funktio tai middleware-thunk-funktio. Nämä ovat niitä seikkoja, joihin olisi hienoa saada joku kokenut front end -kehittäjä pukemaan minulle asioita sanoiksi, jotta oppimiseni nopeutuisi.

Sovelluskehittäjällä on loputon oppimisen polku, sillä tavat ja standardit työskennellä kehittyvät ohjelmointialalla jatkuvasti. Minun tulisi oppia lisää JavaScriptia ja esimerkiksi monimutkaisempaa verkkosivun dokumenttiobjektimallipuun (eng. document object model) manipulaatiota ja näiden taitojen yhdistämistä React-sovelluskehityksen kanssa toimimiseen. Tulevaisuuden työtehtäviä ajatellen minun tulisi myös oppia front end -puolen testausta.

Tällä hetkellä osaamiseni suhteessa työn vaatimuksiin on riittävällä tasolla, sillä saan itsenäisesti tuloksia aikaiseksi. Tiiminvetäjän antaessa työtehtävän tai ehdotuksen työtehtävästä pystyn keskustelemaan kyseisen tehtävän potentiaalisista ongelmakohtista, aika-arviosta ja erilaisista tavoista suorittaa tehtävä. Mikäli tiiminvetäjä antaa minulle vapaat kädet toteutuksen suhteen, kykenen pohtimaan erilaisten ratkaisujen ja toteutustapojen hyviä ja huonoja puolia. Pystyn luomaan esityskelpoisia demoratkaisuja itseni käyttöön sekä muiden nähtäväksi. Kykenen tarjoamaan aikamääreeseen mennessä hiotun ja toimivan ratkaisun ja selittämään auki ohjelmoimani koodin.

Työtehtävieni lisäksi olen myös opastanut muutamaa kollegaani versionhallinnassa ja ollut avuksi heidän kohtaamien koodiongelmien kanssa. Mikäli tiimiimme tulisi nuori front end -kehittäjä, koen, että voisin opastaa häntä suoriutumaan front end -ohjelmoinnin työtehtävistä meidän työympäristössämme.

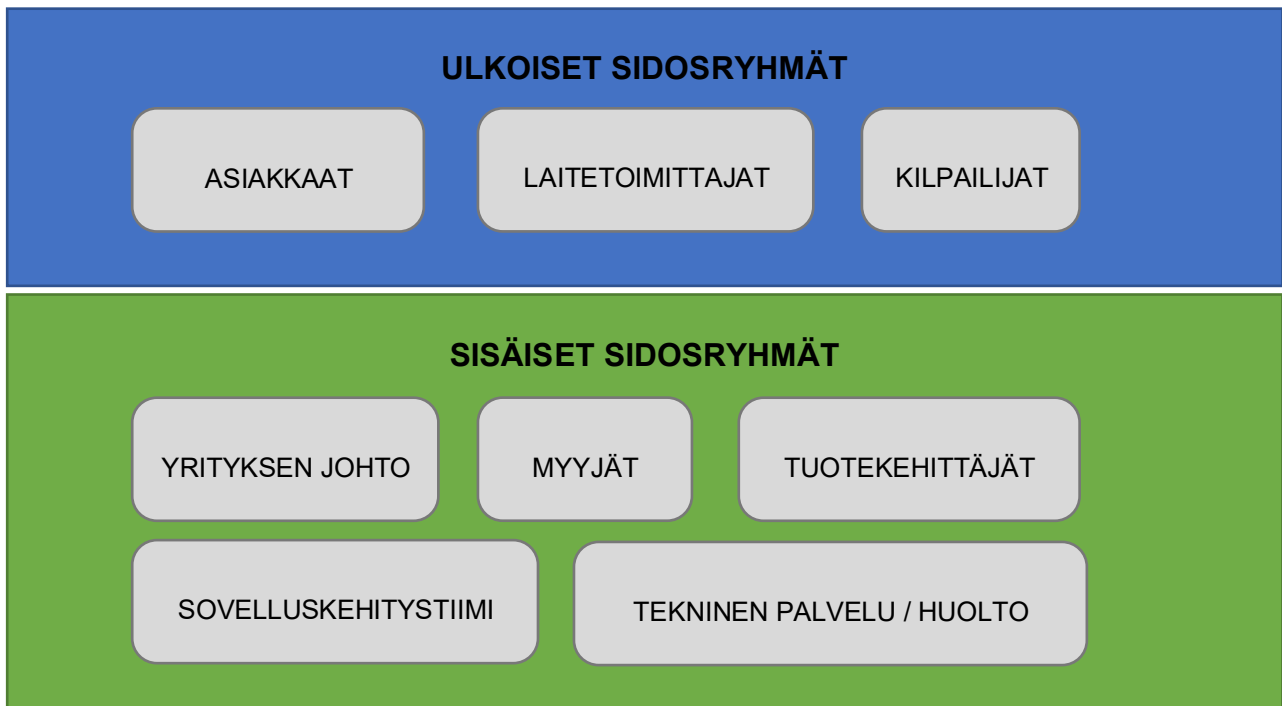
Koen osaamiseni olevan työtehtäviini liittyen taitavan suoriutujan ja asiantuntijan välimaastossa. Tämä siitä syystä, että etsin ongelmiin ratkaisuja itsenäisesti. Löydän minulle annetuista tehtävistä mahdollisia ongelmakohtia ja tuon ne esiin esittäen samalla omat ratkaisuehdotukset kyseisiin ongelmakohtiin. Tunnen taitoni sekä rajani ja osaan avoimesti kertoa, mikäli jokin tehtävä menee yli osaamiseni tason ja tarvitsen sen suorittamiseen lisää aikaa tai apua. Kykenen suoriutumaan suurimmasta osasta minulle annetuista front end -puolen ohjelmointitehtävistä järkevässä ajassa. Minulle ei tarvitse myöskään antaa tehtäviä, vaan kykenen tarvittaessa itse havaitsemaan kehittämäni sovelluksen ongelmakohtia ja tekemään niihin korjauksia tai dokumentoimaan korjausehdotuksia täysin itsenäisesti, eli luomaan itselleni lisää merkityksellisiä tehtäviä.

2.2 Sidosryhmien esittely

Ulkoisia sidosryhmiä on muun muassa asiakkaat, laitetoimittajat ja kilpailijat. Tämän opinnäytetyön kannalta näiden tarkempi läpikäynti ei ole olennaista, sillä en tule raportoimaan mitään asiakkaisiin tai ulkoisiin sidosryhmiin liittyvää suojatakseni salassa pidettäviä ja arkaluonteista seikkoja. Omiin työtehtäviini kuitenkin vaikuttaa ulkoiset sidosryhmät siinä mielessä merkittävästi, että esimerkiksi asiakkaiden suunnalta tulee toiveita kehittämäni sovelluksen suuntaan. Omiin työtehtäviini ei kuitenkaan lukeudu asiakasrajapinnassa toimiminen, joten kaikki tämän kaltainen tieto tulee korviini eri reittejä, kuin suoraan asiakkaalta.

Sisäisiin sidosryhmiin kuuluu yrityksen työntekijät sekä konserni kokonaisuudessaan, johon yritys kuuluu. Oman työn kannalta olennaisia sisäisen sidosryhmän henkilöitä tai ryhmiä ovat myyjät, johto, tuotekehittäjät, tekninen palvelu ja huolto sekä oma sovelluskehitystiimi. Työni jatkuvuuden kannalta on tärkeää, että yrityksen työntekijät uskovat sovelluskehityksen kannattavuuteen. Mikäli

esimerkiksi johto ei uskoisi sovelluskehityksen kannattavuuteen, ei siihen todennäköisesti allokoitaisi resursseja ja täten työtehtäväni mahdollisesti lopetettaisiin. Mikäli myyjät eivät usko sovelluskehityksen kannattavuuteen tai itse tuotteeseen (sovellukseen), vaikeuttaa se tuotteen myyntiä ja täten vähentää sen kannattavuutta. Mikäli sovelluskehitystiimi ei itse usko omaan tekemiseen tai omaan tuotteeseen, on muun yrityksen vaikea tällöin myöskään uskoa tuotteen kannattavuuteen. Alla oleva kuva 1 havainnollistaa sidosryhmiä.



Kuva 1. Sidosryhmät

2.3 Työpaikan vuorovaikutustilanteet

Olen pääsääntöisesti ja päivittäin tekemisissä sovelluskehitystiimin kanssa. Sovelluskehitystiimi pitää viikoittain useampia lyhyitä palavereja, joiden tarkoitus on pitää tiimi ajan tasalla toistensa tekemisistä sekä ajan tasalla ajankohtaisista ja tärkeistä asiakastapauksista ja -tilanteista. Näitä lyhyitä palavereja kutsutaan "daily sprinteiksi", eli päivittäispalavereiksi. Muutaman viikon välein pidämme pidemmän ja kokonaisvaltaisemman palaverin, jossa kertaamme mitä olemme saaneet aikaiseksi ja luomme katseet seuraaviin viikkoihin: asetamme tavoitteita ja otamme uusia tehtäviä työn alle. Tätä kahden viikon välein pidettävää pidempää ohjelmistokehityspalaveria kutsutaan nimellä "sprint plan".

Olen harvemmin tekemisissä muiden kuin sovelluskehitystiimin työntekijöiden kanssa ammatillisessa mielessä, sillä mikäli sovelluskehitystiimin ulkopuolisilla henkilöillä on kysymyksiä kehittä-

määmmme IIoT-reunalaskentaa hyödyntävään digitaaliseen alustaan liittyen, ottavat he pääsääntöisesti yhteyttä sovelluskehitystiimin tiiminvetäjään. En myöskään koe, että minulla olisi juurikaan mitään annettavaa tiettyihin sovelluskehitystehtävien ulkopuolisiin aiheisiin kuten asiakkuuksiin, myyntiin, muiden käyttämiin työkaluihin tai laiteyksiköiden tekniseen suunnitteluun.

3 Seurantajakson raportointi viikkoanalyysineen

Raportoin työstäni ohjelmistokehittäjänä 40 työpäivän ajan. Tulen kertomaan kirjoittamastani koodista ja näyttämään esimerkkejä ja tapoja lähestyä erilaisia kohtaamiani pulmia. Viikkoanalyysissä pureudun viikon tapahtumiin ohjelmistokehittäjän näkökulmasta. Saatan tutkia kirjoittamaani koodia kriittisesti, etsiä vaihtoehtoisia toimintatapoja tai pohtia omaa sekä tiimin toimintaa ketjessä ohjelmistokehitysympäristössä. Viikkoanalyysien tarkoitus on syventää osaamista ja etsiä lisää tietoa aiheista, jotka ovat sillä viikolla olleet tekemisessä läsnä.

Tässä opinnäytetyössä, kun sanon kehittäväni "sovellusta", tarkoitan kehittäväni IIoT-reunalaskentaa hyödyntävää digitaalista alustan front end -puolta.

Todettakoon vielä, että kaikkien koodia sisältävän kuvien koodia on muokattu niin, että ne eivät paljasta arkaluonteista tietoa. Suurimmaksi osaksi vain muuttujanimiä ja muuttujien arvoja on manipuloitu. Lisäksi koodia sisältävistä kuvista ei selviä toteutuksellisia kokonaisuuksia, sillä ne ovat vain pieniä osia suuressa kokonaisuudessa. Koodia sisältävillä kuvilla haluan selkeyttää, mistä puhun ja kerron viikkoraporteissani.

3.1 Seurantaviikko 1

Maanantai 13.2.2023

Olen työstänyt sovelluksen käyttöliittymään kaavailtua toiminallisuutta muuttaa erilaisten laiteyksiköiden sisällä olevien laitteiden parametreja. Laitekokonaisuus eli laiteyksikkö, pitää sisällään laitteita, jotka voivat esimerkiksi mitata erilaisia kaasupitoisuuksia tai lämpötiloja sekä puhdistaa ja käsitellä kaasuja. Ilman tätä ominaisuutta näitä laiteparametreja voisi päivittää ainoastaan lähdekoodissa olevia konfiguraatiotiedostoja muuttamalla, joka on työlästä ja vaatii aina koodin uudelleen ajamisen tuotantoon.

Käyttöliittymässä voidaan ottaa esille yhden laiteyksikön laitteet kortteina, jotka sisältävät erinäistä tietoa. Olen rakentanut näille korteille painikkeen, josta aukeaa modal-ikkuna, joka hakee rajapinnasta parametritiedot kyseiselle laitteelle. Modal-ikkunaan renderöityy käyttäjälle tekstinsyöttökenttiä sen mukaan, mitä get-pyyntö palauttaa. Esimerkiksi alla olevassa kuvassa 2 näkyvä objektilista voitaisiin hakea backendistä ja tämän perusteella haluamme renderöidä käyttäjälle käyttöliittymää. Kuvassa 3 näkyy rakentamani konfiguraatiomodal-ikkuna, joka hakee siis arvonsa rajapinnasta, jotta samaa modal-ikkunakomponenttia voidaan käyttää mille tahansa laitteelle. Kuvissa näkyvä data on keksittyä esimerkkidataa.

```
[
  {
    "parameterName": "Parameter 1",
    "type": "numeric",
    "value": 100
  },
  {
    "parameterName": "Parameter 2",
    "type": "numeric",
    "value": 100.6
  },
  {
    "parameterName": "Parameter 3",
    "type": "dropdown",
    "dropValues": ["1", "2", "3"],
    "value": 1
  },
  {
    "parameterName": "Parameter 4",
    "type": "dropdown",
    "dropValues": ["yes", "no"],
    "value": "no"
  }
]
```

Kuva 2. Esimerkki laiteparametreista, jotka voitaisiin hakea rajapinnasta.

Device 1 parameter configuration

Parameter Name	Old Value	Current Value
Parameter 1	100	101
Parameter 2	100.6	100.6
Parameter 3	1	2
Parameter 4		no

Below the parameters is a virtual keyboard with keys: ``, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, -, =, backspace, tab, q, w, e, r, t, y, u, i, o, p, [,], \, caps, a, s, d, f, g, h, j, k, l, ;, ', < enter, shift, z, x, c, v, b, n, m, ., /, shift, .com, @, and a large empty text input field.

At the bottom are two buttons: **Save** and **Close**.

Kuva 3. Parametrien konfiguraatiomodal-ikkuna.

Tiiminvetäjä ehdotti, että kuvassa 3 näkyvä tallenna-nappi ("save") ei lähetä rajapintaan kaikkien parametrien tietoja, vaan ainoastaan niiden parametrien, joita on muutettu. Tällöin rajapintaa ei tulisi pommitettua turhilla pyynnöillä, jos modal sisältää vaikka 10 kpl parametrejä, mutta käyttäjä muuttaa niistä vain yhtä. Mielestäni ajatus kuulosti järkevältä, mutta jokaisen parametrin perään en

mielelläni laittaisi erillistä save-nappia. Mielestäni käyttäjän näkökulmasta on työläämpää painaa tallenna-nappia erikseen jokaisen muutetun parametrin kohdalla, kuin tallenna-napin painaminen kerran kaikkien muutosten jälkeen. Lähdin suunnittelemaan siis toteutusta, jossa yhdellä tallenna-napilla saadaan lähetettyä tallennuspyyntö vain niistä parametreistä, joita on muutettu.

Kuvassa 4 voimme nähdä minun aikaisemmin rakentaman toteutuksen, jota lähdin muuttamaan. Vanhassa saveParameters-funktiossa lähetin post-pyyntöä rajapintaan jokaisesta parametrin riippumatta siitä, oliko käyttäjä antanut parametrille uutta arvoa.

```
const saveParameters = async () => {
  const state = parametersState;
  let formattedState = [];
  try {
    state.map((parameter, index) => {
      let obj = {};
      obj.parameterName = parameter.id;
      obj.hwId = parameter.hwId;
      obj.value = parameter.value;
      formattedState.push(obj);
    })
    await Promise.all(formattedState.map(async (parameter) => {
      const payload = { parameter };
      await DeviceParameters.post(config.CONFIGURATION_URL, payload);
    })))
    setErrorMsg("");
    setParametersState([]);
    onHide();
  } catch (e) {
    console.log(e);
    setErrorMsg("Error. Save not successful");
  }
}
```

Kuva 4. Vanha saveParameters-funktio.

Kuvassa 5 on koodaamani uusi funktio, joka lähettää vain niistä parametreistä post-pyyntöä, joille käyttäjä on antanut uuden arvon. Asetin kuvassa 2 näkyvien parametrien tiedot talteen taulukko-muuttujaan nimeltä oldValues. FormState pitää sisällään tiedon kaikista modal-ikkunassa olevissa parametreistä muutoksineen. Tämän jälkeen käyn map-funktion avulla formStaten tiedot läpi ja vertaan sen arvoja oldValues listan arvoihin. Mikäli arvot poikkeavat toisistaan, on käyttäjä päivittänyt arvoa, ja pistämme tämän parametrin tiedot talteen uuteen taulukkoon nimeltä parameterListContainingOnlyChangedParameters. Kun kaikki formStaten sisältämien parametrien arvot on käyty läpi ja verrattu niitä oldValuesissa tallessa olevien parametrien alkuperäisiin arvoihin, on meillä tiedossa kaikki ne parametrit, joita on muutettu. Nyt pystymme tekemään post-pyyntöä ainoastaan näille muutetuille parametreille.

```

const saveParameters = async () => {
  setSaving(true)
  const state = formState;
  let parameterListContainingOnlyChangedParameters = [];
  let formattedState = [];

  //check which parameters were changed and push them on
  parameterListContainingOnlyChangedParameters
  state.map((changedParameter, index) => {
    if (changedParameter.value !== oldValues[index].value) {
      parameterListContainingOnlyChangedParameters.push(changedParameter)
    }
  })

  //format changed parameters to contain only necessary values, push them to
  formattedState list
  try {
    parameterListContainingOnlyChangedParameters.map((parameter, index) => {
      let obj = {};
      obj.parameterName = parameter.parameterName;
      obj.hwId = hwId;
      obj.value = parameter.value;
      formattedState.push(obj);
    })
    console.log("saving these parameters...", formattedState)
    //save paramaters one at a time
    await Promise.all(formattedState.map(async (parameter) => {
      const payload = { parameter };
      await DeviceParamaters.post(config.DATA_URL, payload);
    }))
    setSaving(false)
    setErrorMsg("");
    setFormState([]);
    onHide();
  } catch (e) {
    e.message ?
      setErrorMsg("Error. " + e.message)
      :
      setErrorMsg("Error. Save was not successful");
    setSaving(false)
  }
}

```

Kuva 5. Päivitetty saveParameters-funktio.

Sain parametrien konfiguraatiomodal-ikkunan viimeistelyä vaille valmiiksi ja se jäi odottamaan lopputestejä, jotka suorittaisinkin, kun back end olisi valmis parametrien osalta. Luomani parametrien tallennus lähettää pyynnöt front endistä, mutta back end ei vielä osaa käsitellä pyyntöjä. Back end -ohjelmointi ei kuulu minun tämänhetkisiin työtehtäviini.

Tiistai 14.2.2023

Yksi pidempiaikainen ja haastava tehtäväni on ollut rakentaa niin kutsuttua 'graafista käyttöliittymää' sovelluksessa käsiteltäville laiteyksiköille. Käytännössä tämä tarkoittaa ainakin näin aluksi teknistä kaaviopiirros -tyylistä vektorigrafiikkakuvaa, jossa olisi paineltavia elementtejä laitteen ohjaamiseksi. Tulen jatkossa käyttämään vektorigrafiikkakuvasta lyhennettä SVG-kuva. Ehkä joskus tulevaisuudessa ulkoasua voitaisiin hioa näyttävämmäksi, mutta tärkeintä olisi nyt aluksi saada jonkinlainen toimiva ja monistettava ratkaisu ohjelmoitua.

Ajatus on luoda melko yksinkertaisia piirroksia sopivalla ohjelmalla ja liittää piirroksen eri osiin tunnistekenttiä (eng. ID), joihin päästään sitten koodissa käsiksi. SVG-kuva haetaan back endin rajapinnasta front endin käyttöön. Front endin koodin tulisi olla järkevästi rakennettu, jotta se osaisi liittää esimerkiksi aina tietynlaisen funktion tietylle SVG-kuvan yksittäiselle elementille ID:n perusteella. Ajatellaan, jos vaikka kymmenellä eri laiteyksiköllä on jokaisella omat SVG-kuvat, mutta jokaisessa kuvassa on yksi yhteinen elementti. Frontin koodin tulisi osata liittää tietynlainen funktio ja/tai tyylittely tuolle elementille selkeällä ja yksinkertaisella funktiolla.

Olen tuskailut ja tuskailen edelleen sovelluksen monimutkaisen logiikan yhdistämistä back endistä haettavan SVG-kuvan sisäisiin elementteihin. Olen saanut tiimin ulkopuolelta kokeneemman sovel-luskehittäjän apua, jonka avulla olen päässyt alkuun. Tällä hetkellä olen saanut lisättyä SVG-kuvaan onClick-funktiota, joilla voi ohjata laiteyksikön tiettyjä toimintoja ja näitä toimintoja käyttäessä SVG-kuvaan ilmestyy animaatioita. Hyvin tärkeä seikka seuraavaksi olisi saada yhdistettyä kaikki oleellinen tilatieto Reduxilla toteutetusta sovelluksen tilanhallinnasta mukaan graafiseen käyttöliittymään. Esimerkkinä, jos sovellus suljetaan, meillä täytyy sovelluksen uudelleenkäynnistytksen yhteydessä olla tieto siitä, mitkä kuvan painikkeista ovat aktiivisia ja mitä kaikkia animaatioita kuvassa tällä hetkellä pyörii.

Tiistaina en edennyt merkittävästi tämän tehtävän tiimoilta ja käytin työaikaani myös kollegan auttamiseen. Kollega työstää back end -koodia käyttäen C#-kieltä, joka on minulle heikosti tuttu ohjelmointikieli. Olen koodannut Unity-nimisellä ohjelmalla pelejä ja sitä kautta tunnen C# jonkin verran. Kykenen silti tarjoamaan apua koodillisissa ongelmissa ja joskus ulkopuolisen silmät näkevät jotain, mitä omat eivät. Kollegani sai tyyppivirheviestiä, joka johtui siitä, että muuttujalle, joka oli lista, yritettiin syöttää objektityypistä sisältöä. Hyödynsin avunantoon ChatGPT-tekoälyä, joka on osoittautunut voimakkaaksi työkaluksi koodatessa.

Päivän aikana tein myös pieniä muutoksia suunnittelutyökalulla SVG-kuvaan, jota tullaan käyttä-

mään graafisessa käyttöliittymässä. Muutosten jälkeen päivitetty SVG-kuva täytyy puskea käyttämämme Azure Storageen, jotta se tulee back endin käyttöön ja pystyn front endin koodilla hakemaan sen sovelluksen käyttöön.

Keskiviikko 15.2.2023

Kollegani oli saanut parametrien tallennuksen back endin osalta toimimaan ja lähdin itse testaamaan, lähteekö tallennuspyyntö oikein front endiltä back endille ja onnistuuko tallennus. Tallennuspyyntö antoi vastaukseksi virhekoodin "400 bad request", eli jokin oli pielessä. Tutkailin kollegan tekemää testitallennuspyyntöä, jolla hän oli onnistunut tallennuksen tekemään ja vertasin sitä omaani. Huomasin välittömästi, että minulla arvokentät lähtevät viestikuormassa (eng. payload) merkkijonotyyppisinä (eng. string), kun kollegan tekemissä testeissä arvokentät olivat numerotyyppisiä. Korjasin front endiltä lähtevän viestikuorman arvokentät numeroiksi, mutta sain edelleen vastaukseksi "400 bad request".

Nyt jouduinkin tovin pohtimaan, että mikä tässä menee pieleen, sillä kollegoiden tekemät testit toimivat moitteetta. Lopulta huomasin virheen, joka on havaittavissa maanantain 13.2.2023 raportissa kuvassa 5. Olin asettanut viestikuorman sisällön vahingossa ylimääräisen objektin sisälle seuraavalla koodilla: `const payload = {parameter}`. Koska tässä parameter on jo itsessään objekti, alustuu viestikuorma seuraavalla tavalla: `{{"parameterName": "Parameter 1", "value": 100}}`. Korjasin viestikuorman oikeanlaiseksi eli `{"parameterName": "Parameter 1", "value": 100}` ja tallennuspyyntö meni läpi ja palautti viestin "200 OK success".

Tämän jälkeen palasin työstämään graafista käyttöliittymää. Iso osa ajasta kului edelleen koodin tutkailuun ja erilaisten lähestymistapojen pohdintaan. Sain kaivettua sovelluksen Redux-tilanhallinnasta tietoa laitteiden erilaisista tiloista ja aloin hahmotella, miten vien tiedon järkevästi graafiselle käyttöliittymälle. Kuvassa 7 näkyy rakentamani funktio, jolla käsittelen kuvan 6 arvoja, jotka on haettu sovelluksen Redux-tilanhallinnasta. Tilatieto sattui olemaan tässä tapauksessa objekti, joka sisälsi objekteja, joten läpikäymiseksi käytin JavaScriptin `Object.values`-metodia. Tulevaisuudessa kuvan 7 koodipätkää täytyy mahdollisesti jotenkin selkeyttää, mutta tässä vaiheessa se saa kelvata.

Koska tarkoitukseni on hyödyntää kuvassa 6 näkyvän datan `commandName`-arvoja myös käyttöliittymän tyyllittelyyn, täytyy minun muokata niitä. Tyyllittelyn toteutan niin, että `commandName` arvo tullaan antamaan jollekin SVG-kuvan sisäiselle elementille luokka-attribuuttina. Luokalla taas voi tavalliseen tapaan olla liuta erilaista CSS-koodia sitä koristamaan. HTML-kielessä luokka-attribuutti ei saa sisältää välilyöntejä, joten tämän vuoksi joudun muokkaamaan kuvassa 6 näkyviä `commandName`-arvoja. Eli esimerkiksi `commandName`-arvo "Gas 1" muokattaisiin muotoon "gas1",

jotta se olisi CSS-ohjelmointikielen nimeämiskäytänteiden mukainen.

Onneksi JavaScriptin `replace()`-funktiolla välilyöntien poistaminen merkkijono-tyyppisestä muuttujasta ei ole vaikeaa. Kuvassa 7 näkyy, kuinka alustan `commandName`lle uuden arvon seuraavasti: `commandName: singleValveStatus.commandName.replace(/ +/g, "").toLowerCase()`.

Tämän jälkeen esimerkiksi arvo "Gas 1" olisikin "gas1".

```
const uiCommandStatusList = {
  "Device 1": {
    "Gas 1": {
      "hwId": "DeviceID",
      "commandName": "Gas 1",
      "isCommandOpen": false,
      "group": "group 1"
    },
    "Gas 2": {
      "hwId": "DeviceID",
      "commandName": "Gas 2",
      "isCommandOpen": false,
      "group": "group 1"
    }
  },
  "Device 2": {
    "Command X": {
      "hwId": "DeviceID",
      "commandName": "Command X",
      "isCommandOpen": false,
      "group": "group 2"
    }
  }
}
```

Kuva 6. Sovelluksen Redux-tilanhallinnasta haettu objekti.

```
useEffect(() => {
  let newValveStatusList = []
  uiCommandStatusList &&
    Object.values(uiCommandStatusList).map((deviceStatus, index) => {
      Object.values(deviceStatus).map((singleValveStatus, index) => {
        if(singleValveStatus.isCommandOpen === true) {
          let formattedSingleValveStatus = {
            ...singleValveStatus,
            commandName: singleValveStatus.commandName.replace(/ +/g,
            "").toLowerCase()
          }
          newValveStatusList.push(formattedSingleValveStatus)
        }
      })
    })
  setValveStatusList(newValveStatusList)
}, [uiCommandStatusList])
```

Kuva 7. `UseEffect()`-Funktio, joka käy läpi kuvan 6 dataa.

Torstai 16.2.2023

Päätin muokata kuvan 7 koodia hieman. Ensinnäkin ymmärsin, että saatoin tarvita command-Namea alkuperäisessä muodossaan logiikan rakentamiseen, joten en halunnutkaan kajota sen kirjoitusasuun. Toisekseen nimeämiseltään ei käynyt järkeen, että commandName muuttujan arvo syötettäisiin tyyli luokaksi jollekin elementille. Päätin siis luoda uuden muuttujan nimeltä graphicsGlass, jolle annan arvoksi muokatun commandNamen arvon. Eli commandName pysyy alkuperäisessä muodossaan commandName muuttujassa, ja meillä on uusi muuttuja nimeltä graphicsGlass. Lisäksi muutin replace-metodille syötettyä "regular expression" -lauseketta poistamaan myös kenoviivat.

```
let formattedSingleValveStatus = {
  ... singleValveStatus,
  graphicsClass: singleValveStatus.commandName.replace(/\s|\\/g,
  "").toLowerCase()
}
```

Kuva 8. Muokkasin kuvan 7 koodia.

Tämän myötä pääsinkin etenemään tehtävässä taas askeleen eteenpäin. Kirjoitin CSS-koodia lisää eri graphicsClass-luokille, sekä rakensin logiikkaa, jolla näitä luokkia lisätään SVG-kuvan elementeille. Koitan lyhyesti avata, miten luokan lisääminen back endistä haetulle front endissä renderöitävälle elementille voidaan muun muassa tehdä. Back endistä tulee esimerkiksi merkkijono, joka muunnetaan front endissä dokumenttiobjektiksi (eng. document object). Tämä renderöidään käyttöliittymään. Seuraavaksi haetaan tästä dokumenttiobjektista elementtejä niiden tunnistekenttien (eng. ID) perusteella käyttäen esimerkiksi querySelector-metodeita, joita tavallisessa JavaScriptissä paljonkin käytetään. Koska olemme Reactin ympäristössä, tallennetaan referenssi haettuun elementtiin Reactin useRef-hookin avulla. Tämän jälkeen voimme liittää etsittyyn elementtiin luokkia esim. setAttribute()-metodilla.

Viikkoanalyysi 1

Viikon aikana sain viimeistelyä parametrikonfiguraatiomodal-ikkunan ja pääsin eteenpäin pidempi-
aikaisessa tehtävässä eli graafisessa käyttöliittymässä.

Yksi katselukulma, jonka haluan tuoda viikkoanalyyseissä esiin, on oman koodin kriittinen tutkiminen. Koska kollegani eivät kykene kovinkaan syvällisellä tasolla minun koodiani analysoimaan, päätin kokeilla ChatGPT-tekoälyä. Kysyin ChatGPT-tekoälyltä mielipidettä kuvan 5 koodista ja ChatGPT-tekoäly ehdotti minulle kolmea muutosta:

1. "Use Array.prototype.filter instead of Array.prototype.map to filter the list of changed parameters."

2. "Use Array.prototype.map to format the list of changed parameters."
3. "Use a for loop instead of Array.prototype.map to save parameters one at a time."

Eli suomennettuna käytä filter()-metodia map()-metodin sijaan. Lisäksi formatoiduin listan tekemiseen hyödynnä map()-metodia. Kolmanneksi parametrien tallennukseen hyödynnä for-toistorakennetta map()-metodin sijaan. ChatGPT-tekoäly toteaa, että muutokset johtavat parempaan suorituskykyyn, sillä emme tällöin luo turhia taulukoita käyden niitä läpi useaan kertaan. Se, paljonko nämä muutokset parantavat suorituskykyä, on täysin riippuvainen tekijöistä kuten liikkuvan datan määrä näissä funktioissa ja laskentaa suorittavan tietokoneen tehot.

Lähdin selvittämään, pitävätkö nämä väitteet paikkansa ja en löytänyt yhdenmukaista saatikka johdonmukaista mielipidettä näihin. Esimerkiksi suoritin itse nopeustestin ChatGPT-tekoälyn ensimmäisestä ehdotuksesta ja testin mukaan minun kirjoittama map()-metodia hyödyntävä ratkaisu olisi nopeampi, kuin ChatGPT-tekoälyn ehdottaman filter()-metodin käyttäminen. Testeissä käytin testidatana taulukkoa, joka sisälsi 10 objektia, joilla jokaisella oli muuttujat nimeltä "name" ja "value". Testin koodi ja sen tulokset löytyvät liitteestä 2.

Koen, että minun kooditapauksessani (kuva 5) kyseisen koodipätkän suorituskykyoptimointi on hiusten halkomista, sillä funktioni pyörittää erittäin pientä määrää dataa. Kun datan määrä on pieni, optimointi saattaisi nopeuttaa funktion suorittamista esimerkiksi 0.01 sekuntia. Deepak Gupta -niminen ohjelmoija on suorittanut myös testejä JavaScriptin eri taulukkometodeilla ja hänen testeisensä hyvin pienillä dataseteillä map()- ja filter()-metodien ero datan läpikäynnissä toistorakentein oli 0.01 sekuntia (Gupta 2018). Tosin hänen testissensä filter-metodi oli nopeampi, kun taas minun testissäni map()-metodi oli nopeampi. Testimme on toteutettu eri datalla, eri vuosina, eri tehoisilla tietokoneilla ja eri JavaScript-versioilla, joten niiden vertaaminen keskenään ei ole tieteellisesti varteenotettavaa.

Aiheesta tietoa etsiessäni muun muassa "senior software engineer" Shane Tarleton peräänkuuluttaa map()-metodin käyttöä for-toistorakenteen sijaan. Tärkeimmäksi argumentiksi hän nostaa, että funktionaalisessa-ohjelmoinnissa, jota map()-metodin käyttö edustaa, on koodia helpompi lukea ja sille on helpompi luoda automatisoitua testausta. (Tarleton 2022.) Tällä tavalla ajateltuna ChatGPT-tekoälyn optimointiehdotusta numero 3 ei myöskään ole välttämättä järkevä toteuttaa.

Koodin optimoinnissa on otettava huomioon muitakin seikkoja, kuin vain performanssi. Esimerkiksi koodin luettavuus ja johdonmukaisuus muun sovelluksen koodin kanssa ovat seikkoja, joita ei missään nimessä voi sivuuttaa. Oma mielipiteeni on, että ne ovat tärkeämpiä seikkoja, kuin suorituskykyoptimointi, sillä selkeää koodia on helpompi lukea ja ymmärtää. Tällä on suurta merkitystä esimerkiksi, kun joku ulkopuolinen lukee koodiasi tulevaisuudessa ja haluaa tehdä siihen muutoksia.

Jos töissä tutkailisin jokaista kirjoittamaani funktiota suurennuslasin läpi ja miettin erilaisia tapoja tehdä sama asia uudestaan, mutta hieman paremmin, en varmaankaan saisi paljoa aikaiseksi viikon aikana. Koodatessani uusia asioita ja onnistuessani saamaan jotain toimimaan tulen kuitenkin koko ajan oppineeksi lisää koodauksesta ja erilaisista tavoista tehdä asioita. Taito kirjoittaa selkeää koodia ja optimoitua koodia tulee ajan kanssa: koodaamaan oppii koodaamalla.

3.2 Seurantaviikko 2

Maanantai 20.2.2023

Laiteyksikkö, jolle olen kehittämässä graafista käyttöliittymää ei ollut tänään toiminnassa syistä, joita en tiennyt. Kollegani eivät myöskään saaneet laiteyksikköön yhteyttä, joten tiesin, että vika ei ole vain minun tietokoneellani tai verkkoyhteyksissä. En pysty ohjelmoimaan tehokkaasti graafista käyttöliittymää ilman tuota laiteyksikköä, sillä tarvitsen juuri tuolta laiteyksiköltä tietoa rajapintakyselyiden kautta front end -koodin kirjoittamiseen. Päätin käyttää päivän Jest-testauskehityksen opiskeluun.

Jest on JavaScript-testauskehys ja kuten voi päätellä, se on luotu työkaluksi JavaScript-koodin testaamiseen (Meta Platforms 2023a). Se pyrkii olemaan yksinkertainen käyttää ja se toimii muun muassa suosittujen kehysten tai kirjastojen kanssa kuten React, Vue ja Angular (Meta Platforms 2023a). Yksinkertaisuutta Jestiin tuo se, että sen käyttöönotto ei vaadi suurimassa osassa projekteja minkäänlaista konfiguraatiota ja Jestillä on kattava sekä ajantasainen dokumentaatio (Platforms 2023). Monet tunnetut yritykset ja sovellukset käyttävät Jestiä. Esimerkkeinä mainittakoon Spotify, Instagram, Facebook ja Airbnb (Meta Platforms 2023a).

En avaa tässä kaikkea Jestistä oppimaani yksityiskohtia myöten, mutta käydään läpi yksi testauksen peruspalikoista eli 'matcher'-metodi. Matcherin avulla kerromme esimerkiksi, mitä oletamme jonkin funktion palauttavan tai mitä oletamme jonkin arvon olevan (Meta Platforms 2023b). Annan yksinkertaisen esimerkin: Haluamme testata, että funktio, jonka tarkoitus on laskea kaksi numeraalista arvoa yhteen, toimii oikein.

```
function Sum (num1, num2) {return num1 + num2}
```

Yksinkertainen Jest-testi voitaisiin kirjoittaa muun muassa seuraavasti.

```
test('function() Sum Test', () => {
  expect(Sum(4, 7)).toBe(11)
})
```

Testille annetaan parametreinä ensin testin nimi ja seuraavaksi callback-funktio, joka sisältää testin logiikan (Meta Platforms 2023b). Tämä kyseinen keksimäni esimerkkitestin antaa Sum()-funktiolle arvot 4 ja 7 ja käyttää 'toBe'-matcheria katsoakseen, onko Sum-funktion palauttama arvo 11. Syitä,

miksi tämä testi voisi joskus epäonnistua on, että joku menee ja muuttaa `Sum()`-funktion logiikkaa. Ehkä joku toteaa, että `Sum()`-funktio voisikin vastaanottaa listan arvoja, ja funktio laskisi kaikki listassa olevat numerot yhteen. Luonnollisesti tällaisen muutoksen jälkeen täytyisi meidän kirjoittaa myös tämä testi uusiksi.

Entä miksi testaus on tärkeää? Traversy Median YouTube-kanavan videolla 'React Testing Crash Course' sovelluskehittäjä Mitchel vastaa mielestäni tähän kysymykseen hyvin. Testauksella haluamme varmistua, että sovelluksemme toimii oletetulla tavalla, vaikka siihen tehtäisiin muutoksia (Traverse Media 30.8.2021, 1:25 – 1:59 min). Lisäksi testien avulla sovelluksen toimivuuden testaaminen on huomattavasti nopeampaa, kuin sovelluksen toimintojen manuaalinen läpikäyminen (Traverse Media 30.8.2021, 1:25 – 1:59 min).

Itse olen useammankin kerran ohjelmoidessa jotain toimintoa, onnistunut rikkomaan samalla jonkin muun toiminnon tai ominaisuuden sovelluksessa. Koska kehittämässäni sovelluksessa on laaja ja monimutkainen käyttöliittymä, tuntuu sen testaaminen yksin manuaalisesti mahdottomalta tai vähintäänkin erittäin paljon aikaa vievältä. Odotan mielenkiinnolla sovellukseemme suunnitteilla olevaa automatisoitua frontend-puolen testausta ja sitä, että pääsen oppimaan aiheesta lisää.

Tiistai 21.2.2023

Sain muodostettua yhteyden laiteyksikköön, jota tarvitsin graafisen käyttöliittymän kehittämiseen ja jatkoin siis tämän tehtävän parissa. Lisäsin visuaalisuutta muun muassa asettamalla niille painikkeille oman värin, jotka ovat aktiivisia, eli käyttäjä on niitä klikannut. Koska painikkeet ovat vektorigrafiikkaa, niiden väriä ei voi muokata tutuilla "color" ja "background-color" ominaisuuksilla, vaan täytyy muokata niiden "fill" ja "stroke" ominaisuuksia. "Fill" ja "stroke" ominaisuuksille voi kuitenkin syöttää värejä CSS-kielestä tutuin keinoin, eli rgb-arvolla, hex-koodilla tai värin nimellä (MDN Web Docs 2023a).

Lisätessäni funktioita painikkeille, joilta logiikka vielä puuttui, huomasin, että laiteyksikön yksi sisäinen laite ei vastaanottanut lähettämiäni komentoja. Tai mikäli vastaanotti, se ei reagoinut niihin. Kokeilin lähettää komentoja laitteelle toisessa ympäristössä testatakseni, oliko syy minun koodisani. Laiteyksikön sisäinen laite ei vastaanottanut komentoja toisessa ympäristössäkään, joten päättelin, että ongelma oli minusta riippumaton. Tämä hidastaisi jälleen koodaamistani, sillä tarvitsin laitteelta rajapintakyselyihini vastauksia.

Koska laite ei reagoinut sille lähetettäviin komentoihin, siirryin työstämään SVG-kuvaa. Tarkoitus on, että laiteyksiköillä olisi 2 kuvaa: toinen vertikaalisille näytöille ja toinen horisontaalisille näytöille. Riippuen siis näyttöpäätteen koosta ja asennosta voimme tuoda näytölle sopivamman grafiikan näkyviin. Tyypillisesti laiteyksiköiden kylkeen asennetaan kosketusnäyttö pystyasentoon, jonka avulla

laiteyksikköä voi operoida paikan päällä. Laiteyksiköitä voi myös ohjata etänä, kuten minä suurimmaksi osaksi teen testilaitteiden kanssa. Ihmisillä on tyypillisesti tietokoneidensa näytöt vaakatasossa. Koska kehittämäni sovelluksen käyttöliittymää käytetään sekä pysty- että vaakanäyttöiltä, on tärkeää, että kehittämäni skaalautuvalla vektorigrafiikalla toteutettu graafinen käyttöliittymäosuus taipuu sekä pysty- että vaakanäytöille.

Keskiviikko 22.2.2023

Kun olin päässyt eteenpäin graafisen käyttöliittymän kanssa ja minulla alkoi olla jo jotain toimivaa käsissäni, ajattelin, että nyt olisi aika refaktoroida tekemääni koodia. Moni funktio tai koodilohko oli toteutettu nopeasti, jotta asioita oli päästy testaamaan, mutta koodi ei paikoin ollut kovinkaan järkevää tai siistää, vaikka olikin toimivaa. Aloin katsomaan, mitkä olivat selkeimpiä refaktoroinnin paikkoja.

Olin alun perin luonut erilliset React komponentit `<HorizontalView>` ja `<VerticalView>`, jotka saivat propseina tarvittavaa dataa, jolla renderöidään muun muassa erilaisia mittaustuloksia SVG-kuvaan graafisessa käyttöliittymässä. Propsi on Reactin termi funktion argumentille, eli tarkoittaa samaa asiaa, eli muuttujien siirtelyä funktiolta funktiolle. `<HorizontalView>` renderöi nimensä mukaisesti vaakatasoisen SVG-kuvan eli esimerkiksi 16:9 suhteessa olevalle näytölle ja `<VerticalView>` taas pystysuuntaisen SVG-kuvan eli esimerkiksi 9:16 suhteessa olevalle näytölle. Komponentit kuitenkin toimivat logiikaltaan lähes täysin samalla tavalla ja olin kopioinut koodia toisesta toiseen. Tässä minulla oli siis kaksi komponenttia, joilla on todella paljon identtistä koodia keskenään, eli tässä on varmasti uudelleen miettimisen paikka ja tarvetta refaktorointiin.

Voisin joko nostaa komponenttien logiikkaa ylemmälle tasolle, ja välittää logiikkafunktiot propseina `<HorizontalView>` ja `<VerticalView>` komponenteille. Tämä oli itseasiassa ensimmäinen suunnitelmani. Onneksi välittömästi alkaessani tehdä tätä ymmärsin, että tässä tapauksessa ei ole mitään järkeä olla kahta erillistä komponenttia alun alkaenkaan. Loin siis näiden kahden komponentin tilalle uuden komponentin `<GraphicalView>`, jolle annan propseina eri SVG-kuvan riippuen käyttäjän laitteen näytön leveydestä ja pituudesta. Kuvassa 9 näkyy vanhaa toteutusta, jossa renderöidään joko `<HorizontalView>` tai `<VerticalView>`. Kuvassa 10 näkyy refaktoroinnin jälkeistä toteutusta, jossa meillä on vain `<GraphicalView>` komponentti ajamassa saman asian.

Minulla oli myös melkoisen epämääräisen näköinen kooditoteutus, jossa määrittelin eri laitteiden mittausdataa omiin taulukkoihinsa ja passasin näitä taulukoita `<HorizontalView>` ja `<VerticalView>` komponenteille propseina. Päätin myös uudistaa tämän osuuden. Sain supistettua yli 50 koodiriviä pois luodessani järkevän oloiset toistorakenteet, jossa lisään mittausdataa taulukkoon, joka sisältää kaikkien tarvittavien laitteiden mittausdatan omina taulukoinaan. Lisäksi tämä vähensi uudelle

<GraphicalView> komponentille passattavien propsien määrää ja tältäkin osin selkeytti koodia omaan silmään. Verratessa kuvaa 9 ja 10 voi havaita, että kaikki kuvan 9 'deviceMeasurementsList'-propsit on korvattu kuvassa 10 yhdellä propsilla 'allDevicesMeasurements'.

```
return (
  <Main className="graphical-view" style={{flexDirection: "column", flex:
1}}>
    {graphics ?
      horizontalView === true ?
        <HorizontalView
          svg={graphics}
          sampleTempatureList={sampleTempatureList}
          device1MeasurementsList={device1MeasurementsList}
          device2MeasurementsList={device2MeasurementsList}
          device3MeasurementsList={device3MeasurementsList}
          device4MeasurementsList={device4MeasurementsList}
          device5MeasurementsList={device5MeasurementsList}
          devicesWithValveCommands={devicesWithValveCommands}
          stateList={uiCommandStatusList}
          valveStatusList={valveStatusList}
          systemId={systemId}
        />
      :
        <VerticalView
          svg={graphics}
          sampleTempatureList={sampleTempatureList}
          device1MeasurementsList={device1MeasurementsList}
          device2MeasurementsList={device2MeasurementsList}
          device3MeasurementsList={device3MeasurementsList}
          device4MeasurementsList={device4MeasurementsList}
          device5MeasurementsList={device5MeasurementsList}
          devicesWithValveCommands={devicesWithValveCommands}
          stateList={uiCommandStatusList}
          valveStatusList={valveStatusList}
          systemId={systemId}
        />
      :
        <>
          <p style={{color: "white", fontSize: "50px"}}>LOADING</p>
        </>
      }
    </Main>
  );
```

Kuva 9. <HorizontalView> ja <VerticalView> komponentit.


```

return (
  <Main className="graphical-view" style={{flexDirection: "column", flex:
1}}>
    {currentGraphicSVGString ?
      <GraphicalView
        svg={currentGraphicSVGString}
        allDevicesMeasurements={allDevicesMeasurements}
        devicesWithValveCommands={devicesWithValveCommands}
        stateList={uiCommandStatusList}
        valveStatusList={valveStatusList}
        systemId={systemId}
      />
      :
      <>
        <p style={{color: "white", fontSize: "50px"}}>LOADING</p>
      </>
    }
  </Main>
);

```

Kuva 10. <GraphicalView> komponentti.

Torstai 23.2.2023

Minua häiritsi edelleen sama tiistain ongelma laiteyksikön sisäisestä laitteesta, joka ei reagoinut sille lähetettyihin komentoihin. Katsoin selaimen kehittäjän työkalujen verkkopaneelista, että komennon lähetys näytti lähtevän oikein. Komento meni perille ja laite palautti oikeannäköisesti vastauksen. Aloin epäillä, että frontin koodissa tai sovelluksen tilanhallinnassa on nyt tapahtunut jokin muutos, joka on mahdollisesti rikkonut käyttöliittymän reagoinnin tämän laitteen komentojen osalta.

Koitin löytää syitä, miksi näin on käynyt ja miksi käyttöliittymä ei toiminut oikein, mutta en onnistunut toistaiseksi selvittämään tätä ongelmaa. Varmaan eniten käytetty keino front end -ongelmien vianjäljitykseen (eng. debugging) on konsolilokittaminen ja tämä tehdään siis koodillisesti `console.log()`-metodilla. Lokitin front endissä kulkevaa dataa ja kaikki näytti kulkevan funktioissa oikein ja oikeassa muodossa. En ymmärtänyt, miksi tieto ei tallentunut Reduxin tilanhallintaan oikein, vaikka näihin toimintoihin en ollut koskenut. Päätin ottaa toistaiseksi taas etäisyyttä tähän ongelmaan, ja keskittyä välillä johonkin muuhun. Olen omassa työskentelyssä huomannut, että välillä on hyvä ottaa pieni tauko ja siirtää ajatukset muualle. Kun myöhemmin palaat ongelman pariin uusin silmin, saatat yhtäkkiä huomata, missä ongelma on.

Löysin graafisen käyttöliittymän komponenttien koodista lisää refaktoroitavaa ja jatkoin siitä. Tällä erää selkeytin ja tiivistin graafisen käyttöliittymän painikkeiden koodia. Olin alun perin kirjoittanut jokaiselle painikkeelle oman funktionsa, mutta näissä funktioissa oli paljon samankaltaisuutta. Uutta funktiota varten täytyi vain poistaa kovakoodatut arvot ja vaihtaa ne muuttujiksi. Lisäksi uuden funktion täytyi pystyä käsittelemään muutamat erikoistapaukset. Tähän JavaScriptin "switch"-

lauseke tuntui sopivalta lähestymistavalta. Asetin "switch"-lausekkeeseen poikkeustapaukset ja oletustapaukseksi ohjelmoin painikkeen normaalin toiminnan.

Perjantai 24.2.2023

Opiskelin lisää Jest-testausta ja tällä kertaa katsoin PedroTechin 16.6.2022 julkaiseman videon "Testing in React Tutorial- Jest and React Testing Library." Olen katsonut Pedrolta muitakin videoita, joten luotin tämän olevan kelvollinen opetusvideo. Tapa, jolla opettelen, on koodata kaikki videoissa olevat tapaukset myös itse ja välillä tehdä jotain pieniä koodillisia muutoksia, jotta homma ei menisi vain kopioimiseksi. Koodaamisen opettelussa olen huomannut, että pelkkä videoiden katselu ei vie sinua paljoa eteenpäin, vaan täytyy myös saada tuntumaa itse koodin kirjoittamiseen.

Video oli melko lyhyt ja käsitteli osittain jo samoja asioita, kuin mitä olin ehtinyt aiemmin tällä viikolla opiskella, mutta toimi hyvänä kertauksena. Lisäksi videolla puhuttiin Jestin käyttämisestä nimenaan React-ympäristössä, kun aikaisemmin olin tutustunut Jestiin tavallisen JavaScriptin kanssa.

Aloin etsiä lisää opetusmateriaalia ja yksi hyvin suosittu YouTube-kanava koodauksen opetteluun on nimeltään freeCodeCamp.org. Kanavalla on yli 7 miljoonaa tilaajaa ja heiltä löytyy erittäin laajasti tutoriaalia erilaisten ohjelmistokehittäjien tekemänä. Videot ovat välillä jopa 16 tuntisia ja ovat erittäin kattavia ja opettavaisia oman kokemukseni mukaan.

Aloin katsoa 2 tunnin videota nimeltä "React Testing Course for Beginners – Code and Test 3 Apps". Videolla ohjelmistokehittäjä Nikolay Advolodkin käy aluksi läpi React-applikaation luomisen ja "CI-pipelin" alustamisen tämän applikaation tuotantoversion ajamiseen. "CI/CD-pipeline" on käytännössä ohje tai vaiheittainen lista, joka käydään läpi sovelluksen asentamisen yhteydessä. Kun sovellus asennetaan, käy tietokonekone tämän listan läpi vaihe vaiheelta suorittaen automaattisesti jokaisen kohdan. Tällä pyritään välttämään inhimillisiä erheitä, joita ihminen saattaisi tehdä asennuksen yhteydessä. (Red Hat 2022.)

Pipelinet ovat erittäin hyödyllisiä, sillä niiden avulla sovelluskehitysprosessia voidaan nopeuttaa ja koodin laadusta voidaan olla varmempia (Red Hat 2022.) Jos sovelluksen asentamisen yhteydessä CI-pipeline ei pysty suorittamaan jotain kohtaa, antaa se asiasta virhekoodin, josta ihminen voi päätellä ongelman laadun (Advolodkin 23.5.2022, 12:15 – 12:41).

Entä miten pipelinet liittyvät testaukseen? CI-pipelinessä voidaan tietokonetta käskä suorittamaan asennuksen yhteydessä myös kirjoittamamme testit. Jos jokin testeistä ei mene läpi, saamme siitä ilmoituksen ja tiedämme, että jokin tekemämme koodimuutos on rikkonut ohjelmassa jotain. Koska

tiedämme tarkkaan, mitkä testit eivät menneet läpi, on meillä myös ymmärrys siitä, mikä sovelluksessa ei toimi oikein. Voimme tämän jälkeen tehdä tarvittavat korjaukset ja koittaa suorittaa tuotantoon ajon uudestaan. (Red Hat 2022).

Viikkoanalyysi 2

Sain tällä viikolla opiskeltua testaukseen liittyen sekä edettyä graafisen käyttöliittymän tekemisessä, joten pidän viikkoa onnistuneena. Haluaisin sisällyttää viikkoihini myös jatkossa enemmän opiskelua, mikäli sopivia hetkiä opiskelulle löytyy. Ohjelmistokehityksessä opittavaa tuntuu olevan loputtomasti ja ainakin itse koen opiskelun antavan minulle jatkuvasti parempia eväitä suoriutua työtehtävistäni. Yksi ongelma on löytää netin loputtomasta tarjonnasta se paras opintomateriaali, sillä usein jotkin videot tuntuvat vain osittain hyödyllisiltä. Toisinaan tulee taas katsottua opetusvideota ja huomattua, että ei tämä käsittelekään jotain aihealuetta sellaisella tavalla, kuin olin toivonut.

Omaa työskentelyä tällä viikolla hidasti laiteyksikön sisäinen laite, jolle syötetyt komennot eivät heijastuneet käyttöliittymään, kuten kuuluisi. Tiistaina olin jostain syystä ajatellut, että laitteen rajapinnassa saattaisi olla häikkää, mutta en ollut syystä tai toisesta tarkistanut asiaa kehittäjätyökalujen verkkopaneelisti. Torstaina, kun katsoin asiaa uudestaan, näytti rajapintakysely lähtevän oikein ja saavan vastaukseksi oikean näköistä dataa. Tämä hämmensi minua ja sai minut koittamaan ratkaista ongelmaa uudestaan.

Tiiminvetäjä on ollut tämän viikon poissa, joka on tarkoittanut sitä, että en ole voinut kysyä häneltä apua tämän kaltaisissa ongelmissa. Pystyn itsenäisesti ratkomaan front end -puolen koodiongelmia, mutta kun ongelmiin liitetään kehittämämme digitaalinen reunalaskentaa hyödyntävä tuote kokonaisuutena sekä siihen liittyvät laitepuolen asiat, ei minulla välttämättä ole aina riittävää ymmärrystä selvittää, mistä jotkin ongelmat johtuvat. Joskus ongelmat liittyvät niin sanotusti raudan ja back endin väliseen kommunikaatioon. Eli vaikka front end ja back end toimisivat keskenään, ei back end ja fyysinen teollisuudenlaite kommunikoi välttämättä oikein keskenään. Tällöin ollaan jo kaukana minun vahvuusalueistani.

Olen kuitenkin pyrkinyt siihen, että käytän työaikaani tehokkaasti. En esimerkiksi halunnut käyttää liikaa aikaa tiistain laiteongelman parissa, vaan mieluummin opiskelin jotain aihetta, jonka tiedän olevan minulle ja yritykselleni hyödyksi. Tämän voisi ajatella olevan itsenäistä työtettä, joka on mielestäni hyvä ominaisuus työntekijässä. Mielestäni on tärkeää, että ei lamaannu esimerkiksi tehtävien haastavuudesta, - määrästä tai - epämääräisyydestä, vaan koittaa löytää keinoja olla hyö-

dyllinen ja kehittää itseään myös silloin, kun töissä tulee seiniä vastaan. Tietysti tässä kohtaa joustavuutta tuo se, että mikään aikamääre ei ole tuomassa minulle painetta suorittaa jokin kehitettävä ominaisuus loppuun.

3.3 Seurantaviikko 3

Maanantai 27.2.2023

Pidimme palaverin koko tuotekehitystiimin kesken, eli palaveriin osallistui sovelluskehittäjien lisäksi myös laitesuunnittelijat. Kävimme läpi uuden tulevan laiteyksikön toimintaa. Kävimme läpi esimerkiksi, mitä laitteita tähän tulevaan laiteyksikköön kuuluu ja mitä asioita tulisi ottaa huomioon laitekonaisuutta rakentaessa. Front end -puolen kehittäjänä tässä palaverissa ei vielä ollut minua merkittävästi koskevia asioita, mutta koen, että minun on hyvä olla perillä tulevista tuotteista ja palveluista ja niiden mahdollisista vaikutuksista myös minun tekemiseeni.

Koska tiiminvetäjä oli palannut lomilta, pääsin kysymään häneltä minua viime viikolla vaivanneesta ongelmasta erään laiteyksikön sisäisen laitteen komentoihin liittyen. Ongelma oli toisaalla, kuin front end -puolen koodissa ja saimme syyt sille selvitettyä. Tämän myötä pystyn jatkamaan graafisen käyttöliittymän kehitystä.

Pidimme myös Scrumin mukaisen kehitysjaksosuunnittelun (eng. sprint planning). Kävimme siis läpi, mitä kukin oli tekemässä ja mitä kukin tulisi tekemään seuraavan kehitysjakson aikana. Laitoimme myös Azure-palvelussa olevat tehtävätiketit ajan tasalle niiltä osin, kuin niissä oli päivitettävää.

Ehdin tämän päivän aikana koodata hyvin minimaalisesti, sillä tämä päivä sisälsi paljon enemmän kollegoiden kanssa keskustelua, ajatusten vaihtoa, toisten auttamista, tiimihengen kasvattamista ja sparraamista. Tämän kaltaiset aktiviteetit ovat mielestäni tärkeitä työtehtävistä riippumatta, sillä ne parantavat työilmapiiriä ja -hyvinvointia ainakin oman kokemukseni mukaan.

Tiistai 28.2.2023

Jatkoin graafisen käyttöliittymän kehitystä. Liitin lisää tietoa Reduxin tilanhallintaan laiteyksikön sisäisen laitteen tilasta, jota kutsuttakoot nimellä SL1 (sisäinen laite 1). Eli kun käyttäjä lähettää käyttöliittymästä komennon SL1:lle ja komento onnistuu, haluamme tallentaa tiedon päivittyneestä tilasta Reduxiin. Jouduin tekemään muutoksia toisen henkilön kirjoittamaan koodilogiikkaan tähän liittyen, sillä aikaisempi koodi ei syystä tai toisesta pitänyt Reduxissa tallessa tilatietoja SL1:n ko-

mentojen tilasta. Koska en ole aivan varma, miksi koodi oli alun perin kirjoitettu niin kuin se oli, joudun pitämään tarkemmin silmällä, onko tekemilläni muutoksilla jotain vaikutuksia, joita en välttämättä ole huomannut tai ymmärtänyt.

Lisäksi pohdin mahdollisuutta muutenkin refaktoroida tuota vanhempaa logiikkaa kokonaan tai osittain, sillä minusta tuntuu, että se on osiltaan hankalasti toteutettu. Toisaalta koen, että refaktorointi tuon logiikan osalta veisi minulta ehkä turhan paljon aikaa ja minulle ei ole täysin selvää, miten kirjoittaisin sen paremmin.

Huomasin ongelman käyttöliittymässä tekemieni muutosten jälkeen. Laiteyksiköissä on manuaalinen sekä automaattinen ohjaus. Manuaalisen ohjauksen ollessa päällä, käyttäjä voi lähettää laiteyksikön sisäisille laitteille komentoja, eli käyttäjä voi esimerkiksi antaa komentoja SL1:lle. Olin värjännyt käyttöliittymästä tämän SL1:n komentopainikkeet oransseiksi, mikäli komento oli aktiivinen. Asettaessani laiteyksikön manuaaliohjauksesta takaisin automaattiohjaukselle, huomasin, että SL1 todellinen tila palautui oletusasetuksiin back endin mukaan, mutta käyttöliittymän tilanhallinnassa oli edelleen SL1:n vanha tilatieto. Laite oli siis oikeasti tilassa A, mutta käyttöliittymä antaa ymmärtää sen olevan tilassa B.

Minun täytyy tähän liittyen konsultoida tiiminvetäjää, että onko tuo oletusasetuksiin palaaminen ensinnäkään oletustoiminto tai haluttu ominaisuus. Todennäköisesti on, eli joudun tekemään jonkin näköisen ratkaisun tuohon käyttöliittymän ongelmaan. Ehkä mentäessä manuaaliohjauksesta automaattiohjaukseen, voin vain pyyhkiä SL1:n Redux-tilatiedot. Voisin myös mahdollisesti hakea rajapinnasta tiedon SL1 tilasta, ja päivittää sen sitten Reduxiin sitä kautta, mutta en ole varma, onko meillä sopivaa rajapintaa tätä tarkoitusta varten. Kolmas vaihtoehto voisi olla luoda funktio, joka asettaa SL1:lle oletusarvot myös front endin -puolelle Reduxiin käyttäjän siirtyessä manuaaliohjauksesta automaattiohjaukseen.

Keskiviikko 1.3.2023

Keskustelin päivittäisessä Scrum-palaverissa tiiminvetäjän kanssa tiistain havainnoistani ja oletukseni olivat osuneet oikeaan, eli automaattiohjaus palauttaa laiteyksikön oletusasetuksiin ja tämä on tarkoituksen mukaista. Lisäksi graafisen käyttöliittymän painikkeita ei pitäisi voida käyttää, mikäli laiteyksikkö on automaattiohjauksella. Kokouksen jälkeen päällimmäisenä mielessäni oli tuo painikkeiden käytöstä poisto automaattiohjauksen aikana, joten lähdin toteuttamaan sitä.

Sovelluksessa on jo olemassa toiminnot automaatti- ja manuaaliohjaukseen liittyen ja Reduxin tilanhallinnassa on tallessa tieto siitä, onko laiteyksikkö automaatti- vai manuaaliohjauksella. Täten minun tarvitsi vain hakea tuo tieto komponenttiini käyttäen React-Reduxin `useSelector()`-funktiota.

UseSelector()-funktio on siis React-Redux kirjaston "hook" eli funktio, jonka on luotu datan hakemiseen Reduxin tilahallinnasta React-pohjaisessa sovelluksessa (React-Redux 2022). UseSelector()-funktioilla haettu arvo päivittyy komponenttiisi automaattisesti, mikäli arvo muuttuu Reduxissa (React-Redux 2022). UseSelector()-funktion käyttö voisi yksinkertaisesti näyttää tältä:

```
const serviceModeStatus = useSelector((state) => state.serviceModeStatus)
```

Loin tyyli luokan nimeltä "disabled" ja tein tälle CSS-koodia, jossa värjään painikkeita harmahtaviksi, jotta ne näyttävät olevan niin sanotusti pois käytöstä. Tämän jälkeen lisäsin <GraphicalView> komponenttiini lausekkeen, joka lisää "disabled"-tyyli luokan SVG-elementtiin, mikäli laite ei ole manuaaliohjauksessa. Alla näkyy if-lauseke, jossa tyyli luokka lisätään, sekä pieni pala tyylikoodia esimerkkinä.

```
if (serviceModeStatus !== "ManualActive") {
  svgRoot.setAttribute("class", "disabled")
}
```

```
svg.disabled .edgeunit_graphics #button-device1-span2 rect {
  fill: rgb(88, 84, 81);
  cursor: not-allowed;
}
```

Lisätessä logiikkaa <GraphicalView> komponenttiini, yksi graafisen käyttöliittymän oleellinen useEffect()-hook alkoi vaikuttaa hieman kaoottiselta. Osa tästä useEffect()-hookista on nähtävissä kuvassa 11. Kuvasta 11 voimme havaita, että käsittelen if-else-ehtolausekkeilla erilaisia tilanteita, jotta saamme graafisen käyttöliittymän SVG-elementtiin liitettyä haluamiamme tyyli luokkia.

Kuvassa 11 loppupuolella esiintyvä "listOfActiveClasses"-muuttuja pitää sisällään tyyli luokkanimiä riippuen käyttäjän lähettämistä komennoista laitteelle. Eli kun käyttäjä painaa painiketta "air", lisäämme tyyli luokan "air" SVG-elementtiin. Ongelmana on kuitenkin, että käyttämämme tyyli luokka "default" ei ole mikään laitekomento, eli sille ei ole painiketta, jota käyttäjä painaisi. Tämä tarkoittaa sitä, että joudumme erikseen erittelemään tilanteet, joissa haluamme käyttää "default"-tyyli luokkaa.

Päätin koittaa selkeyttää koodia jonkin verran, ennen kuin tilanne muuttuu pahemmaksi. Kuvassa 12 on useEffect()-hookin päivitetty versio, joka pyrkii samaan aikaiseksi saman, kuin kuvan 11 koodi, mutta vähemmällä rivimäärällä. Jouduin tätä varten tekemään tosin joitakin rivejä lisää CSS-koodia, mutta otan mieluummin 20 riviä enemmän CSS-koodia, jos sillä saadaan pois 20 riviä JavaScript if-else-ehtolauseita.

```

useEffect(() => {
  if (!svgRoot) {
    return;
  }
  if (serviceModeStatus !== "ManualActive") {
    svgRoot.setAttribute("class", "default left disabled");
    return;
  }

  // when valveStatusList is empty, we are in default state, SET class to
"default left"
  if (valveStatusList.length === 0) {
    svgRoot.setAttribute("class", "default left");
    return;
  }

  // when systemprobe class is the only one in the the list, SET class to
default
  else if ((valveStatusList.length === 1) &&
(valveStatusList[0].graphicsClass === "systemprobe")) {
    svgRoot.setAttribute("class", "default left");
    return;
  }

  // when dcsstop class is the only one in the the list, SET class to
default
  else if ((valveStatusList.length === 1) &&
(valveStatusList[0].graphicsClass === "dcsstop")) {
    svgRoot.setAttribute("class", "default left");
    return;
  }

  // otherwise, we use the valveStatusList to determine what classnames
to give svgRoot
  else if (valveStatusList.length > 0) {
    svgRoot.classList.remove(...svgRoot.classList) // remove all old
classes

    let listOfActiveClasses = [];
    valveStatusList.map((singleValveStatus, index) => {
      listOfActiveClasses.push(singleValveStatus.graphicsClass)
    })
    //if none of the device1 commands are not pressed and for some reason
default class is not set. Add it to the class list.
    if (listOfActiveClasses.includes("air") === false &&
        listOfActiveClasses.includes("span1") === false &&
        listOfActiveClasses.includes("span2") === false &&
        listOfActiveClasses.includes("default") === false
        ) {
      listOfActiveClasses.push("default")
    }
    svgRoot.classList.add(...listOfActiveClasses) // add all new classes
  }
}

```

Kuva 11. Yksi graafisen käyttöliittymä -komponentin useEffect()-hookeista.

```

// This handles SVG element's class-attributes. We add and remove classes
according to user inputs.
// valveStatusList contains information on active classes that should be added
// You can style these classes with CSS as usual.
// For example "svg.bypass #line_device1Left" will add styles to
line_device1Left when bypass class is active
useEffect(() => {

  if (!svgRoot) {
    return;
  }

  // When not in Manual control, use "default left disabled"
  if (serviceModeStatus !== "ManualActive") {
    svgRoot.setAttribute("class", "default left disabled");
    return;
  }

  // When valveStatusList is empty, we are in default state, SET class to
"default left" (left is default state for device1)
  else if (valveStatusList.length === 0) {
    svgRoot.setAttribute("class", "default left");
    return;
  }
  else {
    // Get list of active classes
    let listOfActiveClasses = [];
    valveStatusList.map((singleValveStatus, index) => {
      listOfActiveClasses.push(singleValveStatus.graphicsClass)
    })

    // If listOfActiveClasses don't contain "air", "span1" or "span2", add
default class to the list.
    if (
      (listOfActiveClasses.includes("air") === false) &&
      (listOfActiveClasses.includes("span1") === false) &&
      (listOfActiveClasses.includes("span2") === false)
    ) {
      listOfActiveClasses.push("default")
    }

    // Remove all old classes
    svgRoot.classList.remove(...svgRoot.classList)
    // Add all new classes
    svgRoot.classList.add(...listOfActiveClasses)
    return;
  }

}, [ svgRoot, valveStatusList, serviceModeStatus ]);

```

Kuva 12. Refaktoroitu versio kuvan 11 koodista.

Torstai 2.3.2023

Keskiviikon aikana huomasin <GraphicalView> komponentissa ja siihen liittyvissä komponenteissa ja funktioissa lisääkin paranneltavaa. Koska olen vasta koittamassa saada tätä kokonaisuutta toimimaan ja tämä on ollut pitkä prosessi, on monet alussa tekemäni funktiot mielestäni ainakin näin jälkikäteen katsottuna huonosti toteutettu. Jatkoin keskiviikon teemaa ja parantelin aikaisemmin luomiani funktioita selkeämmiksi ja paremmin tarkoitustaan palveleviksi.

Aikaisemmat painikkeiden käsittelijät näyttivät seuraavalta.

```
// Create click handlers
const onClickAir = () => {graphicalUiCommandButtonPressed("air", "Device-
name1")};
const onClickSpan1 = () => {graphicalUiCommandButtonPressed("span1", "Devic-
ename2")};
```

Kuvassa 13 näkyy graphicalUiCommandButtonPressed()-funktion logiikka. Tässä logiikassa on myös joskus ollut mukana tyyli luokan lisääminen SVG-elementtiin. Funktiolle syötettävää ensimmäistä parametria käytettiin alun perin sekä komennon lähettämiseen back endille, että tyyli luokan nimenä SVG-elementille. Alkuperäinen ongelmani oli ollut, että tyyli luokka ei voi sisältää välilyöntejä tai erikoismerkkejä, kun taas back endille lähetettävä komento saattoi sisältää välilyöntejä. Esimerkiksi "Span 1"-niminen tyyli luokka antaisi virheilmoituksen, sillä CSS-tyyli luokan nimi ei saa sisältää välilyöntejä. Graafisen käyttöliittymän kehittyessä olen jossain kohtaa siirtänyt tyyli luokkien lisäämisen logiikan pois tästä funktiosta, joten graphicalUiCommandButtonPressed()-funktion voisi päivittää järkevämmäksi.

Koska graphicalUiCommandButtonPressed()-funktiolle syötettävää ensimmäistä parametria käytetään enää vain yhteen tarkoitukseen, voimme antaa ensimmäisen parametrin suoraan oikeassa kirjoitusasussa. Uudet painikkeiden käsittelijät näyttävät seuraavalta.

```
// Create click handlers
const onClickAir = () => {graphicalUiCommandButtonPressed("Air", "Devi-
cename1")};
const onClickSpan1 = () => {graphicalUiCommandButtonPressed("Span 1", "De-
vicename2")};
```

Kuvassa 14 voimme nähdä, että tämän muutoksen myötä pystymme ehostamaan graphicalUiCommandButtonPressed()-funktion logiikkaa paljon siistimmäksi.

```
// When a button is pressed in SVG image
const graphicalUiCommandButtonPressed = (buttonCommand, deviceName) => {

    let device = devicesWithValveCommands.find(device => device.name ===
deviceName)
    let capitalizeCommand = buttonCommand.charAt(0).toUpperCase() +
buttonCommand.slice(1);
    let valveCommand;

    switch (buttonCommand) {
        // cases are special cases which the default can't handle
        case "toProbe": valveCommand = device.commands.find(cmd => cmd.name
=== "System/Probe"); break;
        case "span1": valveCommand = device.commands.find(cmd => cmd.name ===
"Span 1"); break;
        case "span2": valveCommand = device.commands.find(cmd => cmd.name ===
"Span 2"); break;
        default: valveCommand = device.commands.find(cmd => cmd.name ===
capitalizeCommand)
    }
    onHandleCommand(valveCommand, device.hwid)
}
```

Kuva 13. GraphicalUiCommandButtonPressed()-funktion vanha logiikka.

```
// When a button is pressed in SVG image
const graphicalUiCommandButtonPressed = (buttonCommand, deviceName) => {

    // Find the device that matches the device the button is bound to
    let device = devicesWithValveCommands.find(device => device.name ===
deviceName)

    // Find the corresponding command from the device's commandlist
    let valveCommandObject = device.commands.find(cmd => cmd.name ===
buttonCommand)

    onHandleCommand(valveCommandObject, device.hwid)
}
```

Kuva 14. GraphicalUiCommandButtonPressed()-funktion päivitetty logiikka.

Perjantai 3.3.2023

Keskustelin päivittäisessä scrum-palaverissa lisää tiiminvetäjän kanssa siitä, miten hoidamme käyttöliittymän tilan päivittämisen laitekomentojen osalta tilanteessa, jossa käyttäjä aktivoi automaatti- tai manuaaliohjauksen. Ongelma on siis, että laiteyksikön sisäiset laitteet voisivat olla tilassa A, kun taas käyttöliittymä väittää niiden olevan tilassa B. Tulimme tulokseen, että tyhjentämme komentojen tilatiedot front endin tilahallinnasta käyttäjän siirtyessä automaatti- ja manuaaliohjauksen välillä. Mikäli tulevaisuudessa tämä ratkaisu ei ole riittävän hyvä, voimme luoda monimutkaisemman logiikan, joka vaatisi muutoksia sekä front endin, että back endin puolelle.

Minun kannaltani tämä ratkaisu oli helppo toteuttaa. Minun tarvitsi vain käytännössä lisätä yksi koodirivi Reduxin logiikkaan. Kuvassa 15 on tuo rivi näkyvillä, eli asetan siinä "state.overview.commandsStatusListin" tyhjäksi objektiksi. Olen piilottanut sekä sensuroinut kuvasta kaiken tähän muutokseen liittymättömän logiikan.

Kuvassa 15 on siis Redux-toolkit kirjaston createSlice()-funktio, joka sisältää logiikkaa, jolla on tarkoitus hallita Reduxissa tallessa olevaa tilaobjektia (redux-toolkit 2022). Tässä tapauksessa tilaobjektin nimi on "ui", sillä on oletustila "uiDefaultState", sillä on reducer()-funktioita ja myös "extra-Reducer"-logiikkaa. Reducerit hallitsevat vain oman slicen toimintoja ja tilaa, ja extraReducerit voivat taas vaikuttaa muihinkin sovelluksessa oleviin sliceihin (Redux-toolkit 2022). Kuvassa olevan extraReducerin sisällä oleva toiminto aktivoituu, kun "somethingAPICall"-rajapintakutsu saa vastauksen 200 OK.

```
const uiSlice = createSlice({
  name: "ui",
  initialState: uiDefaultState,
  reducers: { . . . many functions here . . . },
  extraReducers: (builder) => {
    builder
      .addCase(somethingAPICall.fulfilled, (state, action) => {
        if (commandType === "Auto" || commandType === "Manual") {
          // Reset commandStatusList every time we enter or leave manual
mode. This is because devices SHOULD enter default state when in automatic
mode.
          // Meaning, that the UI should be in default state also in terms
of commands
          state.overview.commandsStatusList = {};
        }
      })
  },
});
```

Kuva 15. Reduxin createSlice()-funktio sisältäen extraReducerin commandStatusList-tilamuuttujan tyhjentämiseen.

Muutoksen jälkeen tein hieman hienosäätöä muuallakin koodissa ja sain käyttöliittymän tilan palautumaan oletusasetuksiin käyttäjän siirtyessä automaatti- ja manuaaliohjauksen välillä. Tämä ei vienyt kauaa, joten siirryin seuraavien ongelmien kimppuun.

Päivitin myös <GraphicalView> komponentin responsiivisuus-logiikkaa. Meillä on siis kaksi erilaista näkymää: pysty- ja vaakanäkymä. Olin aikaisemmin katsonut selaimen ikkunan kokoa hyödyntämällä web-API:n window-ominaisuuden "innerHeight"-arvoa ja tämän perusteella renderöinyt joko vaaka- tai pystymallisen SVG-kuvan käyttöliittymään. Window.innerHeight palauttaa siis selaimen korkeuden arvon pikseleinä pois lukien selaimen yläpalkin (MDN Web Docs 2023b). Koodin logiikka oli tämän suuntainen:

```
if(window.innerHeight > window.innerWidth) {
    setCurrentGraphicSVGString(objectWithSvgStrings.verticalGraphic);
} else {
    setCurrentGraphicSVGString(objectWithSvgStrings.horizontalGraphic)
}
```

Päivitin koodia niin, että se hyödyntääkin `HTMLElement.offsetHeight` arvoa. Eli koodi näyttää nyt seuraavanlaiselta:

```
if((graphicalView.offsetHeight > graphicalView.offsetWidth) && (graphicalView.offsetWidth < 910)) {
    setCurrentGraphicSVGString(objectWithSvgStrings.verticalGraphic);
} else {
    setCurrentGraphicSVGString(objectWithSvgStrings.horizontalGraphic)
}
```

Eli nyt katsomme `<GraphicalView>` komponentin kokoa, emmekä selaimen ikkunan kokoa. Tähän en myöskään käyttänyt liikaa aikaa, sillä SVG-kuvat saattavat vielä muuttua ulkonäöltään, pituudeltaan tai korkeudeltaan, joten minun ei kannata hioa tätä logiikkaa enempää tässä vaiheessa.

Loppuajan testailin graafisen käyttöliittymän toimivuutta ja etsin potentiaalisia ongelmakohtia, sekä aloin pohtia keinoja luoda erilaisia 'wau'-effektejä tyyliittelyn ja animaation keinoin. Ajattelin, että ehkä painikkeissa olevat venttiilikuvat voisivat pyörähtää niitä painaessa ja tämä kuvaisi kivasti sitä, että käyttäjä avaa venttiilin. Loin CSS-koodilla animaation hyödyntäen transform-ominaisuutta ja pyörittäen kuvaa y-akselinsa ympäri komennolla `rotateY(180deg)`.

Animaatio ei kuitenkaan toiminut, sillä venttiilinkuva lähti lentämään villisti kauas omalta paikaltaan, eikä pyörähtänyt itsensä ympäri, niin kuin oli tarkoitus. Löysin apua ongelmaan "developedbyed"-nimiseltä YouTube-kanavalta, jossa videon henkilöllä oli itse asiassa täysin identtinen ongelma, kuin minulla (developedbyed 23.11.2019, 11:30 – 11:43). Lisäsin animaatioon rivit "transform-origin: center" sekä "transform-box: fill-box" ja ongelma korjaantui (developedbyed 23.11.2019, 11:50 – 12:12).

Viikkoanalyysi 3

Tiistain 28.3.2023 raportin ensimmäisessä kappaleessa, kerroin, että en ollut varma, olisiko tekemilläni koodimuutoksilla jotain sellaisia vaikutuksia, joita en ollut osannut ottaa huomioon. Tämän kaltaisiin tilanteisiin automatisoitu testaus voisi tuoda apua. Esimerkiksi funktionaalinen testaus, jonka tarkoitus on siis testata, että sovelluksen toiminnot toimivat, voisi auttaa minua katsomaan nopeammin, rikkoivatko tekemäni koodimuutokset sovelluksessa jonkin toiminnon (TestingXperts 2022).

Tein myös havainnon, että saatan unohtaa välillä jonkin asian tai yksityiskohdan, josta olemme tiimin kanssa keskustelleet Scrum-palaverissa. Unohtaminen on inhimillistä, mutta jos ei ole varma

siitä, että muistaa olennaiset asiat, on syytä kirjoittaa nuo asiat itselleen muistiin. Esimerkiksi keskeisen viikon 1.3.2023 Scrum-palaverissa keskustelin siitä, että laiteyksikkö palautuu oletustilaan mentäessä automaattiohjaukselle. Koska keskustelin myös muista asioista, unohtui minulta tuo oletustila-asia, enkä sen vuoksi tullut koodanneeksi siihen ratkaisua, kuin vasta perjantaina.

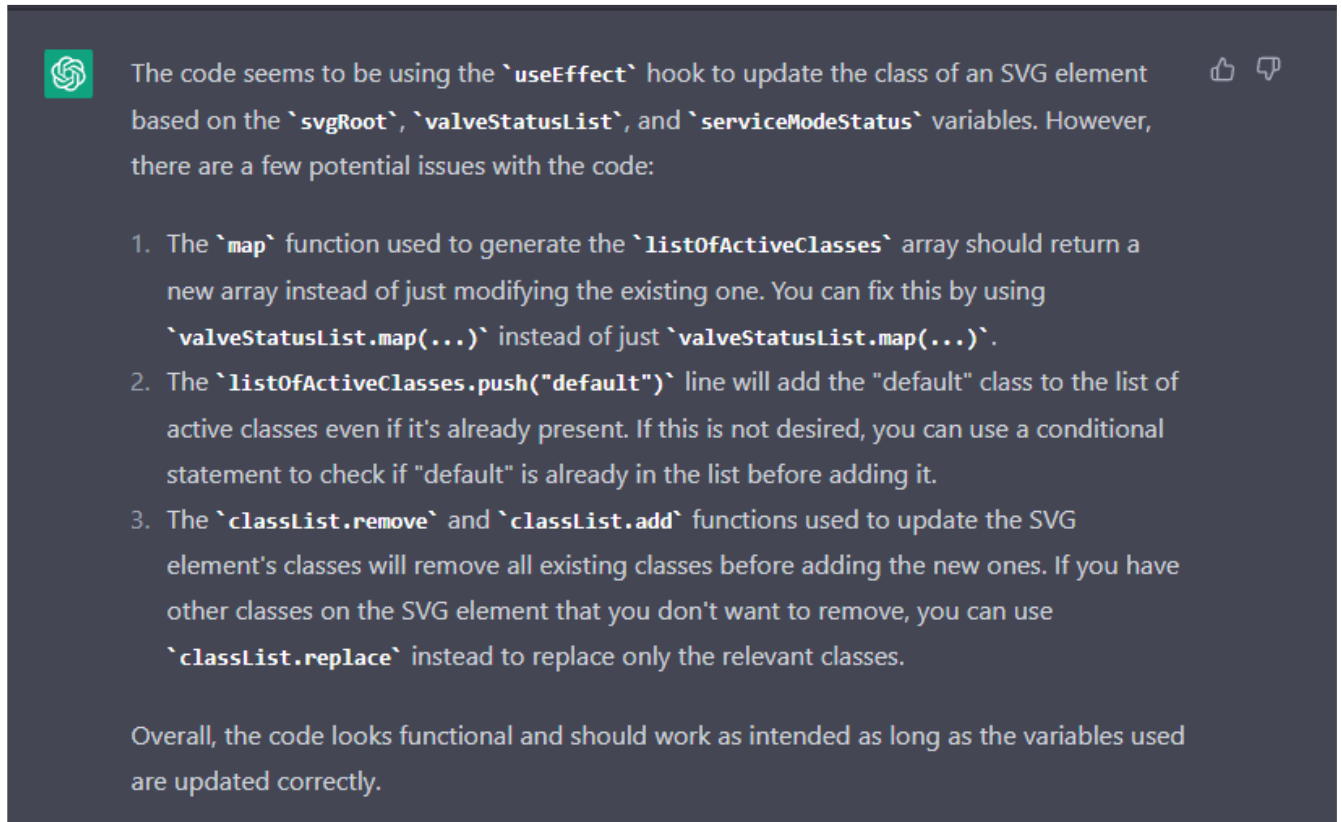
Minun tulisi laittaa enemmän itselleni ylös muistiin asioita esimerkiksi palavereissa esiin nousevista kehitysehdotuksista sekä varsinkin minulla työn alla olevista asioista. Olen pannut merkille myös, että aika ajoin joudun kysymään tiiminvetäjältä saman asian uudestaan, koska en enää muista hänen antamaa vastausta ja en ole kirjannut asiaa ylös. Asioiden ylös kirjaaminen ja esimerkiksi päivittäisten tehtävälistojen ylläpitäminen kasvattaa tehokkuutta, estää lamaantumasta työmäärän ollessa suuri ja toimii turvana, jos unohdat asioita (Mind Tools Content Team; Bhandari 2015). Otan työn alle olla aktiivisempi asioiden ylös kirjaamisessa.

Koodillisesti olen tyytyväinen viikon aikana kirjoittamiini päivityksiin ja optimointeihin. Kysyin kuitenkin ChatGPT-tekoälyn mielipidettä kuvan 12 koodiin. Kuvassa 16 on ChatGPT-tekoälyn vastaus. ChatGPT-tekoäly ei löytänyt suoria virheitä, mutta löysi mahdollisesti parannettavia seikkoja. Ensimmäinen ChatGPT-tekoälyn ehdotus on olla muuntamatta olemassa olevaa taulukossa, vaan luoda uusi taulukko vanhan muuntamisen sijaan. Toinen ehdotus on tehdä tarkistus, onko "default"-luokka olemassa taulukossa, ennen kuin lisäämme sen sinne. Kolmannessa kohdassa ChatGPT-tekoäly tuo esiin mahdollisuuden olla käyttämättä `classList.remove` -metodia ja käyttää sen sijaan `classList.replace` -metodia, mikäli haluaisimme vain korvata joitakin luokkia.

Ensimmäinen ChatGPT tekoälyn ehdotus taulukoiden muuntamisen tai "mutatoimisen" välttämiseksi, niin kuin ohjelmointikielessä puhutaan, on hyvä perussääntö, jota noudattaa. Syy, miksi et halua tyypillisesti muunnella olemassa olevia taulukoita on, että jokin toinenkin funktio saattaa käyttää niitä. Eli jos sinulla on jokin laajemmassa käytössä oleva taulukko ja haluat jossain tilanteessa käyttää sen sisältämää dataa sitä hieman muunnellen, luot luonnollisesti uuden taulukon vanhan taulukon arvojen muuttamisen sijaan. Kuvan 12 tapauksessa näin ei kuitenkaan ole, vaan `listOfActiveClasses`-taulukkoa käytetään turvallisesti ainoastaan oman funktion sisällä, eikä sillä ole mitään tarkoitusta kuvassa 12 näkyvän `useEffect()`-hookin ulkopuolella.

ChatGPT-tekoälyn kohta 2 on itse asiassa hyvä huomio ja minulla on alun perin ollut koodissa tarkistus, sisältääkö taulukko arvoa "default" ja lisää vasta sitten taulukkoon arvon "default", mikäli sitä ei taulukossa ole. Kuvassa 11 tämä tarkistus on tehtynä, mutta olen jostain syystä unohtanut sen kuvan 12 koodissa, joka oli siis päivitetty versio.

ChatGPT-tekoälyn kohta 3 on pikemminkin vain huomio, että on olemassa myös `classList.replace` -metodi, mikäli sovelluskehittäjä eli tässä tapauksessa minä en ole tällaisen metodin käyttöä pohtinut. En kuitenkaan halua käyttää `replace`-metodia, vaan nimenomaan poistaa kaikki vanhat arvot `classList.remove`-metodilla ja sen jälkeen lisätä uudet arvot `classList.add`-metodilla.



Kuva 16. ChatGPT-tekoälyn analyysi kuvan 10 koodista.

3.4 Seurantaviikko 4

Tällä viikolla käytän paljon termiä taulukko ja viittaan sillä kahteen eri asiaan. Mikäli puhun JavaScript array -objektista, sanon taulukko (array). Mikäli puhun taulukosta ilman, että kirjoitan sen perään sulkuihin "array", puhun datan tyypillisestä esitystavasta eli taulukosta. Taulukossa on pystysarakkeita sekä vaakarivejä. Näillä riveillä ja sarakkeilla on soluja, jotka sisältävät tyypillisesti erilaisia arvoja. Esimerkiksi taulukointiohjelma Excelin perusnäky on taulukkonäky.

Maanantai 6.3.2023

Scrum-palaverissa esittelin sovelluskehitystiimille tekemääni graafista käyttöliittymä -osuutta. Kokonaisuudessa on vielä hiottavaa, mutta perustoiminnot ja idea alkavat olla kasassa. Ehdotin, että seuraavaksi voitaisiin piirtää viimeistellymmän näköistä SVG-kuvaa ja luoda kuvia myös eri laiteyksiköille, jotta päästään kokeilemaan, kuinka kirjoittamani koodi taipuu, kun se joutuu käsittelemään

erilaisia SVG-kuvia. En voi luoda ja piirtää SVG-kuvia yksin, sillä en ole riittävän selvillä eri laiteyksiköiden yksityiskohdista, joita voidaan haluta sisällyttää mukaan SVG-kuviin.

Graafisen käyttöliittymän toimiminen eri laiteyksiköille edellyttää jonkin verran koodillisia lisäyksiä laitekohtaisesti, mutta olen pyrkinyt kirjoittamaan funktiot ja logiikan niin, että ne olisivat mahdollisimman universaaleja. Tämä tarkoittaa muun muassa sitä, että koodissa on pyritty välttämään kovakoodattuja arvoja. Kovakoodatut arvot eivät ole muutattavissa, joten ne eivät ole joustavia. Yksinkertaisena esimerkkinä voimme kuvitella funktion, joka laskee kaksi numeroa yhteen. Mikäli funktio kovakoodattaisiin laskemaan yhteen numerot 2 ja 2, ei funktio osaisi tehdä mitään muuta, kuin laskea $2 + 2$ yhteenlaskun. On siis järkevämpää luoda funktio, jolle voidaan joustavasti syöttää mitkä tahansa numerot, jotka se sitten osaa laskea yhteen.

Sain myös uuden tehtävän Scrum-palaverissa. Energiamittauslaitteet tarvitsevat raporttinäkymän, ja minun tulisi alkaa hahmotella, miten se toteutettaisiin front endin puolella ja miltä raportti voisi näyttää. Tiimimme toinen jäsen on työstänyt taustalogiikkaa datan keräykseen liittyen ja hänellä on minulle esimerkkejä siitä, miltä rajapinnasta haettava data voisi mahdollisesti näyttää. Kuvassa 17 näkyy esimerkki haettavan energiamittausdatan mahdollisesta rakenteesta. Kuvassa on siis yhden laiteyksikön kuukausiraportin data, jonka "Value" -taulukossa (array) on objekteja, jotka sisältävät tietyn sisäisen laitteen ("Hwld") mitattavan arvon ("ValueName") mittauksia kuukauden jokaiselta päivältä ("Values"). Kuvassa ei ole mittausdataa kuun jokaiselta päivältä.

Tiiminvetäjä on ehdottanut, että raportti raportissa voisi olla esimerkiksi jonkinlainen 'heatmap' -näkyminen. "Heatmapilla" tarkoitetaan, että taulukon solut värjätään eri sävyisiksi solujen sisältämien arvojen mukaan. Esimerkiksi taulukon suuret arvot voitaisiin värjätä punaisiksi ja pienet arvot sinisiksi. Energiamittaus -kontekstissa meillä voisi olla kalenteritaulukko, joka värjää punaisella värillä ne päivät, joina energian kulutus on ollut suurimmillaan.

Aloin hahmotella karkeasti perinteistä taulukkonäkymää, jossa olisi päivämäärät vasemmassa pystysarakkeessa ja mitattavan arvon nimi taulukon yläosassa vaakasarakkeessa. Hahmotelma on nähtävissä kuvassa 18. Yksi ratkaistava asia, jonka huomasin, oli että mitattavat arvot (kuva 17 "ValueName") voivat olla eri laitteilta (kuva 17 "Hwld"). Tässä esimerkkitiedossa näin tosin ei ole. Eli jos asetan pelkästään mitattavien arvojen nimet taulukkoon, ei ole suoraan selvää, minkä laitteen mittauksesta on kyse, ellen sitten yhdistä arvoja "Hwld" ja "ValueName" toisiinsa. Pohdin, että onko järkevämpää luoda erilliset taulukot eri "Hwld" -arvoille, vai yksi iso taulukko, jossa "Hwld" näkyisi "ValueNamen" vieressä.

```
{
  "id": "12345",
  "createdAt": "2023-03-02T06:35:28.309Z",
  "ReportDate": "2023-03-01T00:00:00.000Z",
  "ReportName": "MonthAvg",
  "Header": "ReportMessage",
  "CustomerId": "xxxxx",
  "Value": [
    {
      "HwId": "abc",
      "ValueName": "L1TotalEa+",
      "Unit": "kWh",
      "Values": {
        "2023-03-01": 0,
        "2023-03-02": 0
      }
    },
    {
      "HwId": "abc",
      "ValueName": "L1TotalEa-",
      "Unit": "kWh",
      "Values": {
        "2023-03-01": 7,
        "2023-03-02": 6,
        "2023-03-03": 8
      }
    },
    {
      "HwId": "abc",
      "ValueName": "L1TotalEap",
      "Unit": "kVAh",
      "Values": {
        "2023-03-01": 13,
        "2023-03-02": 14
      }
    }
  ]
}
```

Kuva 17. Esimerkki mahdollisesta energiamittausdatasta.

	L1TotalEa+	<u>L1TotalEa-</u>	<u>L1TotalEap</u>
2023-03-01	0	7	13
2023-03-02	0	6	14
2023-03-03	-	8	-

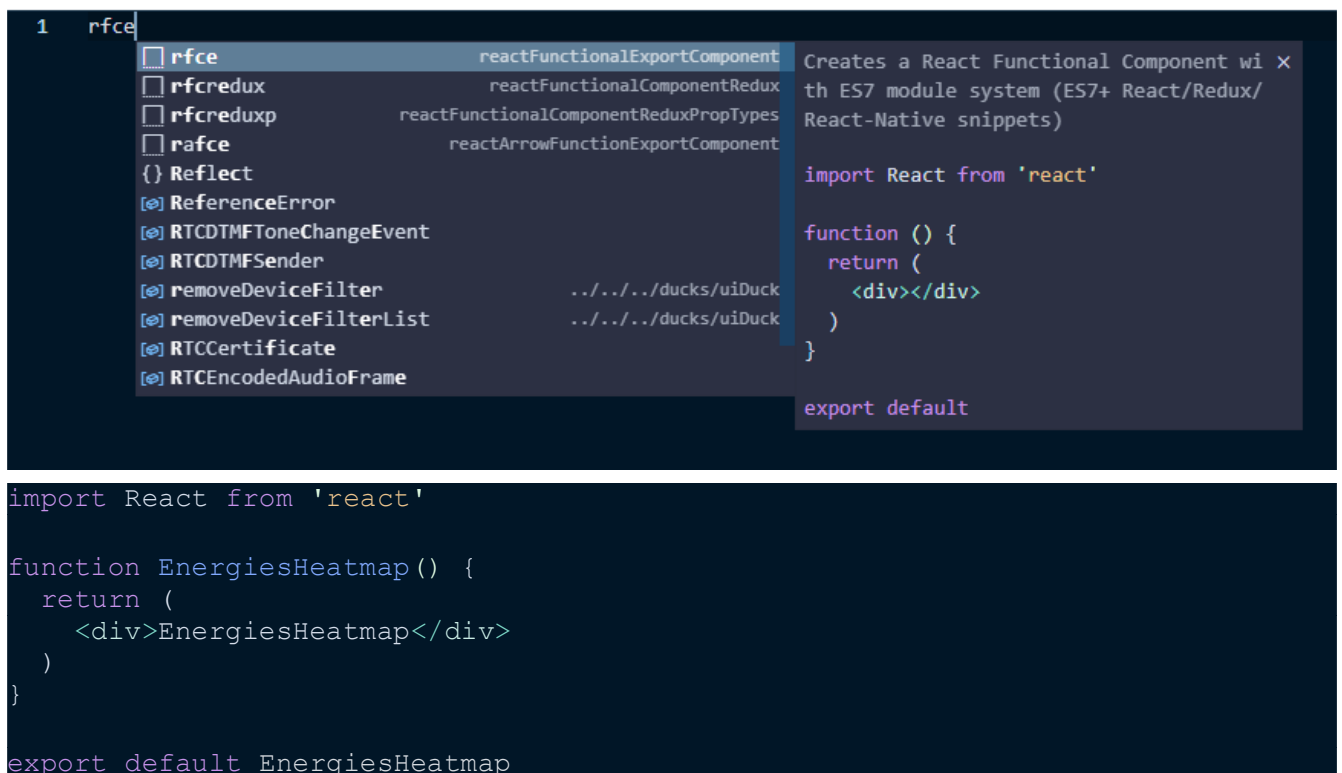
Kuva 18. Ensimmäinen karkea hahmotelma kuvan 17 datan esittämisestä taulukkomuodossa.

En ehtinyt saada koodillisesti mitään toimivaa aikaiseksi ja osa ajastani kului myös heatmap-ominaisuutta pohtiessa. Minun tulee joko rakentaa heatmap-toiminallisuus itse tai mahdollisesti hyödyntää jotain olemassa olevaa kirjastoa. Googletin hakusanoilla "react chart library" ja "react heatmap" ja löysin nopeasti melkoisen valikoiman erilaisia vaihtoehtoja. Esimerkiksi kirjastot kuten Nivo

HeatMap, Apexcharts Heatmap, react-grid-heatmap ja react-heat-map voisivat olla avuksi heatmap-toimintoa koodatessa (Nivo; Apexcharts 2023; arunghosh 2022; uiwjs 2022).

Tiistai 7.3.2023

Aloin koodata maanantain tekemän hahmotelman (kuva 18) perusteella jotain konkreettista, jotta pääsisin eteenpäin tehtävässä. Hyödyntäen Visual Studio Codeen saatavilla olevaa laajennusta (eng. extension) ”ES7+ React/Redux/React-Native/JS snippets” loin komennolla ”rfce” React-komponentin pohjan. ”Snippetit” ovat ikään kuin pikakomentoja, joiden avulla voidaan kirjoittaa koodia nopeammin. Kuvassa 19 on havainnollistettu snippetin käyttöä. Kuvan yläosassa kirjoitan ”rfce” ja painan ”enter”-painiketta tietokoneen näppäimistöstä. Kuvan alaosan koodi ilmestyy tämän jälkeen ruudulle.



```

1 rfce
  rfce      reactFunctionalExportComponent
  rfcredux  reactFunctionalComponentRedux
  rfcreduxp reactFunctionalComponentReduxPropTypes
  rafce     reactArrowFunctionExportComponent
  {} Reflect
  [e] ReferenceError
  [e] RTCDTMFToneChangeEvent
  [e] RTCDTMFSender
  [e] removeDeviceFilter      ../../../../ducks/uiDuck
  [e] removeDeviceFilterList  ../../../../ducks/uiDuck
  [e] RTCCertificate
  [e] RTCEncodedAudioFrame

import React from 'react'

function () {
  return (
    <div></div>
  )
}

export default
import React from 'react'

function EnergiesHeatmap() {
  return (
    <div>EnergiesHeatmap</div>
  )
}

export default EnergiesHeatmap

```

Kuva 19. ”Snippetin” käyttö.

Minulle ei ollut suorilta käsin selvää, miten lähdän hakemaan kuvan 17 datasta arvot talteen tavalla, että saisin niistä järkevästi taulukon luotua. Ensimmäisessä nopeassa versiossa olin kirjoittanut toistorakenteet huonosti ja muun muassa päivämäärät tulostuivat kolmeen eri vaakasarakkeeseen ja taulukko ei näyttänyt lainkaan kuvan 18 hahmotelman mukaiselta. Koodi oli yksinkertainen ja käytännössä se kävi toistorakentein kuvan 17 ”Value”-taulukon (array) objektien ”Values”-objekteja läpi. Koska jokaisella objektilla on samat päivämäärät ja kuvassa 17 ”Values”-objekteja on kolme, tulostui taulukkoon päivämäärät kolmesti.

Jotta saan asetettua taulukkoon päivämäärät vain yhden kerran, niin kuin kuvassa 18 olin hahmotellut, täytyi minun ottaa päivämäärät jollain tavalla talteen. Annoin ChatGPT-tekoälylle tekaistua dataa, ja kysyin ChatGPT-tekoälyltä, kuinka otan päivämäärät talteen tekaistusta datasta omaan taulukkoonsa (array). ChatGPT-tekoäly loi minulle yhden rivin koodia, joka näkyy kuvan 20 alussa. Eli kuvassa 20 muuttuja "dates" on taulukko (array), joka koodin myötä sisältää arvot ["2023-03-01", "2023-03-02", "2023-03-03"]. En muista aikaisemmin käyttäneeni Array.from()-metodia enkä flatMap() -metodia, joten jouduin hieman selvittämään itselleni, että mitä tässä oikein tehdään.

Array.from()-metodi luo taulukon (array) sille syötettävästä muuttujasta, kunhan muuttuja noudattaa JavaScriptin iteraatioprotokolaa tai on valmiiksi taulukon (array) kaltainen objekti (MDN Web Docs 2023c). Iteraatioprotokolan ymmärrys ei ole tarvittavaa tässä kohtaa, vaan riittää että ymmärtää, että Array.from() palauttaa taulukon (array). New Set() -konstruktori luo objektin, joka sisältää ainoastaan uniikkeja avainarvoja, eli tämän avulla poistamme testidatassa useassa objektissa esiintyvät samat päivämäärät (MDN Web Docs 2023d). FlatMap()-metodi yhdistää flat()-metodin ja map()-metodin (MDN Web Docs 2023e). Käytännössä flat()-metodi siistii taulukon (array) arvot yhden taulukon (array) sisälle, mikäli taulukon (array) sisältämät arvot ovat esimerkiksi oman taulukkonsa (array) sisällä (MDN Web Docs 2023e).

Eli taulukko (array) [1, [2], 3, [4,5]] olisi flat() -metodin käytön jälkeen [1, 2, 3, 4, 5]. Object.keys()-metodia käytetään objektin avainarvojen läpikäymiseen ja sort()-metodi asettaa taulukon (array) elementit järjestykseen.

```
// Get unique keys for all the dates
const dates = Array.from(new Set(testData.Value.flatMap(value =>
Object.keys(value.Values)))) .sort();

// Get unique value names
const valueNames = testData.Value.map(value => `${value.ValueName}
(${value.Unit})`);

// Create table header
const header = ["", ...valueNames];

// Create table rows
const rows = dates.map(date => {
  const values = testData.Value.reduce((acc, value) => {
    acc[value.ValueName] = value.Values[date] ?? "-";
    return acc;
  }, {});
  return [date, ...Object.values(values)];
});
```

Kuva 20. Tällä koodilla luodaan kuvan 21 taulukko kuvan 17 datasta.

Kuvan 20 `// Create table rows` kommentin alla oleva koodi on myös ChatGPT-tekoälyn kirjoittama. Siinä käymme läpi toistorakenteen avulla päivämäärät sisältävän `"dates"`-taulukon (array) luoden jokaiselle päivämäärälle `"values"`-objektin. `Reduce()`-metodilla käymme läpi `testData.Value`-taulukon (array) ja poimimme sieltä kaikki mittausarvot päivämäärän mukaan talteen. Mikäli jollekin mittaukselle ei löydy arvoa tietylle päivälle, annetaan sille arvoksi `"-"`. Eli jos katsomme kuvan 17 esimerkkidataa, niin laitamme siitä siis kaikki `"2023-03-01"` päivien arvot omaan taulukkoonsa (array), kaikki `"2023-03-02"` päivien arvot omaan taulukkoonsa (array) ja niin edelleen.

Jos hyödynnämme `console.table()`-metodia, voimme tulostaa selaimen konsoliin taulukon kuvan 20 koodin avulla ja katsoa, onko taulukko oikean suuntainen.

```
console.table([header, ...rows]);
```

Kuvasta 21 voimme havaita, että taulukon runko näyttää hyvältä ja tästä on hyvä jatkaa.

(index)	0	1	2	3
0	0	'L1TotalEa+ (kWh)'	'L1TotalEa- (kWh)'	'L1TotalEap (kVAh)'
1	'2023-03-01'	0	7	13
2	'2023-03-02'	0	6	14
3	'2023-03-03'	'-'	8	'-'

Kuva 21. Kuvan 20 koodilla luotu taulukko, joka vastaa kuvan 18 hahmotelmaa.

Keskiviikko 8.3.2023

Halusin saada tämän päivän aikana taulukon tulostettua sovelluksen käyttöliittymään ja lähdin suoraan jatkamaan siitä, mihin olin tiistaina jäänyt. Käyttöliittymään sai taulukon tehtyä käymällä `"header"`-taulukon (array) ja `"rows"`-taulukon (array) toistorakentein läpi. Asetin taulukot (array) React-Bootstrap `<Row>` elementtien sisään ja jokaisen taulukon (array) sisältämät arvot `<Col>` elementin sisään. Bootstrap on front end -käyttöön luotu sovelluskehys, eli iso paketti valmiiksi luotua koodia, jonka avulla voi nopeasti rakentaa näyttäviä ja siistejä käyttöliittymiä (Bootstrap team). React-Bootstrap on taas nimenomaan React-sovellukselle suunnattu versio Bootstrapista, jossa Bootstrapin komponentteja on muokattu niin, jotta ne sopivat React-ympäristöön paremmin (React-Bootstrap).

Halusin koittaa luoda heatmap -tyylisen ulkoasun taulukkoon itse ilman uusia ulkopuolisia kirjastoja. Eli taulukon solun täytyisi värjättyä sitä voimakkaammalla värisävyllä, mitä suuremman numeraalisen arvon solu sisältää verrattuna taulukon muihin soluihin nähden. Pohdin, että jos annan jokaiselle solulle oman `rgba`-väriarvon `"inline-style"`-ominaisuutta hyödyntäen, voin tuolle `rgba`-väriarvolle syöttää `alpha`-arvon parametrina. `Rgba` tulee siis sanoista `red`, `green`, `blue` ja `alpha` ja `rgba`-notaation avulla värjätään sovelluksen elementtejä. `Alpha`-arvo on värin läpinäkyvyys ja se voi olla

mitä vain 0 ja 1 välillä. Arvon ollessa 0, väri on näkymätön, ja arvon ollessa 1 värin voisi sanoa olevan täydessä voimakkuudessa. (MDN Web Docs 2023f.)

Aloin ohjelmoida ratkaisua, jolla saisin jokaiselle solun arvolle toisiinsa verrannollisen alpha-arvon. Loin "allValuesArray"-muuttujan ja hain siihen kaikki testidatasta löytyvät mittausarvot. Seuraavaksi etsin näistä arvoista pienimmän ja suurimman. Tämän jälkeen hyödynsin yksinkertaista matemaatiikkaa luodakseni jokaiselle arvolle kaveriksi "percentage"-arvon, joka kertoi siis, kuinka suuri tämä arvo on verrattuna listan suurimpaan arvoon. Tämä koodi on nähtävissä kuvassa 22.

```
// Get min and max value from all the measurements
const allValuesArray = rows.map(row => row.map(value => isNaN(value) ? 0 : value));
const flattenedArray = [].concat(...allValuesArray);
const min = Math.min(...flattenedArray);
const max = Math.max(...flattenedArray);

// Add percentage value to measurement values. It is used to determine RGB color Alpha value.
const rowsWithColorPercentage = rows.map(row => {
  const newRow = [];
  row.forEach((value, i) => {
    // First value is the date, give it 0 percentage
    if (i === 0) {
      newRow.push({ value: value, percentage: 0 });
    } else {
      const percentage = value === "-" ? 0 : (((value - min) / (max - min))).toFixed(2);
      newRow.push({ value: value, percentage });
    }
  });
  return newRow;
});
```

Kuva 22. Heatmap-taulukon logiikkaa.

Seuraavaksi piti saada vain tämä uusi "percentage"-arvo käyttöön. Kuvassa 23 näkyy, kuinka annan <Col> elementille "inline-style"-ominaisuutta hyödyntäen väriarvon. Kuvassa 24 on nähtävillä taulukon ensimmäinen versio, jossa on koitettu värjätä soluja niiden sisältämien arvojen mukaan.

```
{rowsWithColorPercentage.map((measurements, idx) => (
  <Row key={idx}>
    {measurements.map((measurement, index) => (
      <Col key={index} className="header" style={{marginTop: "1px",
        backgroundColor: `rgba(220, 70, 40, ${measurement.percentage})`} }>
        {measurement.value}
      </Col>
    ))}
  </Row>
))}
```

Kuva 23. Tässä koodissa luodaan kuvan 24 taulukkoa.

	ABC LITotalEa+ kWh	ABC LITotalEa- kWh	ABC LITotalEap kVAh	ZXY F Hz
2023-03-01	0	7	13	50
2023-03-02	1	6	14	49.7
2023-03-03	2	8	20	51.1
2023-03-04	1	9	13	49.8
2023-03-05	1	4	9	50.1

Kuva 24. Energiaraporttitaulukon työversio, jossa ensimmäinen iteraatio soluja värjäävästä "heatmap"-toiminallisuudesta.

Kuten kuvassa 24 voi havaita, olin koodillani värjännyt suurimmat arvot kirkkaimman punaisiksi, niitä pienemmät arvot alati vähemmän punaisiksi ja kaikista pienimmät arvot eivät näyttäneet lainkaan punaisilta. Tajusin välittömästi taulukkoa katsoessani, että olin tehnyt loogisen erheen. Pystysarakkeiden arvojen yksiköt eli kWh, kVAh ja Hz eivät ole verrannollisia keskenään, eli en voi luoda "heatmappia" verraten kaikkia arvoja keskenään. Jotta "heatmap" toimisi järkevästi tässä taulukossa, minun täytyy "percentage"-arvoa luodessa verrata ainoastaan yhden mittauspisteen arvoja keskenään, eikä taulukon kaikkia arvoja keskenään. Esimerkiksi, jos katsomme kuvan 24 toista pystysaraketta, täytyy meidän tässä sarakkeessa verrata keskenään ainoastaan arvoja [0, 1, 2, 1, 1]. Jos näille luodaan "percentage" arvot, olisivat ne [0, 0.5, 1, 0.5, 0.5].

Torstai 9.3.2023

Tavoitteena oli tämän päivän aikana korjata energiaraporttitaulukon "heatmap"-logiikka. Minun täytyisi toistorakenteen avulla käydä kuvan 17 "Value" -taulukon (array) objektien "Values"-objektit läpi ja tämän toistorakenteen sisällä löytää jokaisen "Values"-objektin suurin ja pienin arvo sekä määrittämisen avulla asettaa jokaiselle arvolle kaveriksi oma "percentage"-arvo. Kuvassa 25 olen näin tehnyt ja muuttuja "heatMapdata" on siis taulukko (array), joka sisältää jokaiselle mittauspisteelle oman taulukon (array), jotka sisältävät kyseisen mittauspisteen mittausarvot sekä "percentage"-arvot.

```
// Add a new percentage value to each measurement value in given data
const heatMapData = testData.Value.map((measurementPointData) => {

    // Get min and max value from single measurement point measurements
    const measurementPointDataValues =
Object.values(measurementPointData.Values);
    const min = Math.min(...measurementPointDataValues);
    const max = Math.max(...measurementPointDataValues);

    const valuesArray = Object.entries(measurementPointData.Values).map(([date,
value], index, array) => {
        const percentage = value === "" ? 0 : ((value - min) / (max -
min)).toFixed(2);
        return {[date]: value, percentage};
    });

    return {Values: valuesArray};
});

// Create new rows with the desired format
const rowData = [];
for (let i = 0; i < dates.length; i++) {
    const row = [];
    row.push({"value": dates[i], "percentage": "0.00"});
    for (let j = 0; j < heatMapData.length; j++) {
        let valueFound = false;
        for (let k = 0; k < heatMapData[j].Values.length; k++) {
            if (Object.keys(heatMapData[j].Values[k])[0] === dates[i]) {
                row.push({"value": heatMapData[j].Values[k][dates[i]], "percentage":
heatMapData[j].Values[k].percentage});
                valueFound = true;
                break;
            }
        }
        if (!valueFound) {
            row.push({"value": 0, "percentage": "0.00"});
        }
    }
    rowData.push(row);
}
```

Kuva 25. Heatmap-taulukkologiikan toinen versio.

Seuraavaksi piti saada tämän "heatMapData"-muuttujan sisältämä data uudelleen järjesteltyä, jotta se voitaisiin toistorakenteen avulla purkaa taulukkoon. Tulostin selaimen konsoliin console.log()-metodilla "heatMapData"-muuttujan sisällön. Tämän jälkeen muokkasin siitä arkaluontoiset asiat pois ja annoin datan ChatGPT-tekoälyn käsiteltäväksi. Käskin ChatGPT-tekoälyä muokkaamaan datan haluamani kaltaiseksi ja ChatGPT-tekoäly kirjoitti minulle kuvassa 25 näkyvän logiikan osuuden, joka täyttää "rowData"-taulukon (array) "heatMapData"-muuttujan arvoilla hyödyntäen myös "dates"-muuttujaa, joka sisältää siis uniikit päivämäärät. Lopulta sain koodin toimimaan oikein ja

taulukko tulostui käyttöliittymään haluamallani tavalla. Kuvassa 26 taulukon solut värjäntyvät oikein.

	LTTotalEa+ kWh	LTTotalEa- kWh	LTTotalEap kVAh	F Hz
2023-03-01	0	7	13	50
2023-03-02	1	6	14	49.7
2023-03-03	2	8	20	51.1
2023-03-04	1	9	13	49.8
2023-03-05	1	4	9	50.1

Kuva 26. Energiaraporttidatataulukon työversio ja toinen iteraatio soluja värjäävästä "heatmap"-toiminallisuudesta.

Perjantai 10.3.2023

Tavoitteena tälle päivälle oli jatkaa energiaraportin tekemistä kohottamalla sen visuaalista ilmettä, kokeilemalla erilaisia värejä ja katsomalla kuinka taulukko tulostuu erilaisella datalla. Loin itselleni testipainikkeita, joiden avulla pystyin nopeasti vaihtamaan taulukon käyttämää dataa, vaihtamaan värjättyjen solujen väriä sekä tarvittaessa poistamaan solujen värjäyksen. Halusin myös jalostaa komponentin logiikkaa, jotta taulukon solu jäisi tyhjäksi, mikäli kyseisellä mittauspisteellä ei ole mitausarvoa kyseiselle päivälle.

Päivänä aikana selkeytin koodia muun muassa vaihtamalla muuttujanimiä kuvaavimmiksi sekä lisäämällä kommentteja koodiin sellaisiin paikkoihin, jotka mielestäni kommentteja kaipasivat. Koodin kommentointi on ymmärtääkseni vahvasti mielipiteitä jakava aihe ohjelmointikehittäjien keskuudessa, mutta itse pidän kommentoinnista ja mieluummin luen ylikommentoitua koodia, kuin täysin kommentoimatonta koodia.

Tiimillämme oli tänään myös "sprint planning" eli kehitysjakson suunnittelupalaveri, joka kuuluu Scrumin ketterän kehityksen toimintamalliin. Kävimme läpi, mitä tiimi ja sen jäsenet ovat saaneet aikaiseksi kehitysjakson aikana ja mitä otetaan työn alle seuraavaan kehitysjaksoon. Hyvin potentiaalisen asiakkaan suunnalta oli tullut toive sovellukseen liittyen, jonka toteutus osuisi täysin minun tontilleni eli front endin -puolelle. Aloitan heti seuraavan viikon maanantaina tämän uuden tehtävän parissa. Tämä tarkoittaa, että energiaraportin tekeminen saisi jäädä toistaiseksi odottamaan.

Päivän päätteeksi minulla oli energiaraporttitaulukko mielestäni hyvällä mallilla. Sen pariin on hyvä palata myöhemminkin, sillä olin juuri sopivasti päivän aikana lisännyt kommentteja ja selkeyttänyt koodia, jotta sitä olisi helpompi lukea pidemmänkin tauon jälkeen. Kuvat 27 ja 28 ovat esimerkkikuvia taulukosta. Kuvassa 27 on testipainikkeet näkyvillä, joiden avulla olen muun muassa värjännyt

kuvan solut vihreäksi punaisen värin sijaan. Kuvassa 27 näkyy myös, että mikäli mittausarvoa ei ole, jää solu tyhjäksi. Kuvassa 28 on näkyvillä yhden päivän raportti, eli vasemmanpuoleisessa sarakkeessa näkyy tunnit päivämäärien sijaan.

	Showing Day	Change color	Highlighting high values	Change Test Data					
	LTTotalEa+ kWh	LTTotalEa- kWh	LTTotalEap kVAh	F Hz	V1 V	DEVICE XY Value Ab UNIT Zx	DEVICE UA Value lo UNIT Px	DEVICE LOONG NAME VALUE Loong Value UNIT uunit	
2023-03-01	0	7	13	50	226	500	0.1		
2023-03-02	1	6	14	49.7	226.04	470	0.2		
2023-03-03	2	8	20	51.1		510	0.3		
2023-03-04	1	9	13	49.8		601	0.4	1	
2023-03-05	1	4	9	50.1		302			
2023-03-06	1	1	1	50.1		77	1.3	2	
2023-03-07	1	1	1	49.9		123	0.1		
2023-03-08	1	1	1	50		234	0	3	
2023-03-09	1	1	1	49.7		345	1.1	4	
2023-03-10	1	1	1	51		456	0.7	5	
2023-03-11	1	1	1	50.5		723			
2023-03-12	1	1	1	49.9		500			

Kuva 27. Energiaraporttitaulukon työversio, jossa mukana testipainikkeet taulukon manipuloimiseksi.

2023-03-02	LTTotalEa+ kWh	LTTotalEa- kWh	LTTotalEap kVAh
00:00:00	0	0	1
01:00:00	1	1	0
02:00:00	2	5	1
03:00:00	3	1	1
04:00:00	4	11	0
05:00:00	5	3	1
06:00:00	6	4	0
07:00:00	7	10	1
08:00:00	8	9	0
09:00:00	9	8	1
10:00:00	10	7	1
11:00:00	11	6	0
12:00:00	12	5	1
13:00:00	13	4	0

Kuva 28. Energiaraporttitaulukon työversio, jossa energiaraporttidata näytetään päivän jokaiselta tunnilta.

Viikkoanalyysi 4

Tällä viikolla käytin kaikki 5 työpäivää uuden energiaraporttinäkymän kehittämiseen. Ensimmäiseksi nousee mieleen, että vaikka jossain sovelluksessa jokin näkymä tai ominaisuus vaikuttaa yksinkertaiselta katsojan silmään, voi näkymän tai ominaisuuden tekeminen olla ollut haastavampaa ja enemmän aikaa vievää, kuin ulkopuolelta uskoisikaan. Esimerkiksi jos katsomme kuvan 18 hahmotelmaa, voisi kuvitella, että kyllä kuvan kaltaisen näkymän ohjelmoi hetkessä käyttöliittymään. Entä jos data haetaankin monimutkaisesta datarakenteesta ja siinä pitää ottaa huomioon erilaisia seikkoja, kuten se mahdollisuus, että data voi olla rikkonaista, eli sieltä puuttuu palasia? Yksinkertaisen näköisen taulukon tekemiseenkin voi joutua käyttämään yllättävänkin paljon aikaa, kun alkaa miettiä, mitä kaikkea on otettava huomioon lopullisessa versiossa.

Toisena seikkana haluan pohtia muuttujien nimeämistä ja muutenkin nimeämiskäytäntöjä sovelluskehityksessä. Kuvassa 17 olevassa datassa minua häiritsi koko viikon se, että raportin datan tieto on asetettu "Value" -nimisen taulukon sisälle ja tämän sisällä on objekteja, joilla on "Values" -nimisiä objekteja, jotka sisältävät sitten yksittäisiä arvoja, joihin koodissa halutaan mahdollisesti viitata termillä "Value". Jos objektit muutetaan taulukkomuotoon (array), näyttää datan käsittely tältä: "Value[0].Values[0].value". Ainakin minun aivoni menevät tässä kohtaa jo sekaisin, että mistä oikeasti puhutaan, kun kaikkiin arvoihin viitataan aina termillä "Value". Itse nimeäisin "Value"-taulukon (array) esimerkiksi "ReportData"-taulukoksi (array), ja jokaisen dataobjektin sisällä olevan "Values"-objektin "Measurements"-objektiksi. Muutosten jälkeen datan käsittely voisi näyttää tältä: "ReportData[0].Measurements[0].value". Tämän muutoksen jälkeen ei tarvitse edes nähdä alkuperäistä dataa tietääkseen, että tässä käsitellään jonkin raportin dataa, joka sisältää mittaustuloksia.

Funktioiden ja muuttujien nimeämisestä ja koodin kirjoitusasusta on paljon ohjeita ja mielipiteitä löydettävänä. Näihin asioihin ei ole yhtä oikeaa totuutta, mutta on tiettyjä käytäntöjä, joiden noudattamisen on moni sovelluskehittäjä todennut hyväksi. Esimerkiksi sovelluskehittäjät Ran Greenberg, Varun Pujari, Lior Halfon, Apurva Jain kehottavat käyttämään muuttujille ja funktioille merkityksellisiä ja ymmärrettäviä nimiä, vaikka niistä tulisi pitkiä. (Greenberg 10.10.2019; Pujari 27.5.2020; Halfon 5.2.2019; Jain 25.1.2021). En löytänyt aiheesta tieteellisiä artikkeleita, joissa olisi tutkimuksin todistettu näiden nimeämiskäytäntöjen paremmuus. Oma kokemukseni on, että jos nimeän muuttujan kirjaimella "x" tai vaihtoehtoisesti nimellä "amountOfWaterBottlesInStorage", on tuo pitempi muuttujan nimi kuvaavampi, selkeämpi ja ymmärrettävämpi. Kuvaavat nimet auttavat mielestäni erityisesti silloin, kun luet esimerkiksi toisen henkilön kirjoittamaa koodia tai palaat jonkin koodipätkän pariin pitkän tauon jälkeen.

Yksi asia, jota hyödynsin tällä viikolla enemmän kuin normaalisti, oli ChatGPT-tekoäly. Olen aikaisemmin esittänyt ChatGPT-tekoälylle melko yksinkertaisia kysymyksiä tai kysynyt sen mielipidettä

joistakin kirjoittamistani funktioista, mutta en ollut kokeillut ohjeistaa sitä kirjoittamaan koodia puolestani ennen tätä viikkoa. Panin merkille, että ChatGPT-tekoäly hyödynsi paljon sellaisia metodeja ja tapoja, joita itse en ole juurikaan tai jopa lainkaan käyttänyt. Esimerkiksi tiistain 7.3.2023 päiväraportissa kerron, kuinka jouduin selvittämään itselleni, mitä ChatGPT-tekoäly on minulle koodannut, sillä se hyödynsi koodissaan minulle vieraampia metodeja kuten "Array.from()", "new Set()", "flatMap()" ja "reduce()". Tämä oli itse asiassa hyvin opettavaista minulle, sillä sain suoraan käytännön esimerkkejä, miten näitä metodeja voidaan käyttää. Mielestäni on erittäin tärkeää, että mikäli hyödyntää ChatGPT-tekoälyn kaltaista työkalua koodin luomiseen, koittaa edelleen parhaansa mukaan ymmärtää, mitä koodissa tapahtuu.

Viikon aikana kehitin pikkuhiljaa omaa logiikkaa ChatGPT-tekoälyn luoman koodipohjan päälle. Vaihtelin muun muassa muuttujanimiä selkeämmäksi ja korvasin reduce() -metodin käytön tavanomaisemmilla toistorakenteilla, sillä koin reduce()-metodin itselleni haastavaksi sisäistää kunnolla. Toisin sanoen ChatGPT-tekoälystä oli suuri apu ja sen luoma koodi olisi jollain asteella jopa toiminut sellaisenaan, mutta koodin viimeistely ja sen saaminen yhdenmukaiseksi kehittämäni sovelluksen muun koodin kanssa jäi minun käsiini.

ChatGPT-tekoäly on voimakas työkalu, jonka uskon vaikuttavan moniinkin aloihin ja työtehtäviin hyvinkin pian. Esimerkiksi Tanya Tsui niminen henkilö kertoo yksityiskohtaisesti blogissaan, kuinka ChatGPT-tekoäly ohjelmoi hänelle python-koodilla ruudukon soluja, joiden tulisi olla yhteen sopivia geotietokehysten (eng. geoDataFrame) kanssa. Geotietokehys on tässä tapauksessa GeoPandas-kirjaston "GeoDataFrame"-objekti, jota hyödynnetään paikkatietojen käsittelyssä python-koodin avulla (GeoPandas developers 2022). Tanya kertoo, että aluksi ChatGPT-tekoälyn luoma koodi ei toiminut, kuten oli toivottu, mutta kun Tanya pyysi ChatGPT-tekoälyä kirjoittamaan koodin uudelleen ja antoi tälle tarkemmat ohjeet, kirjoitti ChatGPT-tekoäly pätkän hyvää ja toimivaa koodia. (Tsui 17.1.2023).

3.5 Seurantaviikko 5

Maanantai 13.3.2023

Energia raportinäkymän kehitys asetettiin toistaiseksi jäihin ja tehtäväkseni priorisoitui luoda uusi yksinkertaistettu laitenäkymä sensorityyppisille yksinkertaisille laitteille. Sovelluksessa on olemassa kaikille laitteille korttinäkymä, jossa siis tyylliteltyihin suorakulmaisiin laatikoihin ("kortteihin") tulostuu näkyville laitediagnostiikkaa ja mittausdataa. Korttinäkymä on hyvä laitekokonaisuuksille, mutta näille sensorityyppisille yksinkertaisemmille yksinään toimiville laitteille voisi pelkistetty listatyylinen esitysmuoto palvella käyttäjää paremmin. Pelkistetty listanäkymä voisi olla erityisen hyödyllinen tilanteissa, jossa näitä sensorilaitteita on todella paljon.

Tämän näkymän luominen on täysin front endille kuuluva tehtävä, sillä kaikki näkymää varten tarvittavat datayhteydet on jo olemassa ja toiminnassa. Mielestäni tehtävän pystyi jakamaan kahteen pääosaan:

1. Minun täytyy saada uuteen näkymään haettua sensorityyppisten laitteiden tiedot ja niiden mitausdata.
2. Minun täytyy saada esitettyä data listassa ja tyyllitellä lista.

Epäilin alustavasti, että kohta 1 tulee viemään huomattavasti enemmän aikaa. Kunhan data olisi järkevästi haettu komponentteihin, ei minulla menisi kauaa asetella dataa nätisti listamuotoon. Minun ei tässä vaiheessa tarvitse miettiä listalle filteri- tai hakutoimintoja, vaan niitä voitaisiin lisätä tulevaisuudessa ja tarvittaessa.

Olin saanut haettua sensorityyppisiä laitteita sisältävät kohteet omaan taulukkoonsa ja koitin syventyä logiikkaan, jolla olemassa oleva korttinäkymä hakee laitteiden tietoja käyttöönsä. Vaikka olen jo 9 kuukautta koodannut tämän sovelluksen parissa, en todellakaan ole täysin selvillä kaikista tekniikoista ja asioista yksityiskohtia myöten, joita sovelluksen front end -puolen koodi sisältää. Sovelluksesta löytyy vieläkin koodilohkoja, joiden kanssa en ole ollut lainkaan tekemisissä.

Jotta pääsemme käsiksi sensoreiden mittaustietoihin, täytyy selvittää käyttäjän oikeudet ja tarkistaa tämän todennustunnus (eng. authentication token), avata WebSocket-yhteydet sensoreihin ja noiden WebSocket-yhteyksien kautta ottaa sensoreiden keräämä data talteen. WebSocket-tekniologia mahdollistaa jatkuvan kommunikaation serverin ja selaimen välillä, jotta data voi kulkea vapaasti ilman tyypillisesti käytettäviä http-pyyntöjä (MDN Web Docs 2023g). Käyttäjäoikeudet on taas määritelty asiakaskohtaisesti ja niiden perusteella käyttäjä näkee hänen oikeuksillensa sallittuja laitteita sovelluksen käyttöliittymästä.

Kopioin valmista koodia korttinäkymän komponenteista uuteen listanäkymääni ja koitin saada datan haettua listaan, mutta en vielä onnistunut tässä. Työpäivä meni suurimmaksi osaksi jälleen olemassa olevaa koodia tutkiessa sekä pohtiessa tulevan listanäkymän komponenttirakennetta. Työpäivän päätteeksi huomasin, että olin yrittänyt avata WebSocket-yhteyksiä puutteellisilla tiedoilla. Eli toisin sanoen http pyyntö -metodin viestikuormasta (eng. payload) puuttui olennaisia arvoja, joita vaadittiin, että WebSocket-yhteys palvelimen ja selaimen välillä olisi saatu auki.

Tiistai 14.3.2023

Tarkoitukseni oli jatkaa sensorilistanäkymän parissa, mutta aamuksi oli sovittu graafiseen käyttöliittymään liittyvä palaveri. Eräs kollegani oli ilmoittanut kiinnostuksensa piirtää ja luoda grafiikkaa tähän graafiseen käyttöliittymään, joten yhdessä tiiminvetäjän kanssa opastimme ja kävimme läpi,

mitä on jo tehty ja mitä seuraavaksi tulisi tehdä graafisen suunnittelun suhteen. Olin luonut kuitenkin pätevän pohjapiirustuksen, jota hyödyntäen kollega saa varmasti sommiteltua ja luotua minua parempaa jälkeä.

Iltapäivästä jatkoin sensorilistanäkymän kehittämistä. Sain WebSocket-yhteydet auki lisättyäni puuttuvat muuttujat avauspyynnön tietokuormaan. Data ei jostain syystä tullut vielä Redux tilahallintaan asti, eli jokin vaihe tai funktio minulta vielä puuttui sensorilistanäkymästä, joka oli olemassa korttinäkymässä. Minulta puuttui "SignalR"-kirjaston käyttäminen luomissani komponenteissa. SignalR on Microsoftin kehittämä kirjasto, jota hyödynnetään datan lähettämiseen serveriltä selaimelle ilman, että selaimen tarvitsee dataa erikseen pyytää (Microsoft 9.19.2020). Yleistä on, että SignalR ja WebSocket -teknologioita käytetään yhdessä.

Kun olin lisännyt SignalR -toiminnot koodiin, niin sensoreiden data tallentui Redux tilahallintaan ja sieltä pystyin hakemaan datan sensorilistakomponenttien käyttöön. Olin tehnyt alustavasti seuraavanlaisen komponenttirakenteen:

1. SensorListViewContainer.jsx. Tämä hakee kaikki käyttäjän oikeuksien alle lukeutuvat kohteet ja erottelee näistä kohteista erikseen ne, jotka sisältävät sensortyyppisiä laitteita. Nämä kohteet käydään toistorakenteella läpi ja luodaan jokaiselle <SensorRow> -komponentti.
2. SensorRow.jsx. Tämä komponentti käy saamansa kohteen kaikki sensorilaitteet läpi toistorakenteella ja luo jokaiselle sensorille <MeasurementRow> -komponentin.
3. MeasurementRow.jsx. Tämä komponentti tulostaa yhdeltä sensorilaitteelta saadut tiedot käyttöliittymään.

Koitan lähteä tällä rakenteella liikkeelle ja seuraavien päivien aikana tulen kehittämään itse käyttöliittymää sensorilistanäkymän osalta ja tekemään siitä visuaalisesti selkeän, sekä miellyttävän katsoa ja käyttää.

Keskiviikko 15.3.2023

Tarkoitus oli tämän päivän aikana saada ensimmäinen versio sensorilistanäkymästä aikaiseksi. Aloin sommitella dataa käyttöliittymään hyödyntäen React-Bootstrap-kirjaston "Grid" -systeemiä, joka on siis käyttöliittymien sisällön asetteluun tarkoitettu joukko valmiiksi luotuja responsiivisia komponentteja. Grid-systeemi perustuu vahvasti kolmen komponentin ympärille, jotka ovat "Container", "Row" ja "Col". "Container" ympäröi sisältönsä toppausmarginaaleilla eli reunuksilla (eng. padding), jotka toimivat käytännössä samalla tavalla, kuin esimerkiksi reunusten asettaminen

Word-tekstidokumenttiin. "Container"-komponentin sisään asetellaan tyypillisesti "Row"-komponentteja, jotka ovat rivejä. Yleensä "Row"-komponenttien sisään asetellaan vielä "Col"-komponentteja, jotka ovat sarakkeita. Usein näiden "Col"-komponenttien sisään tulostetaan varsinainen sisältö, esimerkiksi minun sensorilistanäkymässäni näihin "Col"-komponentteihin voitaisiin sijoittaa mittausdataa. (React-Bootstrap b.)

Front end -kehityksessä näkymän hienosäätö ja tyyllittelyn iterointi vie ainakin itseltäni usein runsaasti aikaa. Näkymän saaminen kelvollisen näköiseksi on usein nopeaa, mutta monesti sitä eksyy hiomaan jotain tiettyä yksityiskohtaa tarkoituksena saada siitä mahdollisimman täydellinen omaan silmään. En myöskään hyödynnä suunnittelutyökaluja, vaan koodaan ja kokeilen, miltä eri asiat näyttävät. Tässä voisi itsellä kehittymisen paikka löytää jokin sopiva suunnittelutyökalu ja ottaa se haltuun, jotta sitä olisi miellyttävä ja nopea käyttää. Saatan välillä tehdä jotain todella karkeaa ja yksinkertaista visualisointia ennen koodaamista. Tällaisesta erittäin yksinkertaisesta hahmotte-
lusta, jota välillä teen voisi olla toimia esimerkkinä seurantaviikko 4 kuva 18 taulukkohahmotelma, joka on tehty Google Docs työkalulla.

Päivän päätteeksi pääsin tavoitteeseeni ja sensorilistanäkymän ensimmäinen versio oli valmis. Näkymästä puuttuu vielä toiminallisuuksia, joita tämän kaltaiseen listanäkymään tullaan varmasti kaipaamaan kuten hakutoiminnot ja erilaiset suodatustoiminnot.

Torstai 16.3.2023

Sensorilaitteiden data kulkee back endin läpi ennen front endille tuloa ja back end lisää sensoridatan viestikuormaan (eng. payload) erilaisia front endin tarvitsemia tietoja dataan liittyen. Yksi tällainen back endin lisäämä tieto on, ovatko mittausarvot ylittäneet tai alittaneet tiettyjä etukäteen määriteltyjä raja-arvoja. Kutsutaan tätä tietoa "status"-tiedoksi. Tätä statustietoa front end hyödyntää muun muassa siihen, että voimme käyttöliittymässä tämän statuksen perusteella esimerkiksi vaihtaa jonkin tekstikentän värin punaiseksi ilmaisemaan sitä, että mittausarvo on kriittinen.

Halusin listanäkymässä värjätä punaisella ne rivit, joissa mittausarvot ovat kriittisellä tasolla. Ongelmaksi kuitenkin muodostui se, että jostain syystä Reduxin tilahallinnassa oleva tieto sensorilaitteiden data väitti, että kaikkien sensoreiden status oli "ok", vaikka muutaman sensorin mittausarvot olivat todellisuudessa kriittiset. Käyttöliittymän vanhassa korttinäkymässä näiden sensoreiden kortit olivat värjäytyneet punaisiksi, kuten kuuluu, joten tiesin vian olevan sensorilistanäkymän koodissa.

Kävin moneen kertaan läpi koodiani löytämättä syytä ongelmaani. Omituiseksi ainakin omasta mielestäni ongelman tekee muun muassa se, että sensorilistanäkymäni käyttää hyvin pitkälti samoja funktioita, kuin vanha korttinäkymä. Eli miksi tieto, joka haetaan samalla tavalla, samasta lähteestä

ja käsitellään samojen funktioiden avulla front endissä, tallentaa Reduxin tilahallintaan tiedon eri tavalla? Joskus sitä on täysin sokea jonkin ongelman syyllä tai ratkaisulle ja siltä minusta tuntui tämän ongelman kanssa. Tuntien jälkeen jätin tämän ongelman hautumaan, ja siirryin eteenpäin.

Halusin rakentaa sensorilistanäkymään mittausarvojen järjestely- ja suodatustoimintoja käyttäjää varten. Esimerkiksi listan järjestäminen sensorilaitteiden nimen mukaan tai niiden mittaamien arvojen mukaan on varmasti toivottu ominaisuus. Myös tässä törmäsin ongelmiin. Tiistaina kuvailemani komponenttirakenne (<SensorListViewContainer /> → <SensorRow /> → <MeasurementRow />) ja tapa, jolla olin toistorakenteiden avulla käynyt dataa läpi, aiheutti vaikeuksia dataobjektien järjestykseen laittamisessa. Olin alun perin komponentteja rakentaessa ottanut jonkin verran mallia olemassa olevasta korttinäkymästä ja se saattoi olla virhe, sillä korttinäkymä on rakennettu ilman järjestely- tai suodatustoimintoja. Eli sensorilistanäkymä täytyisi rakentaa eri tavalla.

Päivän aikana optioimin koodia, vaihtelin muuttujanimiä kuvaavimmiksi ja hahmottelin suodatustoimintoja. Kuvassa 29 näkyy sensorilistanäkymä, jota olen työstänyt. Kyseisestä kuvasta on sensuroitu asiakas- ja laitetietoja. Kuvassa 29 oranssin väriset sensuurilaatikot ovat kohteita/lokaatioita ja valkoisen väriset sensuurilaatikot ovat kohteessa olevia sensorilaitteita. Lisäsin päivän aikana toiminnallisuuden piilottaa minkä tahansa kohteen laitteet sensorilistanäkymästä klikkaamalla kohteen nimeä. Lisäsin myös toiminnallisuuden, jossa klikkaamalla mitä tahansa sensoririviä, korostuu se rivin väritys.

Level	Free Volume	FB Level	Used Volume	Remaining Feed	Silo Remaining Feed
5.27 m	20.81 m³	43.3 %	15.91 m³	9766 kg	9683 kg
Level	Temperature	Free Volume	FB Level	Used Volume	
104.00 mm	21.00 °C	104 l	91.3 %	1096 l	
0.12 m	21.10 °C	82 l	99.2 %	9918 l	
Level	Temperature	Free Volume	FB Level	Used Volume	
142 m	18.00 °C	2511 l	29.8 %	1489 l	
171 m	5.60 °C	4467 l	10.7 %	533 l	
234 m	13.40 °C	10256 l	-2.6 %	-256 l	
151 m	4.00 °C	7308 l	26.9 %	2692 l	
151 m	7.60 °C	3786 l	24.3 %	1294 l	
248 m	0.50 °C	9893 l	11 %	107 l	
130 m	6.00 °C	3212 l	67.9 %	6708 l	
178 m	6.60 °C	9884 l	1.2 %	116 l	
Level	Temperature	Free Volume	FB Level	Round Free Volume	Round FB Level
0.68 m	20.8 °C	907 l	54.7 %	272 l	72.8 %
178.00 mm	0.00 °C				
Volume Flow	Velocity	Mass Flow	Temperature		
0.00	0.00	0.00	0.00		

Kuva 29. Sensorilistanäkymän työversio 3 työpäivän jälkeen.

Perjantai 17.3.2023

Päivän tavoitteena olisi saada sensorilistanäkymään tieto sensorilaitteen hälytystilasta. Eli mikäli jokin mittausarvo on ylittänyt tai alittanut sille määritellyn hälytysraja-arvon, näkyisi kyseinen laite sensorilistanäkymässä punaisen sävyllä korostettuna. Mikäli aikaa jäisi, koittaisin myös lisätä käyttäjää varten toiminallisuuksia, kuten jonkinlaisen suodatus- tai järjestelytoiminnon listan arvoille.

Päivän Scrum-palaverissa esittelin sensorilistanäkymää tiimille. Tiiminvetäjä sekä tuotteen myyntipäällikkö olivat sitä mieltä, että sensorilistanäkymä on jo siinä vaiheessa, että sitä voitaisiin esitellä tiettyjen asiakkaiden suuntaan. Tiiminvetäjä ohjeisti viimeistelemään näkymää ja päivän päätteeksi tekemään työhaarastani vetopyynnön (eng. pull request) kehityshaaraan (eng. development branch). Tämä siis tarkoittaa, että tekemäni koodi yhdistetään koko kehitystiimin käyttämään kehityshaaraan.

En onnistunut tämänkään päivän aikana saamaan sensorilistaan tietoa laitteen hälytystilasta ja aloin jo turhautua. Samoin kuin torstaina, päätin luovuttaa ja siirryin eteenpäin. Ohjelmoin käyttäjää varten eräänlaisen suodatustoiminnon, joka korostaa käyttäjän haluamat arvot koko sensorilistasta. Käyttäjä voi määritellä itse raja-arvon suuruuden ja valita, minkä mitattavan arvon nimen haluaa korostaa näkymästä. Kuvassa 30 olen korostanut kaikki yli ("Over") 33 % täyttöasteen ("Fill Level") sisältämät solut. Luomani toiminnon avulla käyttäjä voi siis täysin vapaasti valita minkä tahansa mittausarvon nimen, ja värjätä sensorilistanäkymästä kaikki valitun mittausarvon solut, jotka menevät käyttäjän määrittelemän raja-arvon yli tai ali.

Highlight Fill Level - Over - 33 X						
	Level	Free Volume	Fill Level	Used Volume	Remaining Feed	Silo Remaining Feed
	5.49 m	2167 m³	41.0 %	15.05 m³	9242 kg	8090 kg
	Level	Temperature	Free Volume	Fill Level	Used Volume	
	104.00 mm	21.00 °C	104 l	91.3 %	1096 l	
	0.12 m	21.80 °C	80 l	99.2 %	9920 l	
	Level	Temperature	Free Volume	Fill Level	Used Volume	
	1.42 m	9.30 °C	3506 l	29.9 %	1494 l	
	1.71 m	5.80 °C	4466 l	10.7 %	534 l	
	2.24 m	3.40 °C	10249 l	-2.5 %	-249 l	
	1.50 m	4.10 °C	7240 l	27.6 %	2760 l	
	1.51 m	4.80 °C	3789 l	24.2 %	1211 l	
	2.48 m	4.50 °C	9891 l	1.1 %	109 l	
	1.10 m	5.10 °C	3192 l	68.1 %	6808 l	
	1.78 m	4.00 °C	9888 l	1.1 %	112 l	
	Level	Temperature	Free Volume	Fill Level	Round Free Volume	Round Fill Level
	0.68 m	20.8 °C	907 l	54.7 %	272 l	72.8 %
	177.00 mm	0.00 °C				

Kuva 30. Sensorilistanäkymä, johon on lisätty käyttäjää varten solujen korostustoiminto.

En onnistunut tavoitteessa saada sensorilistanäkymään tietoa laitteen aktiivisesta hälytystilasta, sillä en edelleenkään löytänyt syytä, että miksi tieto hälytyksestä ei päivittynyt Reduxin tilahallintaan. Sensorilistanäkymä on kuitenkin hyvällä mallilla ja esiteltävissä myös asiakkaiden suuntaan, joten olen silti tyytyväinen työpäivään ja koko työviikkoon.

Viikkoanalyysi 5

Tämän viikon tehtävä kuvastaa mielestäni hyvin sitä, että ketterässä kehityksessä voidaan nopealakin aikataululla priorisoida erilaisten ominaisuuksien kehittämistä. Eli jos kertaamme tapahtumia, niin yhden asiakkaan suunnalta oli tullut ehdotus listanäkymästä sensorilaitteille. Koska tiimimme koki listanäkymän palvelevan kaikkia sensorilaitteita omaavia asiakkaita ja sopivan sovellukseen, päätimme toteuttaa sen. Minulta kului yksi työviikko rakentaa listanäkymän ensimmäinen käyttökel-poinen versio. Eli asiakkaan näkökulmasta ajateltuna heidän ehdotuksensa jälkeen kului ainoas-taan viikko aikaa siihen, että heille esitellään toimivaa demoa toivotusta ominaisuudesta.

Omasta mielestä ketterässä kehityksessä haasteita luovat ihmisten kehittämät omat säännöt, oh-jeet, toimintatavat ja ylimääräiset asiat, jotka eivät itsessään mielestäni liity ketterään kehitykseen. Tarkoitan tällä esimerkiksi sitä, että pakotetaan tiimi käyttämään esimerkiksi tietynlaisia työkaluja kuten Kanban (=työtehtävien pilkkominen paperilapuille), tai pitämään tarkasti ajoitettuja ja pituu-deltaan rajoitettuja palavereja. En ole ollut alalla pitkään, mutta minulle on syntynyt sellainen mieli-kuva, että joillekin henkilöille ketterä kehitys tarkoittaa nimenomaan tietynlaisten työkalujen käyttöä ja tarkasti määriteltyjä sääntöjä. En tarkoita, etteikö näistä voi olla apua, mutta itse tykkään pitää asiat yksinkertaisina. Minä koen ketterän kehityksen tarkoittavan mukautumiskykyä ja rohkeaa ite-raatiota. Pidän myös ketterän kehityksen manifestista, joka mielestäni tiivistää hyvin ketterän kehi-tyksen ajatusta neljään kohtaan. Ketterän kehityksen manifestin neljä kohtaa ovat:

1. Yksilöt ja vuorovaikutus ovat tärkeämpiä, kuin prosessit ja työkalut
2. Toimiva sovellus on tärkeämpi, kuin kattava dokumentaatio
3. Asiakasyhteistyö on tärkeämpää, kuin sopimusneuvottelut
4. Muutoksiin mukautuminen on tärkeämpää, kuin suunnitelmien noudattaminen (Beck ynnä muut 2001).

Pohditaan tiimimme toimintaa ja verrataan sitä manifestin kohtiin. Alkuperäinen suunnitelma oli, että kehitän energiaraporttinäkymää ja graafista käyttöliittymänäkymää, mutta mukauduimme tilan-teessa ja priorisoimme uuden sensorilistanäkymän niiden sijaan (manifesti kohta 4). Koska halu-taan melko nopeasti jotain toimivaa, priorisoin kehitystyössä ominaisuuden toimivaksi saattamisen sen dokumentaation sijaan (manifesti kohta 2). Manifestin kohdan 3 voisi ajatella myös toteutu-neen, sillä ehdotus listanäkymästä tuli alun perin asiakkaan suunnalta. Kohta 1 toteutui myös siinä

mielessä, että minulla oli vapaus suunnitella ja toteuttaa listanäkymä haluamallani tavalla hyödyntäen haluamiani työkaluja.

3.6 Seurantaviikko 6

Maanantai 20.3.2023

Päivän pääasiallinen agenda oli pitkä yksikkötestauspalaveri ulkoisen ohjelmistokehitysyrityksen kanssa, joka on palkattu suunnittelemaan ja tekemään yksikkötestausta sovellukseemme. Kävimme heille läpi kehittämämme kokonaisuuden toteutusta alkaen tiedonkeruusta erilaisilla laitteilla ja antureilla teollisuuden eri ympäristöissä ja päättyen lopulta sovelluksen front endin ja pilven teknologioihin ja hierarkiaan. Tiiminvetäjämme selittäessä kokonaisuuden auki ulkopuolisille sai minut muistamaan, kuinka monimutkaisesta kokonaisuudesta on kyse tässä meidän IIoT-reunalaskentaa hyödyntävässä digitaalisessa alustassa. Reunalaskenta (eng. edge computing) tarkoittaa datan prosessointia lähellä datan lähdettä (Microsoft 2023). IoT tulee sanoista 'Internet of Things' ja tarkoittaa laitteita ja koneita, jotka keräävät dataa ja ovat liitettynä verkkoon (Microsoft 2023b). IIoT tulee sanoista "Industrial Internet of Things", ja tarkoittaa käytännössä samaa kuin IoT, mutta teollisuuden ympäristössä.

Tiistai 21.3.2023

Päivän tavoitteena olisi edetä viikolla 4 työstämässäni energiaraporttinäkymässä, sillä viikolla 5 työstämäni sensorilistanäkymä oli riittävän hyvällä mallilla toistaiseksi. Viikon mittainen tauko energiaraporttinäkymästä aiheutti sen, että ajatukseni olivat edelleen täysin sensorilistanäkymän koodissa. Liiallinen tehtävien välillä hyppiminen voi mielestäni johtaa siihen, että tehtävissä eteneminen hidastuu turhankin paljon. Toisaalta itse koen myös mukavaksi välillä saada vaihtelua varsinkin silloin, jos on todella pitkään työstänyt yhtä ominaisuutta.

Seurantaviikko 4 kuvissa 27 ja 28 näkyy, mihin vaiheeseen olin energiaraporttinäkymässä jäänyt. Raporttinäkymään oli toivottu myös jonkinlaista ympyräkaaviota (eng. pie chart), joka kuvaisi selkeästi, mitkä laitteet ovat kuluttaneet eniten energiaa ja missä suhteessa ne ovat kuluttaneet energiaa laitteiden kokonaiskulutukseen nähden. Aloin etsiä vaihtoehtoja ympyräkaavion toteuttamiseen ja pohdin mahdollisuutta luoda ympyräkaaviokoodi itse alusta asti.

Pohdinnan jälkeen päätin, että kokeilen aluksi käyttää Recharts-kirjastoa, jota on jo käytetty sovelluksessa. Tämän jälkeen kokeilen hahmotella ympyräkaavion tekemistä itse. Recharts-kirjasto on siis luotu datan visualisointia varten erilaisiin kaavioihin React-pohjaisessa sovelluksessa (Recharts Group 2023). Sain molemmilla keinoilla luotua hyvin perinteisen ympyräkaavion energiaraporttiin.

porttinäkymään. Päästyäni tähän vaiheeseen päätin jättää ympyräkaavion tekemisen täysin ja jatkaa energiaraportin taulukon parissa. Ajattelin, että on parempi saada heatmap-tyylinen taulukkonäkymä ensin valmiimmaksi ja vasta tämän jälkeen alkaa miettiä, miten rakennan ympyräkaavion raporttinäkymään mukaan.

Päivän aikana yhdistin energiaraporttinäkymän rajapintaan, josta vielä toistaiseksi sai haettua vain puutteellisesti dataa. Tämä oli silti hyödyllistä, sillä rajapinnan kautta tullut data ei täysin vastannut muodoltaan kollegani minulle antamaa testidataa, joka on nähtävissä seurantaviikko 4 kuvassa 17. Tämä tarkoitti, että minun täytyy muuttaa koodiani, jolla käyn dataa läpi front endin puolella.

Hahmottelin myös energiaraporttinäkymän komponenttirakennetta järkevämmäksi, sillä viikolla 4 koodaillessani lähes kaikki logiikka ja energiaraporttikoodi oli yhden tiedoston eli yhden komponentin alla. Reactissa ja funktionaalisessa koodauksessa yleisestikin olisi syytä pitää funktiot pieninä. Mikäli funktion koko kasvaa koodillisesti, on funktiota todennäköisesti syytä pilkkoa useampiin pienempiin funktioihin (Meta Open Source 2023).

Keskiviikko 22.3.2023

Pohdin tapoja näyttää energiaraporttidataa selkeästi ja sitä, tulisiko ympyräkaavio samaan näkymään heatmap-tilukon kanssa, vai olisiko syytä erotella nämä toisistaan. Raportteja voi myös hakea useammalta kuukaudelta, joten täytyy myös keksiä hyvä tapa navigoida tarpeen tullen eri kuukausien välillä, tai sitten näyttää eri kuukausien raportit allekkain. Koitin visualisoida mielessäni ympyräkaavion ja heatmap-tilukon rinnakkain asettelua, ja en saanut mielessäni tätä sommittelua toimimaan, ellei ympyräkaavio olisi hyvin pieni. Halusin ainakin nyt aluksi erotella ne toisistaan. Aloin hahmotella ensimmäistä versiota energiaraportin näkymien välisestä navigoinnista.

Kokeilin React-Bootstrap-kirjaston välilehti (eng. tabs) -komponenttia. Välilehtikomponentilla voi luoda vaivattomasti välilehtinavigoinnin käyttöliittymään (React-Bootstrap c). Oletustyyli ei sopi lainkaan käyttöliittymään, joten muokkasin värimaailman käyttöliittymän teeman mukaiseksi. Kuvassa 31 on esimerkki, miltä välilehtikomponentin käyttö näyttää koodissa. Kuvan 32 yläosassa näkyy, miltä välilehtikomponentilla rakennettu välilehtinavigaatio näyttää itse käyttöliittymässä.

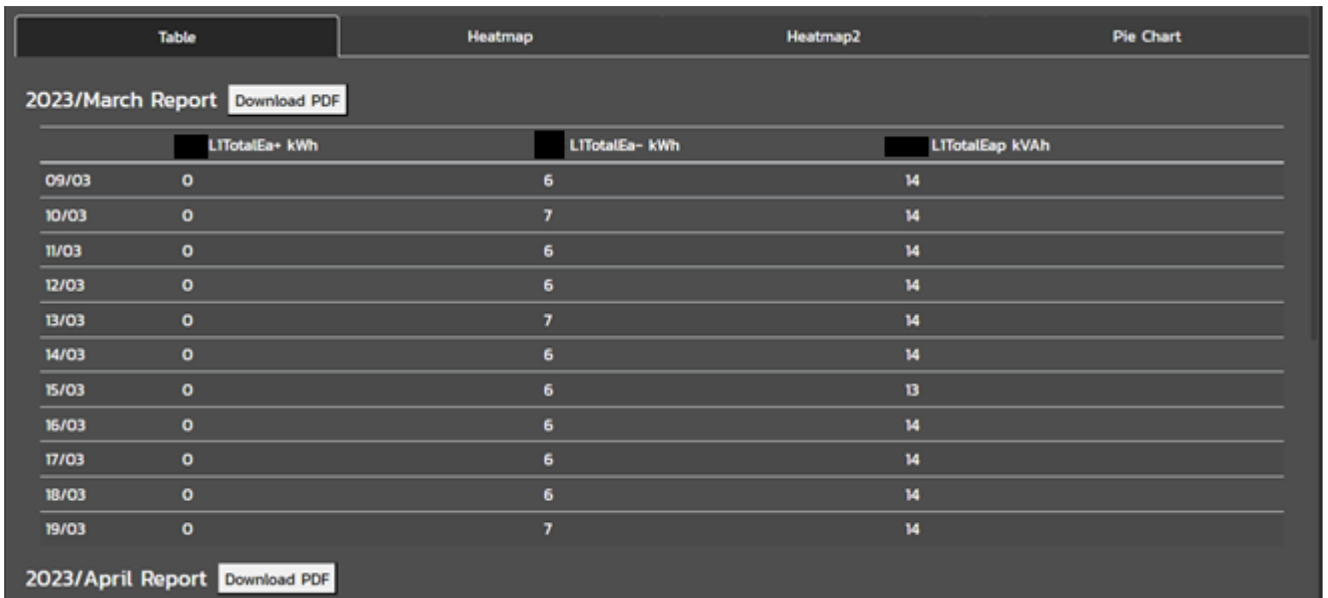
En ollut riittävän tyytyväinen viikolla 4 tekemääni heatmap-tilukkokomponenttiin, joka on nähtävillä kuvassa 28. Päätin kokeilla rakentaa tilukon uudestaan alusta alkaen ja pitää asiat yksinkertaisena. Kun hyvännäköinen tilukko olisi luotu, voisin tämän jälkeen koittaa lisätä siihen heatmap-ominaisuudet, eli solujen värjäämisen niiden sisältämien arvojen perusteella. Kuvassa 32 on näkyvillä React-Bootstrap-kirjaston tilukkokomponenttia (eng. table) hyödyntävä hyvin perinteisesti rakennettu tilukkonäkymä.

```

<Container className="my-2 energies-report" fluid>
  <Tabs
    defaultActiveKey="Table"
    className="mb-3"
    justify
  >
    <Tab eventKey="Table" title="Table" >
      <EnergiesTable />
    </Tab>
    <Tab eventKey="Heatmap" title="Heatmap">
      <EnergiesHeatmap
        reportData={reportData}
        heatMapColor={heatMapColor}
        highlightValues={highlightValues}
      ></EnergiesHeatmap>
    </Tab>
    <Tab eventKey="Heatmap2" title="Heatmap2">
      <EnergiesHeatmap2 />
    </Tab>
    <Tab eventKey="PieChart" title="Pie Chart">
      <EnergiesPieChart />
    </Tab>
  </Tabs>
</Container>

```

Kuva 31. React-Bootstrap-kirjaston välilehtikomponentin käyttöä.



2023/March Report		2023/April Report	
	LITotalEa+ kWh	LITotalEa- kWh	LITotalEap kVAh
09/03	0	6	14
10/03	0	7	14
11/03	0	6	14
12/03	0	6	14
13/03	0	7	14
14/03	0	6	14
15/03	0	6	13
16/03	0	6	14
17/03	0	6	14
18/03	0	6	14
19/03	0	7	14

Kuva 32. Energiaraportin työversio, jossa on toteutettu välilehtinavigointi erilaisten raporttinäkymien välillä.

Torstai 23.3.2023

Tarkoitukseni oli jatkaa raporttinäkymän työstämistä, mutta eksyin Pluralsight -nimiselle sivustolle, joka tarjoaa informaatioteknologia-alan opintomateriaalia ja kursseja (Pluralsight 2023). Kehitystii-
mimme on saanut mahdollisuuden hyödyntää Pluralsightin palveluja ja kursseja, joten käytin ai-
kaani selaillessa läpi minua mahdollisesti kiinnostavia kursseja. Mielestäni on hyvä ja tärkeää, että
työpaikka koittaa kannustamisen lisäksi myös konkreettisesti tarjota mahdollisuuksia itsensä kehit-
tämiseen ja oppimiseen.

Kurssivideoita katsoessa muistin, että raporttinäkymään oli joskus kaivattu ominaisuutta lähettää
raportti sähköpostiin järkevässä muodossa. Lähdin etsimään keinoja tämän toteuttamiseen. Tietoa
etsiessä hakusanoilla on erittäin iso merkitys. Esimerkiksi, kun hain tietoa hakusanoilla "export
table to PDF", löysin ohjeita erilaisiin ohjelmiin kuten Adobe Acrobat, Excel, Power BI, Word ja sii-
hen, kuinka voit tallentaa dokumentin tai taulukon PDF-muotoisena tietokoneellesi. Sen sijaan, kun
aloin hakea tietoa lauseilla kuten "export html to pdf with javascript" tai "html table to pdf in react",
aloin saada jo järkevämpiä hakutuloksia.

Löysin useitakin keinoja ja esimerkkejä, kuinka saada taulukko koodista tai käyttöliittymästä PDF-
dokumentin sisälle. Yksi tietty kirjasto alkoi esiintyä kuitenkin useissa lähteissä ja ratkaisuisa. Esi-
merkiksi nimimerkki Techsolutionstuff mielestä "jsPDF"-kirjasto on yksi parhaista kirjastoista muut-
tamaan HTML-koodia tekstiksi PDF-dokumentin sisään JavaScriptin avulla (Techsolutionstuff
2022). Karakaya (2022) ohjeistaa myös blogissaan, kuinka React-projektissa voidaan "jsPDF"-kir-
jastolla tuoda kuvaa ja tekstiä PDF-dokumenttiin.

Päätin kokeilla kirjaston käyttämistä käytännössä. Loin uuden kehityshaaran tätä kokeilua varten ja
asensin tässä haarassa projektiin jsPDF-kirjaston komennolla "npm install jsPDF". Hetken koodailtu-
ani päätin jälleen koittaa hyödyntää ChatGPT-tekoälyä ja muotoilin sille ohjeet tarkalleen seuraa-
vasti: "I have a table of data in a website and I want to have a button that will export the contents of
the table to a PDF document. I'm using React and jsPDF external library." ChatGPT-tekoäly antoi
5-kohtaiset ohjeet ja olennaisin osa oli kohta 3, jossa oli pääasiallinen koodi pyyntöni toteuttami-
seen. ChatGPT-tekoälyn antamien ohjeiden olennaisin osa, eli kohta 3, on nähtävissä kuvassa 33.

Koodi toimi itse asiassa yllättävän hyvin, eikä exportToDPF()-funktio itsessään ollut kovinkaan han-
kala ymmärtää. Koodia voisi kyllä parantaa, sillä mielestäni tapa, jossa ChatGPT-tekoäly työntää
sekä "headers"-arvot, että "rowData"-arvot samaan "data"-taulukkoon (array) on epäselvä. Tämä
johtaa siihen, että ChatGPT-tekoäly käyttäessään jsPDF-olion autoTable()-metodia, joudutaan
slice()-metodilla hakemaan oikeat arvot "data"-taulukosta (array). Tuo näyttää ainakin omaan sil-

mään pieneltä hirtitykseltä. Omasta mielestä paljon järkevämpi tapa olisi luoda kaksi taulukkoa (array) tuon yhden datataulukon (array) sijaan. Kahden taulukon (array) nimet voisivat olla esimerkiksi "headers" -taulukko (array) ja "rows" -taulukko (array). Tämän jälkeen voisimme antaa autoTable()-metodille "head" ja "body" arvot selkeästi ilman tuota slice() -metodilla kikkailua.

3. Define a function to handle the export to PDF, which will create a new jsPDF instance, iterate through the rows of the table, and add each row to the PDF document:

```
javascript Copy code

const exportToPDF = () => {
  const doc = new jsPDF();
  const table = document.getElementById('my-table'); // Replace with your table ID
  const rows = table.querySelectorAll('tr');
  const headers = table.querySelectorAll('th');
  const data = [];

  // Add headers to data array
  headers.forEach(header => {
    data.push(header.innerText);
  });

  // Add rows to data array
  rows.forEach(row => {
    const rowData = [];
    row.querySelectorAll('td').forEach(cell => {
      rowData.push(cell.innerText);
    });
    data.push(rowData);
  });

  // Add data to PDF document
  doc.autoTable({
    head: [data.slice(0, headers.length)],
    body: data.slice(headers.length),
  });

  // Save PDF document
  doc.save('table.pdf');
};
```

Kuva 33. ChatGPT-tekoälyn kirjoittama funktio, jolla taulukko tuodaan koodista PDF-dokumenttiin.

Perjantai 24.3.2023

Esittelin palaverissa kehitystiimille kuvan 32 näkymää ja ajatuksiani energiaraporttinäkymään liittyen. Keskustelimme yhdessä, mitä asioita kannattaa lähteä jatkojalostamaan ja mitä ei. Välilehti-navigointi saattaa olla turha, sillä se luo lisää monimutkaisuutta käyttöliittymään. Energiaraportin olisi hyvä olla sellainen, että kaikki olennainen mahtuu yhteen näkymään. Energiaraportin ollessa suuri voidaan tarvittaessa vierittää (eng. scroll) raporttia ylös ja alas.

Päätimme, että välilehdet voisivat pysyä toistaiseksi energiaraportissa. Ensimmäinen välilehti olisi energiaraportin "Overview", jossa yhdistetään sekä taulukkonäkymä, että pieni ympyräkaavio energian kulutuksesta. Toinen välilehti olisi "Table", jossa on vain taulukko. Kolmas välilehti olisi "Pie Chart", jossa ympyräkaavio olisi suurempi ja selkeämpi ja se voisi sisältää myös erilaisia toimintoja, jolla ympyräkaavion arvoja tai ulkonäköä voitaisiin muuttaa. Tärkeintä olisi kuitenkin saada nyt energiaraporttinäkymän ensimmäinen viimeistellyn oloinen "Overview"-välilehti valmiiksi. Kun ensimmäinen versio viedään tuotantoon, saamme käyttäjäpalautteen myötä paremmin ymmärrystä, onko erilaisille näkymille edes tarvetta tai kannattaako niihin tässä kohtaa käyttää enempää aikaa.

Viikkoanalyysi 6

Tällä viikolla en saanut mitään toiminnallisuutta valmiiksi asti, vaan iteroin ja kehitin energiaraporttinäkymän koodia eteenpäin. Uutena asiana nousi esiin erilaiset keinot tallentaa näkymiä sovelluksesta käyttäjän tietokoneelle PDF-dokumenttina. Sain hyvinkin nopeasti luotua logiikan tälle toiminnolle kiitos ChatGPT-tekoälyn. Vaikka kuvan 33 koodi toimii vain yksinkertaisille taulukoille, jotka on rakennettu käyttäen `<tr>` ja `<th>` -elementtejä, on sitä helppo lähteä kehittämään paremmin omaan tarpeeseen soveltuvaksi. Koodi vaatii hiomista muun muassa siinä, että "data" -taulukon (array) ei sisällytetä sekä taulukon otsikko- että soluarvoja.

ChatGPT-tekoälyä joitakin viikkoja hyödyntäneenä voin omasta kokemuksesta sanoa, että se on mielestäni erittäin voimakas työkalu, joka tehostaa sovelluskehittäjän tuottavuutta. Sen sijaan, että joudut itse etsimään dokumentaatiosta tai erilaisista lähteistä vastauksia kysymyksiisi, ChatGPT-tekoäly antaa sinulle nopeasti kattavan vastauksen kysymykseesi. Tarkoitin kattavalla vastauksella sitä, että ChatGPT-tekoäly usein selittää auki kirjoittamansa koodin. Esimerkiksi, jos löydän Stackoverflow-verkkosivuilta vastauksen johonkin kysymykseeni, sisältää vastaus hyvin usein ai-noastaan pätkän toimivaa koodia. Eli erona on, että ChatGPT-tekoäly kertoo, mitä koodissa tapahtuu ja tarvittaessa myös, miksi koodi kirjoitettu tällä tavalla. Oikein käytettynä et siis vain kopioi koodia sokeasti sitä ymmärtämättä, vaan opit erilaisia tapoja ratkaista ongelmiasi ja opit ymmärtämään logiikan näiden ratkaisujen takana. Tämä on siis oma kokemukseni.

Aiheesta on lyhyessä aikavälillä ilmaantunut todella paljon materiaaleja niin blogikirjoitusten kuin YouTube-videoidenkin muodossa. ”Web Dev Cody” -nimisellä YouTube-kanavalla sovelluskehittäjä Cody Seibert esittelee ChatGPT-tekoälyn vahvuuksia uusien asioiden oppimisessa. Hän kertoo, että ei ole juurikaan ohjelmoinut käyttäen ”Go” -kieltä ja kysyy ChatGPT-tekoälyltä, kuinka toistorkenteet toimivat ”Go” -kielessä. Videolta voi katsoa, että ChatGPT-tekoäly antaa selkeän vastauksen ja vastailee myös Cody Seibertin esittämiin jatkokysymyksiin. (Seibert joulukuu 2022, 14:25 – 16:25).

Suosittu opetuskanava ”Programming with Mosh” esittelee videollaan ”ChatGPT Tutorial for Developers - 38 Ways to 10x Your Productivity” ChatGPT-tekoälyn hyötyjä. Mosh Hamedani kysyy muun muassa, mitkä ovat 3 parasta kirjaa opetella Java-kieltä. ChatGPT-tekoäly antaa vastauksen ja tämän jälkeen Hamedani pyytää tekoälyä tiivistämään ensimmäisen kirjan tärkeimmät ydinasiat. ChatGPT-tekoäly antaa kattavat vastaukset Hamedanin kysymyksiin. (Hamedani tammikuu 2023, 5:06 – 5:45.) Hamedani on huomannut myös saman kuin minä, eli sen, että ChatGPT-tekoäly selittää auki käyttämänsä metodit ja logiikan tarjotessaan koodia (Hamedani tammikuu 2023, 6:29 – 7:05).

ChatGPT-tekoäly ei anna aina parhaita vastauksia ja ei myöskään anna aina oikeita vastuksia, joten käyttäjän on silti luettava ChatGPT-tekoälyn vastaukset läpi ajatuksella. Oman kokemukseni mukaan ainakin toistaiseksi sitä on syytä käyttää vielä vain työkaluna, apuna ja inspiraatiota, eikä käyttäjän tule sokeasti luottaa sen tuottamiin vastauksiin ilman omaa ajattelua.

3.7 Seurantaviikko 7

Maanantai 27.3.2023

Edellisen viikon perjantain aikana ja sen jälkeen olin tullut energiaraporttinäkymän suhteen siihen tulokseen, että välilehtinavigointi oli huono idea ja turha ominaisuus. Mielestäni käyttäjän kannalta selkeämpi vaihtoehto on tehdä vain yksi hyvä näkymä, ja tätä näkymää voisi mahdollisesti manipuloida muutamalla painikkeella.

Poistin välilehtinavigoinnin ja lisäsin tilalle 2 painiketta. Ensimmäinen painike piilottaa ympyräkaavion ja laajentaa taulukkonäkymää. Toinen painike manipuloi taulukon heatmap-toimintoja korostaen joko pieniä arvoja, suuria arvoja tai piilottaen arvojen korostuksen kokonaan tehden taulukosta tavallisen näköisen. Lisäsin myös toiminallisuuden, jossa käyttöliittymä lähettää get-rapapintapyyntöä palvelimelle välittömästi käyttäjän avatessa ensimmäistä kertaa energiaraporttinäkymän auki. Tämän jälkeen ruudulle ilmestyy automaattisesti viimeisin energiaraportti käyttäjän nähtävälle.

Autoin myös kollegaani, joka oli piirtämässä graafista käyttöliittymäosuutta varten eräälle laiteyksikölle päivitettyä SVG-kuvaa. Opastin piirto-ohjelman käytössä sekä selitin osiltaan auki sitä logiikkaa, jota front end käyttää SVG-kuvan manipuloimiseen. Esimerkiksi front end hakee kaikki SVG-kuvassa olevat linjat etsimällä SVG-kuvasta `<g>` elementin, jonka `"id"`-ominaisuuden arvo on `"lines"`, ja käymällä toistorakenteella tämän läpi etsien jokaisen `<path>` elementin talteen. Tämän kaltaisen logiikan vuoksi on siis tärkeää, että SVG-kuvan elementit ovat nimetty ja jäsennelty tietyllä tapaa.

Tiistai 28.3.2023

Silmäillessäni energiaraporttidataa (viikko 4 kuva 17) aloin pohtimaan, kuinka monelta mittauspisteeltä (kuva 17 datassa `"Hwld"` edustaa mittauspistettä) energiadataa mahdollisesti tulee todellisuudessa. Jos jokainen mittauspiste sisältää 3 mittausarvoa, eli `"L1TotalEa-"`, `L1TotalEa+"` ja `"L1TotalEap"`, niin esimerkiksi 10 mittauspistettä toisi taulukkoon jo 30 sarakkeen edestä arvoja. Tällainen määrä sarakkeita taulukossa täytyy ottaa käyttöliittymää tehdessä huomioon, sillä ne eivät mahdu käyttöliittymään sellaisenaan, vaan tarvitaan esimerkiksi jonkinlaisen vieritys (scroll) -toiminallisuus.

Lisäksi mittauspisteiltä tulee dataa eri mittayksiköillä: kilowattitunteina (kWh) sekä kilovolttiampeeritunteina (kVAh), joita ei voi suoraan verrata keskenään. Tällä on merkitystä ympyräkaavion kannalta. Ohjeistukseni oli ollut asettaa jokaisen mittauspisteen energiankulutus ympyräkaavioon, mutta halutaanko kulutus näyttää kVAh vai kWh mukaan?

Kysyin tarkennusta energiaraportin tekemistä varten ja sain tiedon, että taulukkoon voisi tulla hyvin paljon arvoja. Tämä tarkoittaa, että taulukko tarvitsee melko varmasti käytettävyyden nimissä jonkinlaisen suodatustoiminnon, jolla käyttäjä voi tarvittaessa piilottaa sarakkeita, joita ei halua nähdä. Lisäksi olisi hyvä, jos käyttäjä voisi valita ympyräkaaviossa näytettävät arvot mittausarvon nimen mukaan itse.

Koska rajapinnasta saatava testidata oli hyvin rajallinen ja sisälsi vain yhden mittauspisteen arvoja, päätin luoda omaa testidataa, joka kuvastaisi paremmin tilannetta, jossa mittauspisteitä on useita. Lisäksi aloin ohjelmoida taulukolle suodatustoimintoa (eng. filter). Päätin lähteä yksinkertaisesta suodatustoiminnosta liikkeelle, jossa voidaan valita mikä tahansa mittausarvon nimi, jonka jälkeen vain tämän mittausarvonimen data tulostuu taulukkoon.

Kuvassa 34 on suodatuspainikkeen pääasiallinen logiikka. Eli `handleFilterChange()`-funktio vastaanottaa argumentin `"valueName"`, joka on käyttäjän valitsema arvo, jonka perusteella suodatus tapahtuu. Jos käyttäjä on valinnut näyttää kaikki arvot, ei suodatusta tapahdu. Muussa tapauk-

sessä käymme "reportData" läpi map()-metodilla, jonka jälkeen jokaisen raportin "Value"-ominaisuus käydään läpi ja sieltä otetaan talteen se objekti, jonka "ValueName" vastaa funktiolle syötettyä argumenttia "valueName", joka oli siis käyttäjän valitsema arvo, jonka perusteella suodatus halutaan tehdä. Eli jos käyttäjä valitsee suodatusarvoksi "L1TotalEa+", poistetaan taulukosta kaikki muut sarakkeet, eli poistetaan jokaisen mittauspisteen "L1TotalEa-" ja "L1TotalEap" -arvot näkyvistä.

```
const handleFilterChange = (valueName) => {
  if (valueName === "all") {
    setFilteredReportData(reportData);
  } else {
    const filteredData = reportData.map((report) => {
      const filteredValues = report.Value.filter((value) => value.ValueName === valueName);
      return { ...report, Value: filteredValues };
    });
    setFilteredReportData(filteredData);
  }
};
```

Kuva 34. Energiaraporttinäkymän suodatusfunktio.

Keskiviikko 29.3.2023

Törmäsin ongelmaan, että rakentamani energiaraporttitaulukko ei toiminutkaan oikein. Jos katsoimme viikko 4 kuva 27 taulukkoa, huomaamme, että jokaisen sarakkeen otsikkorivillä on eri arvot. Todellisuudessa otsikkorivillä voi olla paljonkin samoja arvoja ja mittayksiköitä, mutta mittauspisteen nimi vaihtelee. ChatGPT-tekoälyn avulla kirjoittama koodini ei osannut lisätä taulukon soluihin mittausarvoja oikein, mikäli "ValueName"-muuttujan arvo ei ollut uniikki jokaisella pystysarakkeella. Eli jos taulukoitava data sisältäisi useammalta mittauspisteeltä mittausdataa, ja kaikilla mittauspisteillä olisi saman nimisiä arvoja, kuten "L1TotalEa+", "L1TotalEa-" ja "L1TotalEap", ei taulukkoon tulisi kuin yhden mittauspisteen data näkyviin.

Uudelleenkirjoitin taulukon muodostamiseen käytettyjä funktioita ongelman korjaamiseksi. Muutamat funktiot olivat muutenkin mielestäni hyvin epäselviä, koska olin osiltaan vain kopioinut ChatGPT-tekoälyn koodia sitä juurikaan muuttamatta. Esimerkiksi kuvassa 33 (tai viikko 4 kuva 23) oleva koodipätkä, jossa populoidaan "rowData" niminen muuttuja, on mielestäni hyvin epämääräinen ja vaikeasti luettava. Kirjoitin tämän kokonaan uudestaan ja hyödynsin for-toistorakenteiden sisäkkäin sijoittelun sijaan map()-metodia, joka on itselleni mieluisempi käyttää ja lukea. Kuvassa 35 näkyy ensin vanhaa logiikkaa, jossa populoidaan muuttuja "rowData", ja sen perässä on luomani uusi selkeämpi generateTableRows()-funktio.

```
// Create new rows with the desired format
const rowData = [];
for (let i = 0; i < dates.length; i++) {
  const row = [];
  row.push({"value": dates[i], "percentage": "0.00"});
  for (let j = 0; j < heatMapData.length; j++) {
    let valueFound = false;
    for (let k = 0; k < heatMapData[j].Values.length; k++) {
      if (Object.keys(heatMapData[j].Values[k])[0] === dates[i]) {
        row.push({"value": heatMapData[j].Values[k][dates[i]], "percentage":
heatMapData[j].Values[k].percentage});
        valueFound = true;
        break;
      }
    }
    if (!valueFound) {
      row.push({"value": 0, "percentage": "0.00"});
    }
  }
  rowData.push(row);
}
```

! TÄMÄ UUSI FUNKTIO KORVAA YLLÄ OLEVAN KOODIN

```
function generateTableRows(dates, measurementsWithHeatmapValues) {
  // map through dates, every date should have a row of data.
  const tableRows = dates.map((date, i) => {
    const formattedDate = formatDate(date, reportType)
    const newRow = [
      // add the formatted date as the first element of the row
      { value: formattedDate, percentage: 0 },
      // then generate the row valueobjects after that
      ...measurementsWithHeatmapValues.map(entry => ({
        // get the value and percentage for the current date and index (i)
from the entry.Values object
        value: entry.Values[i][date],
        percentage: entry.Values[i].percentage
      })))
    ];
    // return new row (this adds it to tableRows)
    return newRow;
  });
  return tableRows
}
```

Kuva 35. Uusi generateTableRows()-funktio korvaa vanhan rowData-taulukon populointilogiikan.

Torstai 30.3.2023

Kollegani oli saanut ensimmäisen oman versionsa graafiseen käyttöliittymään käytettävästä SVG-kuvasta ja pyysi minua käymään kuvaa läpi. Kopioin kollegan luoman kuvan itselleni suunnitteluohjelman työpöydälle ja uudelleen nimesin lähes puolet kuvassa olevista elementeistä. Tämä siitä syystä, että halusin muuttaa elementtien nimeämistapaa selkeämmäksi. Lisäksi korjasin muutamia asiavirheen kuvasta. Esimerkiksi yksi kuvassa oleva linja oli vedetty väärin kahden laitteen välille.

Tämän jälkeen tallensin kuvan SVG-tiedostona koneelleni ja siirsin siitä kopion Azure Cloud Storageen, josta muun muassa käyttämämme demolaitteet hakevat tietynlaisia konfiguraatiodietoja käyttöönsä. Avasin demolaitteen käyttöliittymän ja totesin, että mikään luomani tyyllitely ei enää toimi, sillä olin juuri vaihtanut kuvan elementtien tunniste-arvoista (eng. id) puolet. Eli jouduin refaktorimaan myös CSS-tyylitiedostosta tunniste-arvot vastaamaan SVG-kuvan elementtien uusia tunniste-arvoja.

Koska kollegani luoma uusi kuva oli sommiteltu hieman eri lailla ja siinä olevat näytelinjat oli saatu piirrettyä vähemmällä määrällä <path> elementtejä, jouduin muokkaamaan CSS-tyylitiedostoa yllättävänkin paljon. Minulla meni kaiken kaikkiaan noin 4 tuntia pelkästään siihen, että olin ensiksi nimennyt itse SVG-kuvasta elementit uudestaan ja tämän jälkeen tehnyt vastaavat muutokset koodiin. Tämä oli kuitenkin tärkeä tehdä, sillä elementit olivat nyt nimetty selkeämmin ja CSS-tyylit olivat myös huomattavan paljon helpompi lukea tämän myötä.

Olin orientoitunut tämän päivän osalta jo niin vahvasti tämän graafisen käyttöliittymäosuuden kehittämiseen, että käytin loppupäivän sen parissa. Koitin hienosäätää kuvan animaatioita ja tyylejä ja kokeilla erilaisia keinoja tuoda käyttäjälle paremmin esiin, mitä kuvassa tapahtuu.

Perjantai 31.3.2023

Perjantain Scrum-palaverin aikana kävimme tiimin kesken läpi kollegani piirtämää kuvaa graafiseen käyttöliittymään. Tiiminvetäjä antoi siitä palautetta ja huomautti muutamasta asiavirheestä. Työstin tämän palaverin jälkeen koko loppu päivän kollegani kanssa yhdessä tätä kuvaa ja koitimme korjata asiavirheet ja muutenkin viimeistellä kuvaa. Kollegalleni on esimerkiksi vaikea nimetä kuvan elementtejä järkevästi, sillä minä olen kirjoittanut front endin logiikan, jolla kuvaa tulisi manipuloida. Eli on parempi, että minä nimeän kuvan elementit.

Yhdessä tekeminen meni hyvinkin paljon ajatusten ja ideoiden vaihtamiseksi: yhdessä luomiseksi. Koska teen työtehtäviäni hyvin paljon itsenäisesti ja yksin, on minulle melko harvinainen kokemus työstää parityöskentelynä jotain asiaa eteenpäin. Tämä oli antoisaa. Kun on jatkuvassa vuorovaikutuksessa toisen kanssa, tulee saaneeksi palautetta reaaliajassa ja sinulla on koko ajan joku, joka voi tarjota mielipiteensä ajatuksistasi ja ideoistasi.

Kuvan animaatioissa minulla ilmeni ongelma, jossa päällekkäisten elementtien animaatiot eivät toimineet, kuten pitäisi. Ohjaan kuvan animaatioiden tilaa hyödyntäen CSS-ominaisuutta "animation-play-state", jolle voidaan siis antaa joko arvo "paused" tai "running". Kuvassa näytelinjaa pitkin kulkee kaasua. Kaasujen liikkuminen on animoitu yksinkertaisesti piirtämällä linjojen sisään liikkuvia viivoja. Idea on sama, kuin Chris Coyierin ohjeessa "How SVG Line Animation Works" (Coyier 18.2.2014).

Eri kaasut on kuvattu eri väreillä ja vain yksi kaasu liikkuu linjoja pitkin kerrallaan. Kun esimerkiksi punainen kaasu liikkuu, pysäytän vihreän ja sinisen kaasun liikkeen kuvasta ja muutan niiden värit valkoiseksi, joka on linjojen oletusväri. Punaisen kaasun kulkiessa valkoisia linjoja pitkin se kohtaa valkoiseksi värjätyn pysäytetyn animoidun linjan, joka on prioriteetiltaan korkeammalla tasolla verkkosivun dokumenttiobjektimallin (eng. document object model) -hierarkiassa, jonka vuoksi punainen kaasun näyttää katoavan hetkeksi valkoisen värin alle, kunnes se ilmestyy valkoiseksi värjätyn pysäytetyn linjan alapuolelta taas esiin. Tämä toistuu useaan kertaan, sillä pysäytetyt animaatiot sisältävät kymmeniä valkoiseksi värjätty viivoja.

Ongelmaa ei voi korjata siirtämällä elementtejä dokumenttiobjektimallin sisällä eri tasoille, sillä päällekkäisistä `<path>` elementeistä aina joku on kuitenkin alimmalla prioriteetilla, eli aina jonkun värisen kaasun animaatio olisi viallinen. Kirjoitin CSS-koodiin aktiivisen animaation elementeille korkeammat "z-index"-ominaisuudet ja pysäytetyn animaation elementeille matalat "z-index"-ominaisuudet. Tämä ei toiminut, sillä "z-index"-ominaisuus ei toimi, kuin tietynlaisille elementeille, joihin SVG `<path>` -elementit eivät lukeudu (MDN Web Docs 2023h).

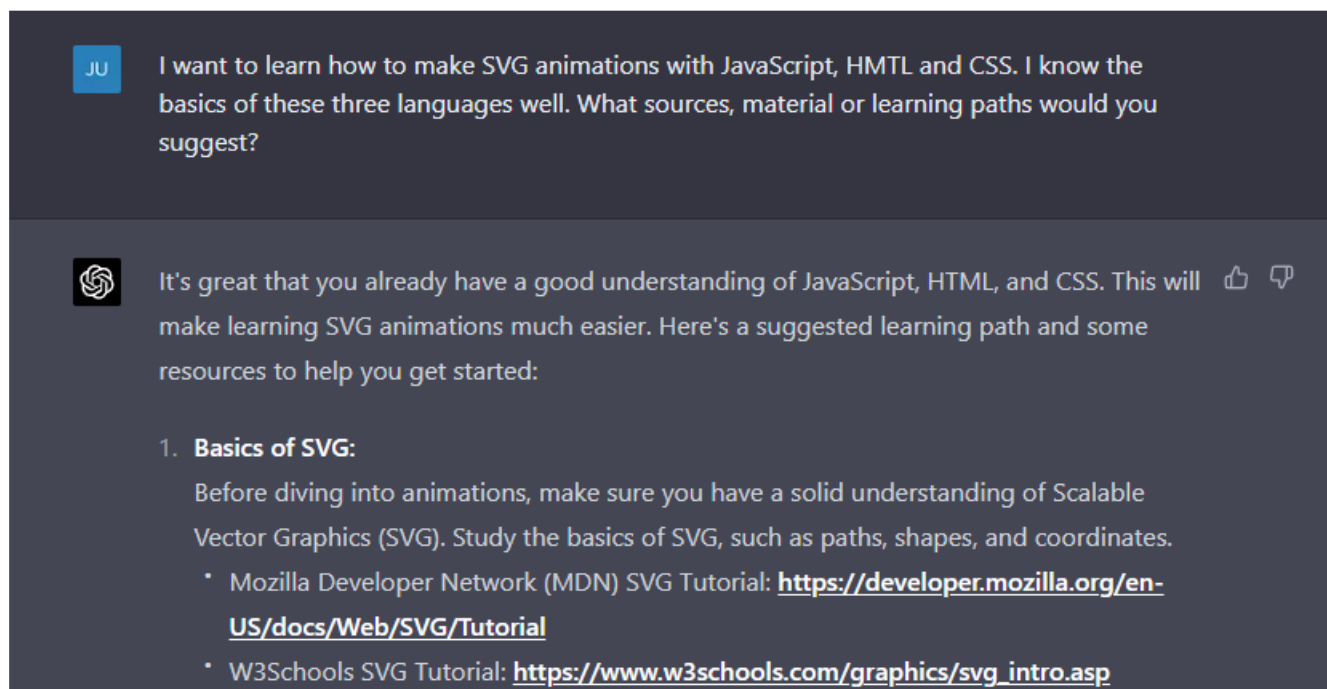
Lopulta keksin ratkaisun, jossa värjäsin pysäytetyt animaatioviivat läpinäkyviksi valkoisen sijaan. Tämän jälkeen punainen, sininen ja vihreä kaasut liikkuvat valkoisia linjoja pitkin sulavasti. Joskus yksinkertaistakin ratkaisua joutuu etsimään, jos ensimmäiset ajatuksesi ongelman ratkaisuun eivät toimineetkaan, mutta et vain osaa päästää niistä irti.

Viikkoanalyysi 7

Tarkoitukseni oli tällä viikolla edetä energiaraportin rakentamisessa ja saada kenties jopa raportinäkymän ensimmäinen versio valmiiksi. En onnistunut tässä, sillä kollegani tarvitsi apuani graafiseen käyttöliittymänäkymään sijoitettavan kuvan suhteen ja tähän kului paljon aikaa. Tulin hyppi-neeksi kahden eri tehtävän välillä, ja tämä vei huomiotani ja keskittymistäni pois energiaraportinäkymän kehittämisestä.

Tällä viikolla minulle tuli tunne, että ohjelmointitaitojeni kehitys on hidastunut. Tuntuu, että opin pikkuhiljaa pieniä uusia asioita front end -ohjelmointiin liittyen työn ohessa, mutta oppimisen tahti ei ole lähellekään samaa, kuin aktiivisesti opiskellessa ohjelmointia. Lisäksi graafista käyttöliittymäosuutta tehdessä olen kokenut, että vektorigrafiikan (SVG) luominen, manipulointi ja animointi JavaScriptin ja CSS avulla on minulla hyvin aloittelijan tasolla. Tämä toisaalta ei ole minulle yllätys. En ole aikaisemmin opiskellessani tai koodatessani ollut juurikaan tekemässä vektorigrafiikan kanssa, jos en laske mukaan erilaisten pienten kuvien (eng. icon) lisäämistä käyttöliittymään.

Kysyin ChatGPT-tekoälyltä, mitä hyviä materiaaleja, lähteitä ja keinoja löytyy vektorigrafiikan animoinnin ja manipuloinnin opiskeluun. Kuvassa 37 ChatGPT-tekoäly ehdottaa aloittamaan vektorigrafiikan perusteiden opettelusta ja tarjoaa 2 lähdettä opiskelun aloittamiseen. ChatGPT-tekoälyn tarjoamat "Mozilla Developer Network" and "W3Schools" ovat molemmat minulle tuttuja lähteitä, joita käytän muutenkin aktiivisesti.



Kuva 37. ChatGPT-tekoälyn vastaus vektorigrafiikan opiskeluun kohta 1.

Mozilla Developer Networkin opetusohjelma (eng. tutorial) vektorigrafiikkaan näyttää melko hyvältä tekstimuotoiselta materiaaalilta. Se käy läpi muun muassa vektorigrafiikan perusteet, - muodot (eng. shapes), - linjat (eng. paths), - tekstielementit ja - manipulaation (eng. transformations) (MDN Web Docs 2023j). Pelkän opetusohjelman läpi lukeminen ei tosin auta minua oppimaan kovinkaan hyvin, vaan minun täytyisi koodata itse jotain konkreettista oppiakseni. Jos kävisin tämän opetusohjelman läpi, avaisin itselleni koodieditorin auki ja kokeilisin tehdä erilaisia yksinkertaisia harjoituksia samalla, kun etenen materiaalissa eteenpäin. Olisi hienoa, jos opetusohjelmasta löytyisi valmiiksi luotuja yksinkertaisia tehtäviä materiaalin rinnalle, jotka haastaisivat lukijaa oppimaan.

Kuvassa 38 ChatGPT-tekoäly kehottaa opettelemaan CSS-kielen animaatioita ja siirtymiä (eng. transition). Molemmat manipuloivat kuvaa ja niillä on joitakin eroja. Animaatiolle on annettava aika-arvot (eng. keyframes), joiden avulla määritellään, miltä animaation kuuluu näyttää minäkin hetkenä (MDN Web Docs 2023j). Tyypillisesti siirtymiä käytetään yksinkertaisempaan kuvamanipulaatioon, kuten painikkeen värin vaihtamiseen kursorin ollessa sen päällä. Voisi siis ajatella, että siirtymäkomento käskee painikkeen siirtyä esimerkiksi vihreästä väristä punaiseen. Siirtymää ei voi

myöskään laittaa toistamaan itseään, toisin kuin animaatiota. Eli jos haluaisi painikkeen, joka vaihtaa väriä jatkuvasti, täytyisi se tällöin animoida.

Kuvassa 38 ChatGPT-tekoälyn kohta 3 ei mielestäni tarjoa hyvää materiaalia JavaScriptin hyödyntämisestä SVG-elementtien manipulointiin. Se tarjoaa jälleen linkkiä Mozilla Developer Networkin sivuille, mutta tällä kertaa linkin takana on vain pitkä lista käytettäviä metodeja, mutta ei kunnan esimerkkejä niiden käytöstä käytännössä. (MDN Web Docs 2023k.) Tekoäly tarjoaa myös linkkiä kolmannen osapuolen kirjastoon, mutta linkki ei ole oikeassa muodossa, vaan vie epämääräiselle pokerisivustolle.

2. CSS Animations and Transitions:

Learn how to create animations and transitions using CSS. This will help you understand how to animate SVG elements using CSS properties.

- MDN CSS Animations: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Animations
- MDN CSS Transitions: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Transitions
- CSS-Tricks Guide to CSS Animation: <https://css-tricks.com/css-animation-libraries/>

3. JavaScript and SVG:

Learn how to manipulate SVG elements with JavaScript to create more complex animations and interactivity.

- MDN Using SVG with JavaScript: https://developer.mozilla.org/en-US/docs/Web/API/SVG_API
- SVG.js: A lightweight library for manipulating and animating SVG: <https://svgjs.com/docs/3.0/>

Kuva 38. ChatGPT-tekoälyn vastaus vektorigrafiikan opiskeluun kohdat 2 ja 3.

Kuvassa 39 ChatGPT-tekoäly kehottaa tutustumaan kolmannen osapuolen kirjastoihin, joiden avulla voit luoda animaatioita helpommin. Ensimmäinen vaihtoehto "Greensock Animation Platform" ei ole ilmainen kaupalliseen käyttöön, vaan vaatii lisenssin ostamisen. Toinen vaihtoehto "Snap.svg" vaikuttaa ihan kelpoiselta, mutta dokumentaation demot eivät sisällä niiden lähdekoodia, joka on oppimisen kannalta huono asia. Lisäksi "Snap.svg"-kirjaston viimeisin päivitys on 6 vuotta sitten, joka on hälyttävä merkki siitä, että kirjasto alkaa olla vanhentunut. Tämä voi muodostua ongelmaksi varsinkin, mikäli sitä haluaisi käyttää modernin viitekehyksen, kuten Reactin kanssa. Myöskään kolmas kirjasto "VelocityJS" ei ole saanut päivitystä 5 vuoteen. Mielestäni on myös parempi opetella perusteita ilman kolmannen osapuolen kirjastoja. Jos myöhemmin käytät kirjastoa nopeuttaaksesi työskentelyäsi, sinulla on edes pieni ymmärrys siitä, miten kirjaston tarjoamat funktiot ja metodit toimivat niin sanotusti "konepellin alla".

4. SVG Animation Libraries:

Familiarize yourself with popular SVG animation libraries to simplify your work and create more advanced animations.

- * GreenSock Animation Platform (GSAP): <https://greensock.com/gsap/>
- * Snap.svg: <http://snapsvg.io/>
- * Velocity.js: <http://velocityjs.org/>

5. Project-based Learning:

Practice your skills by creating small projects or replicating existing SVG animations.

Websites like CodePen (<https://codepen.io/>) and Dribbble (<https://dribbble.com/>) can provide inspiration and help you learn from others.

Kuva 39. ChatGPT-tekoälyn vastaus vektorigrafiikan opiskeluun kohdat 4 ja 5.

Kuvassa 39 ChatGPT-tekoälyn kohta 5 "Project-based Learning" on mielestäni hyvä neuvo. Tekoäly ehdottaa hakemaan inspiraatiota sivustoilta kuten CodePen, joka on mielestäni hyvä ajatus. CodePenissä olevat projektit sisältävät niiden lähdekoodin, joten tiedät, miten projekti on rakennettu. Etsin CodePenistä hakusanalla "svg animation" erilaisia projekteja ja hakusana tuotti hyviä osumia. Voisin poimia jonkin projektin ja kokeilla esimerkiksi muokata sen sisältämien animaatioiden arvoja ja tutkia muutoksieni vaikutuksia. Tämän jälkeen voisin koittaa lisätä kuvaan tai animaatioon jonkin täysin uuden elementin ja koittaa luoda sille oman animaation hyödyntäen projektissa jo valmiiksi olevia animaatioita. Tämän kaltainen tutkiskelu on opettavaista ja auttaa soveltamaan opittuja perusteita käytännössä.

Kuva 40 on viimeinen osa ChatGPT-tekoälyn vastauksesta siihen, mikä olisi hyvä reitti opetella vektorigrafiikan manipulointia JavaScriptin avulla. ChatGPT-tekoäly kehottaa lukemaan aiheesta lisää muun muassa foorumeilta, blogeista tai kirjoista. Kokeilin etsiä mielenkiintoista materiaalia ChatGPT-tekoälyn tarjoaman ensimmäisen linkin takaa ja olin yllättynyt siitä, miten paljon mielenkiintoista luettavaa "Smashing Magazine" -sivuston artikkelit tarjosivatkaan. Löysin muun muassa artikkelin "Magical SVG Techniques", joka sisälsi todella näyttäviä ja hienoja vektorigrafiikkakikkoja ja -miniprojekteja lähdekoodineen (Cosima 10.5.2022).

6. Further Reading and Resources:

Expand your knowledge by following blogs, participating in forums, and reading books on SVG and web animations.

- Smashing Magazine: <https://www.smashingmagazine.com/>
- CSS-Tricks: <https://css-tricks.com/>
- "SVG Animations: From Common UX Implementations to Complex Responsive Animation" by Sarah Drasner: <https://www.amazon.com/SVG-Animations-Implementations-Responsive-Animation/dp/1491939702>

Following this learning path and using these resources should provide you with a strong foundation in SVG animations using JavaScript, HTML, and CSS. Good luck!

Kuva 40. ChatGPT-tekoälyn vastaus vektorigrafiikan opiskeluun kohta 6.

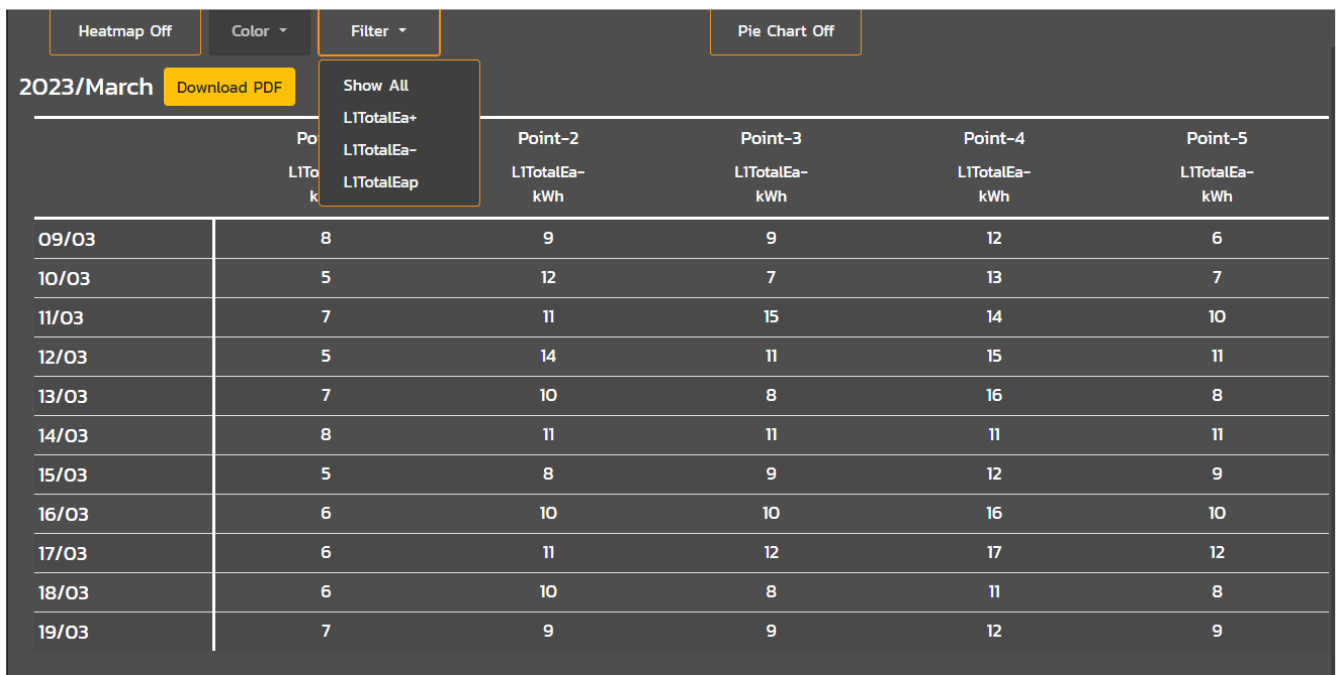
Mielestäni ChatGPT-tekoälyn ohjeet vektorigrafiikan manipulointitaitojen kehittämiseksi olivat kaiken kaikkiaan hyvät. Jos noudattaisin ohjeita ja kävisin listan kohta kohdalta läpi ajatuksella ja käyttäen aikaa, oppisin varmasti paljon lisää uutta ja tämä auttaisi taistelemaan sitä tunnetta vastaan, että oppimisentahti on minulla hidastunut. Mikäli haluaisin kehittää React osaamistani, voisin kysyä ChatGPT-tekoälyltä vastaavanlaista oppimisen polkua esimerkiksi kysymyksellä "I want to learn more about intermediate and advanced React programming. What sources, material or learning paths would you suggest?"

3.8 Seurantaviikko 8

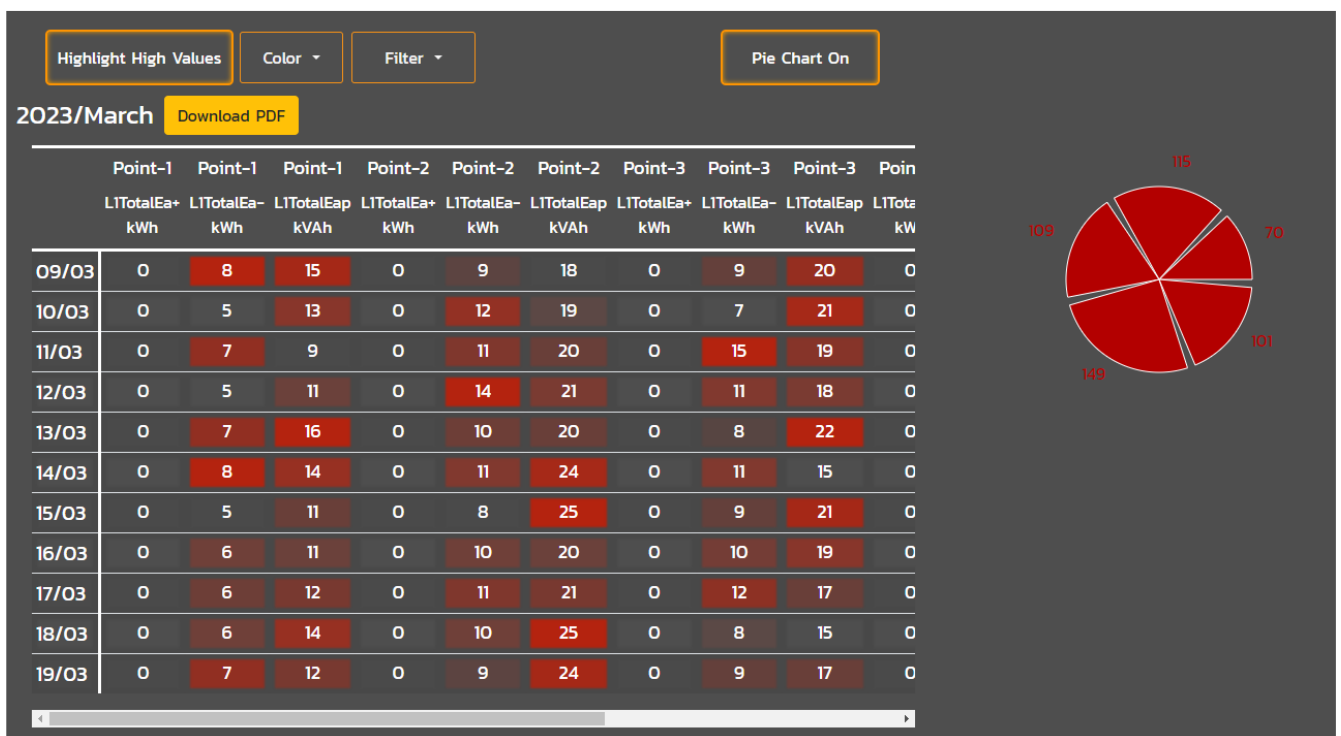
Maanantai 3.4.2023

Tavoitteenani oli tälle päivälle saada energiaraporttinäkymää eteenpäin ja paremman näköiseksi. Halusin kehittää heatmap-toiminallisuuden ulkonäköä, joka oli jäänyt pitkäksi aikaa huomiotta. Viikon 4 kuvissa 27 ja 28 näkyy, mihin olin jäänyt taulukon tyylittelyn suhteen. Seuraavissa kolmessa kuvassa 41, 42 ja 43 näkyy, mihin etenin tämän maanantaina aikana.

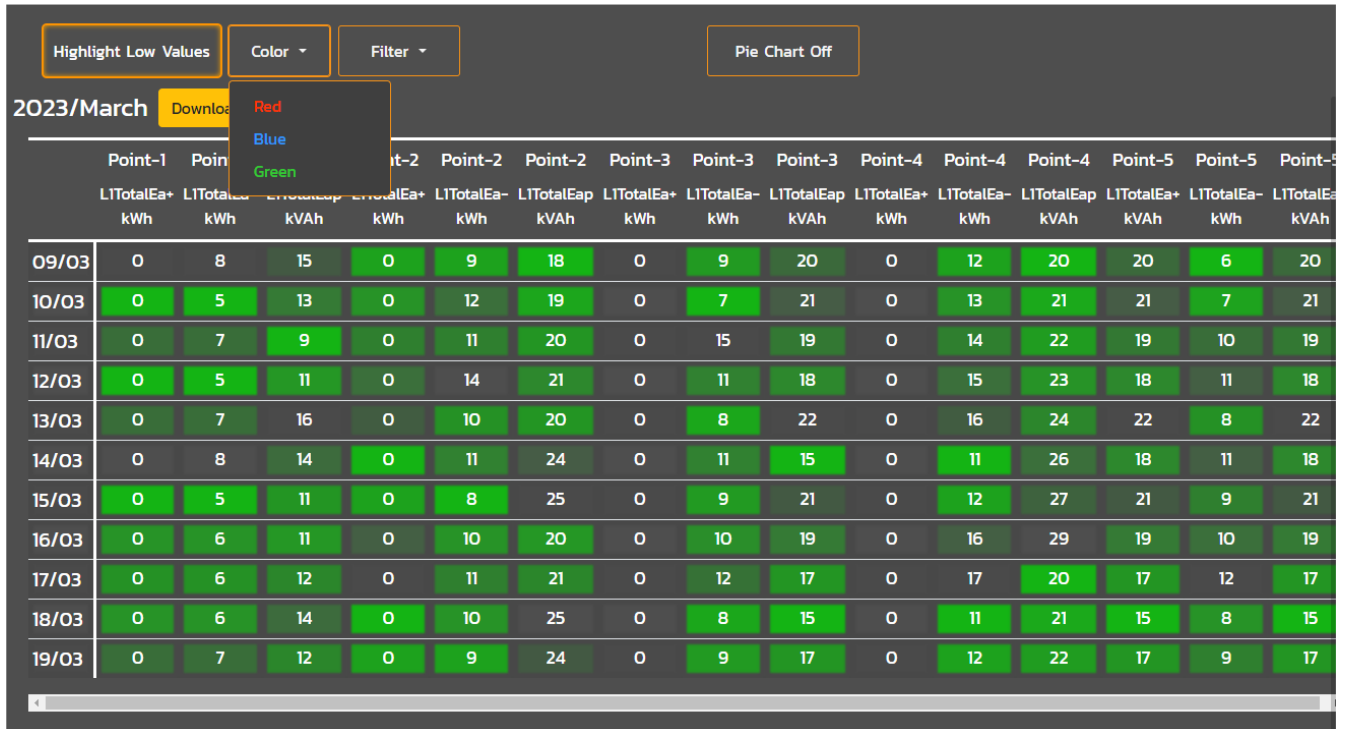
Kuvassa 41 käyttäjä on painanut "heatmap"-toiminnon pois päältä ja piilottanut raportista ympyräkaavion. Lopuksi hän on suodattanut taulukosta näkyviin vain "L1TotalEa"-arvot. Kuvassa 42 käyttäjä korostaa jokaisen sarakkeen suurimmat arvot "heatmap"-ominaisuudella. Taulukko ei mahdu selaimen ikkunaan, joten siinä on rullaustoiminto, jonka avulla käyttäjä voi rullata (eng. scroll) taulukkoa oikealle nähdäkseen lisää taulukkoa. Kuvassa 43 käyttäjä korostaa jokaisen sarakkeen pienimmät arvot "heatmap"-ominaisuudella ja on valinnut värjätuille soluille vihreän värin. Kuvassa 43 käyttäjä on piilottanut ympyräkaavion pois raportista. Kuva 43 paljastaa myös bugin, sillä sarakkeissa, joiden soluissa on pelkästään "0"-arvoja, ei solujen pitäisi olla korostettuna.



Kuva 41. Energiaraporttinäkymän työversio, jossa heatmap ja ympyräkaavio pois päältä.



Kuva 42. Energiaraporttinäkymän työversio, jossa heatmap ja ympyräkaavio ovat päällä.



Kuva 43. Energiaraportinäkymän työversio, jossa heatmap värjää pienimmät arvot vihreinä.

Kokeilin myös täysin erilaista tapaa värjätä soluja, joka on nähtävillä kuvassa 44. Kuvissa 41, 42 ja 43 annan jokaiselle solulle "inline"-tyylinä rgba-väriä alpha-arvon, jonka suuruuden olen laskenut solun sisältävän numeraalisen arvon verrannollisuudesta muihin saman sarakkeen solujen numeraalisiin arvoihin. Kuvassa 44 annan solulle saman laskukaavan perusteella tyyliarvon alpha-arvon sijaan. Kuvassa 44 solut hyödyntävät neljää erilaista tyyliarvoa. Jokaisen sarakkeen pienimmät arvot saavat "very-low-value"-tyyliarvon suurimmat arvot "very-high-value"-tyyliarvon. Väliin putoavat arvot saavat tyyliarvot "low-value" ja "high-value". Vaikka hylkäsin kuvan 44 iteraation, haluan tässä tuoda silti esiin sen, että kokeilen usein erilaisia asioita koittaessa löytää sopivaa tapaa tyyliarvoilla käyttöliittymää.



Kuva 44. Keskeneneräinen ja hylätty työversio energiarapottinäköymästä.

Taulukkoa voisi varmasti parannella lisää, mutta päivän loppupuolella ajattelin siirtyä viimein ympyräkaavion tekemiseen. Kuvassa 42 ympyräkaavio hakee taulukosta kaikki "kWh"-mittayksikön omaavat mittauspisteet ja näyttää niistä 5 suurinta kWh-kulutuksen mukaan. Ympyräkaavio jäi hyvin kesken, joten siinä ei tällä hetkellä näy, että mille mittauspisteille mikäkin ympyräkaavion piirakka kuuluu.

Tiistai 4.4.2023

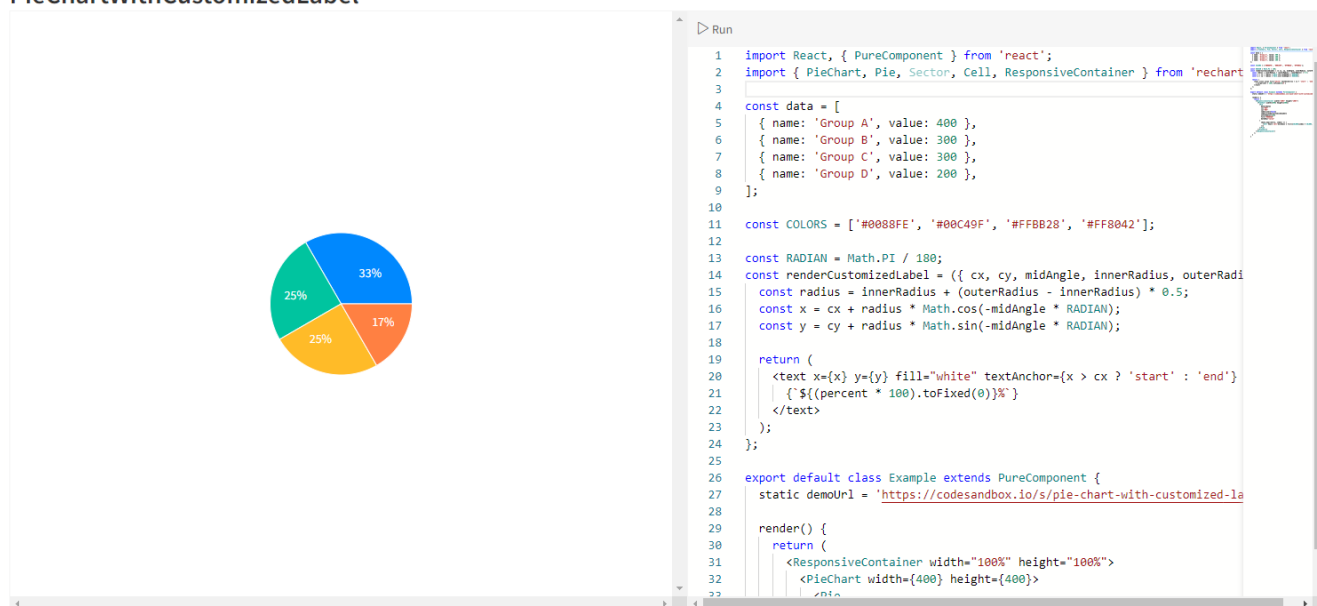
Tällä hetkellä energiaraportin heatmap-ominaisuus korostaa jokaiselta sarakkeelta solut suhteessa oman sarakkeensa muihin arvoihin. Aloin pohtia taulukkoa käyttäjän kannalta ja mielestäni voisi olla tarpeellista korostaa soluja suhteessa taulukon kaikkiin arvoihin, tosin silloin pitäisi jotenkin huomioida eri mittayksiköt. Tiiminvetäjä totesi, että ennen kuin taulukko on käytössä ensimmäisellä asiakkaalla, on vaikea tunnistaa kaikkia tarvittavia ominaisuuksia ja minkä ominaisuuksien kehitystä kannattaa priorisoida. Ehkä tulen lisäämään mahdollisuuden suodattaa taulukkoa mittayksiköiden mukaan, jolloin heatmap-toiminnon voisi laittaa toimimaan tarvittaessa kaikilla saman mittayksikön omaavilla soluilla.

Tutkin Recharts-kirjaston ympyräkaaviokomponenttia ja ympyräkaaviokomponentin konfiguraatiomahdollisuuksia. Mielestäni kirjastot ovat parhaimmillaan silloin, kun niiden dokumentaatiossa on selkeitä esimerkkejä kirjaston käytöstä, sekä itse kirjaston käyttö tuntuu riittävän helpolta. Ympyräkaavion piirtämisessä ja siihen erilaisten elementtien sijoittamisessa tarvitaan geometriaa sekä

trigonometriaa ja matemaattiset taitoni ovat ruosteessa. Minulla olisi kiinnostusta rakentaa ympyräkaavio itse, kuten tein heatmap-ominaisuuden kanssa, mutta minusta tuntuu, että tähän minun on parempi hyödyntää kirjastoa.

Recharts-kirjastossa on 7 erilaista valmista ympyräkaavioesimerkkiä rakennettuna, joiden avulla on hyvä päästä alkuun. Kuvassa 45 on yksi näistä Recharts-kirjaston dokumentaatioissa olevista esimerkeistä. Kuvassa 45 oikealla puolella on dokumentaation tarjoama lähdekoodi esimerkille, ja vasemmalla puolella koodin tuottama ympyräkaavio. (Recharts 2023.)

PieChartWithCustomizedLabel



Kuva 45. Recharts-kirjaston dokumentaatiosta löytyvä ympyräkaaviokomponentti.

Ympyräkaavion kanssa kokeilin erilaisia muotoja ja kirjaston ominaisuuksia, mutta en saanut päivän aikana ympyräkaaviota käyttövalmiiksi. Ympyräkaavion olisi syytä hakea data käyttöönsä käyttäjän suodattamista energiataulukon arvoista. Minulla on vielä pohdinnan ja kehityksen alla paras tapa ohjelmoida taulukon ja ympyräkaavion välinen logiikka ja ne tavat, jolla käyttäjä voi manipuloida molempia. Minun tulee pohtia muun muassa, tarvitseeko ympyräkaavio omia suodatustoimintoja tai ylimääräisiä muita toimintoja, vai riittääkö, että ympyräkaavio muodostuu taulukon arvoista automaattisesti.

Keskiviikko 5.4.2023

Minun tulisi saada energiaraporttinäkymän ensimmäinen käyttökelpoinen versio valmiiksi tämän työviikon loppuun mennessä. Minulle iski kiire saada ympyräkaavio toimimaan ja hiottua energiaraporttinäkymä käyttövalmiiksi muun muassa raportin responsiivisuuden osalta. Responsiivisuuden

osalta jätän kännykät huomiotta, sillä tämän kaltainen raporttinäkymä vaatisi täysin oman muotoilunsa niin pienelle näytölle. Lisäksi sovellus ei ole muutenkaan kännykälle optimoitu, kuin vain joiltakin osin. Koitan saada raportin käyttökelpoiseksi tietokoneen vaaka- sekä pystynäytöille ja mielellään myös tablet-laitteille.

Koska minulla oli 2 työpäivää aikaa toteuttaa loput ominaisuudet, täytyi tehdä välittömiä päätöksiä, mitä toimintoja luulen ehtiväni kehittää valmiiksi. Päätin, että ympyräkaavio muodostuu automaattisesti taulukon arvoista ja käyttäjän suodattaessa taulukkoa, myös ympyräkaavio muodostuu uudelleen käyttäen vain taulukossa näkyviä arvoja. Ympyräkaavio hakisi automaattisesti suurimmat energiakulutukset ja tähän ei käyttäjä pystyisi vaikuttamaan.

Pidin tärkeänä myös lisätä mahdollisuuden suodattaa taulukon arvoja mittayksiköiden mukaan, joten lisäsin tämän mahdollisuuden energiaraporttiin. Kuvassa 46 on funktio `handleFilterChange()`, joka vastaa taulukon arvojen suodattamisesta. Olin aiemmin koodannut suodatuksen "valueName"-arvojen mukaan. Lisäsin funktioon logiikkaa ja nyt funktio vastaanottaa "filterString"-arvon, joka voisi olla esimerkiksi "valueName-L1TotalEa" tai "unitName-kWh". Tämän jälkeen funktio pilkkoo "filterStringin" keskeltä poikki ja luo JavaScript-tilaukkoa (array) nimeltä "filterValues". Tämän jälkeen funktio suorittaa energiaraportin datan suodatuksen käyttäen "filterValues"-tilaukkoa (array) sisältämiä arvoja.

```
const handleFilterChange = (filterString) => {
  // if user has selector filter "Show all", use unfiltered reportData
  if (filterString === "all") {
    setFilteredReportData(reportData);
  } else {

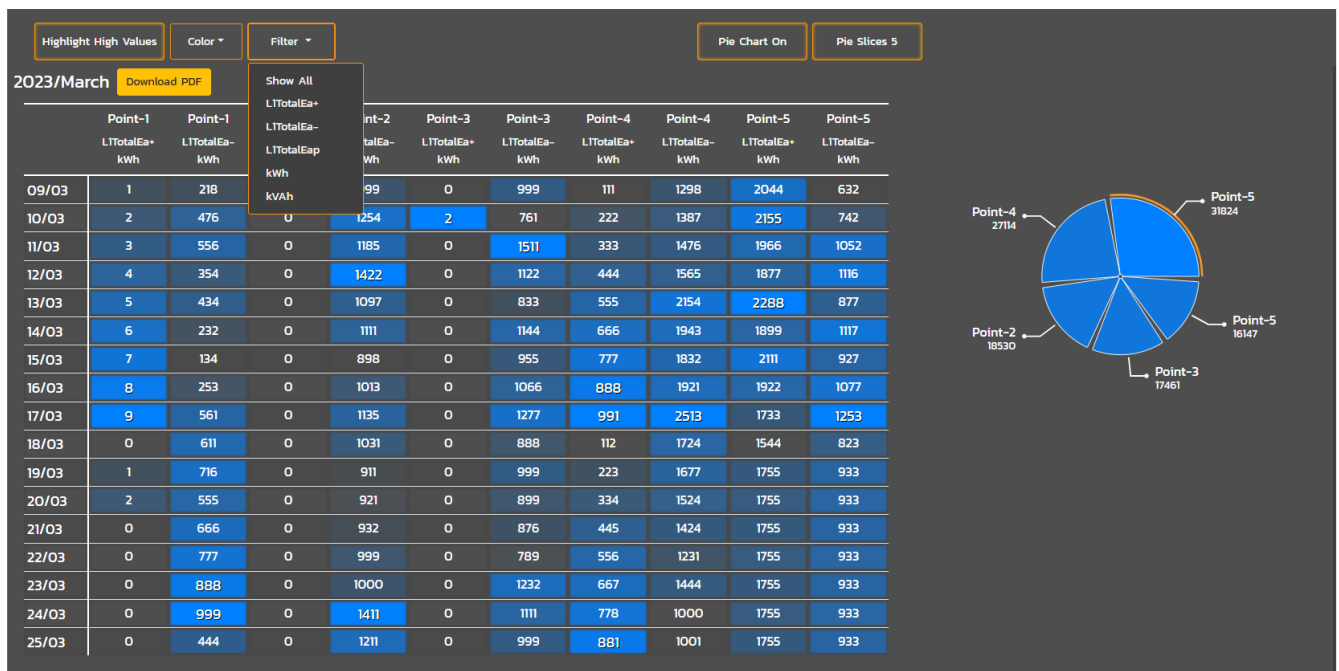
    const filterValues = filterString.split("-"); // e.g: "valueName-
L1TotalEa" turn into ["valueName", "L1TotalEa"]
    let filteredData;

    if (filterValues[0] === "valueName") {
      filteredData = reportData.map((report) => {
        const filteredValues = report.Value.filter((value) => value.ValueName
=== filterValues[1]);
        return { ...report, Value: filteredValues };
      });
    } else if (filterValues[0] === "unitName") {
      filteredData = reportData.map((report) => {
        const filteredValues = report.Value.filter((value) => value.Unit ===
filterValues[1]);
        return { ...report, Value: filteredValues };
      });
    }
    setFilteredReportData(filteredData);
  }
};
```

Kuva 46. Energiaraporttidatan suodatusfunktio.

Päivän loppuun mennessä olin saanut myös ympyräkaavion toimimaan haluamallani tavalla. Lisäsin painikkeen, jolla voi säätää ympyräkaavion piirakoiden lukumäärää 1–5 väliltä. Ympyräkaavio hakee itselleen arvot taulukon datasta ja mikäli käyttäjä suodattaa taulukkoa, myös ympyräkaavio päivittyy.

Kuvassa 47 minulla on suodatuspainike auki ja olen valinnut sieltä arvon "kWh", jonka jälkeen taulukossa näkyy ainoastaan kWh-mittayksikön arvot. Olen myös valinnut "Color"-painikkeen alta sinisen värin raportin heatmap-ominaisuudelle ja ympyräkaaviolle. Heatmap-ominaisuus siis korostaa valitulla värillä jokaisesta sarakkeesta ne solut, joissa on suurimmat arvot. Ympyräkaavioon olen valinnut näytettäväksi 5 arvoa "Pie Slices"-painikkeen avulla. Ympyräkaavion suurin piirakka/segmentti korostuu vielä erikseen pienellä oranssin värisellä ulkoseinällä ja kirkkaammalla värisävyllä.



Kuva 47. Energieraporttinäkymän työversio.

Torstai 6.4.2023

Tavoite tälle päivälle oli saada energieraporttinäkymän ensimmäinen versio viimeisteltä. Energieraporttinäkymän koodi tuli siistiä ja lopuksi tehdä yhdistämispyyntö (eng. pull request) minun työhaarastani yhteiseen kehityshaaraan. Raportti oli hyvällä mallilla, mutta ympyräkaaviolla ei ollut minkäänlaista responsiivisuutta, eli näytön koon muutokset tai selaimen ikkunan koon muutokset tekivät ympyräkaaviosta ja pahimmillaan koko raporttinäkymästä hyvin rikkiäisen näköisen.

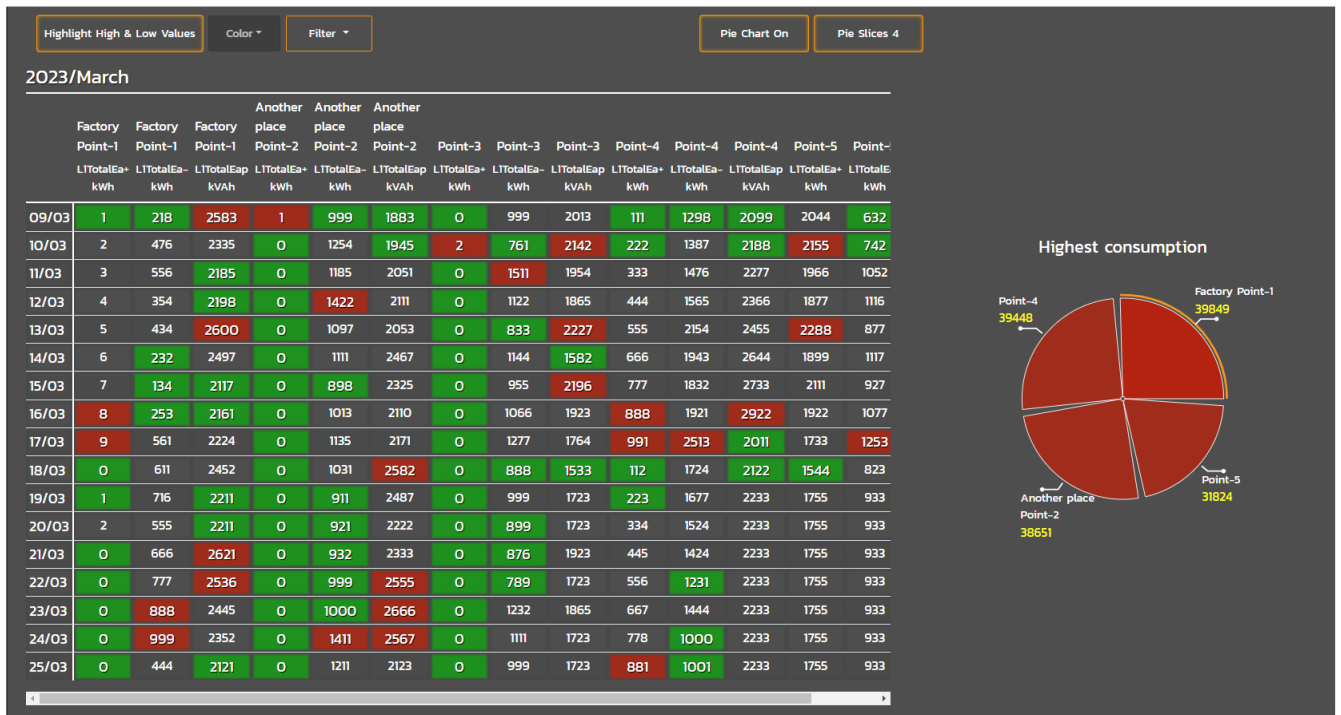
Tähän olisi varmasti useitakin lähestymistapoja. Koska käytän Bootstrap-kirjaston "grid"-ominaisuuksia, voisin hyödyntää niitä ja koittaa saada niillä näkymän responsiiviseksi. Näin itse asiassa teinkin, mutta jotkin ongelmat pysyivät tässä ratkaisussa läsnä. Ensinnäkin ympyrän koko oli aina

muuttumaton, riippumatta näytön koosta tai ympyräkaavioille varatun tilan koosta. Tämä oli ongelma, sillä liian pieni ympyrä näyttää huonolta isolla näytöllä, ja sen sijaan liian iso ympyrä ei vain sovi ja mahdu tabletin kokoiselle näytölle. Toisekseen ympyräkaavion piirakoiden otsikkotekstit katosivat tietyn kokoisissa näytöissä viereisen taulukon alle piiloon tai katkesivat ympyräkaavion oikealta puolelta, mikäli sieltä loppui tila kesken. Jokaisesta ympyräkaavion piirakasta lähtee pieni valkoinen osoitinviiva, jonka päässä on siis kyseisen piirakan otsikkoteksti. Tämä on nähtävillä muun muassa kuvassa 47. Kuvassa 47 piirakan tekstit mahtuvat ruudulle hyvin, sillä näyttö on leveä.

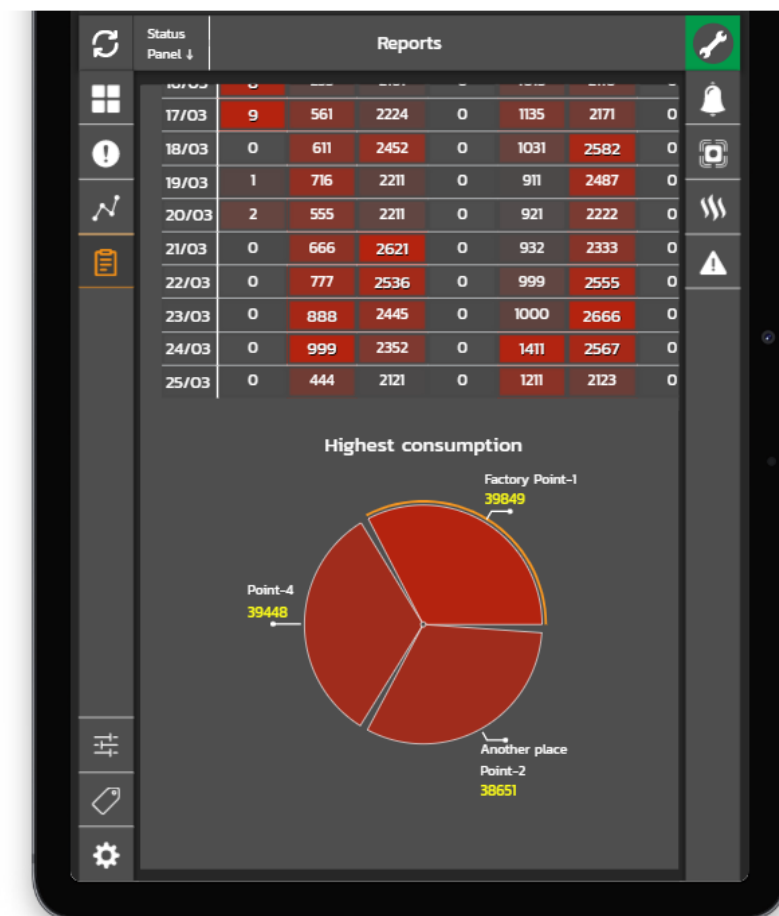
Päätin ohjelmoida ympyrän koon muuttumaan käyttäjän selaimen näytön koon mukaisesti. Tämän jälkeen ongelmallinen osuus oli saada ympyräkaavion piirakoiden tekstit sopeutumaan myös erilaisiin näyttökokoihin. Halusin, että piirakoiden tekstit hakeutuvat lähemmäs itse ympyräkaaviota pienillä näytöillä, ja että tekstit voivat tulla ympyräkaavion päälle tarvittaessa. Tämä ei ollut yksinkertainen tehtävä ainakaan minulle, joten hyödynsin jälleen kerran ChatGPT-tekoälyä työkaluna.

Kuvassa 48 on energiaraporttinäkymän ensimmäinen valmis versio. Kuvassa 48 olen asettanut heatmap-toiminnon korostamaan suuret arvot punaisina ja pienet arvot virheinä. Kuvassa 49 sen sijaan näkyy, miten valmis versio taipuu tablet-laitteen näytölle. Tablet-laitteella pystyasennossa ympyräkaavio asetetaan suosiolla taulukon alapuolelle. Kuvassa 49 reunoilla näkyy myös muuta sovelluksen käyttöliittymää, mutta energiaraportti on siis kuvassa keskellä.

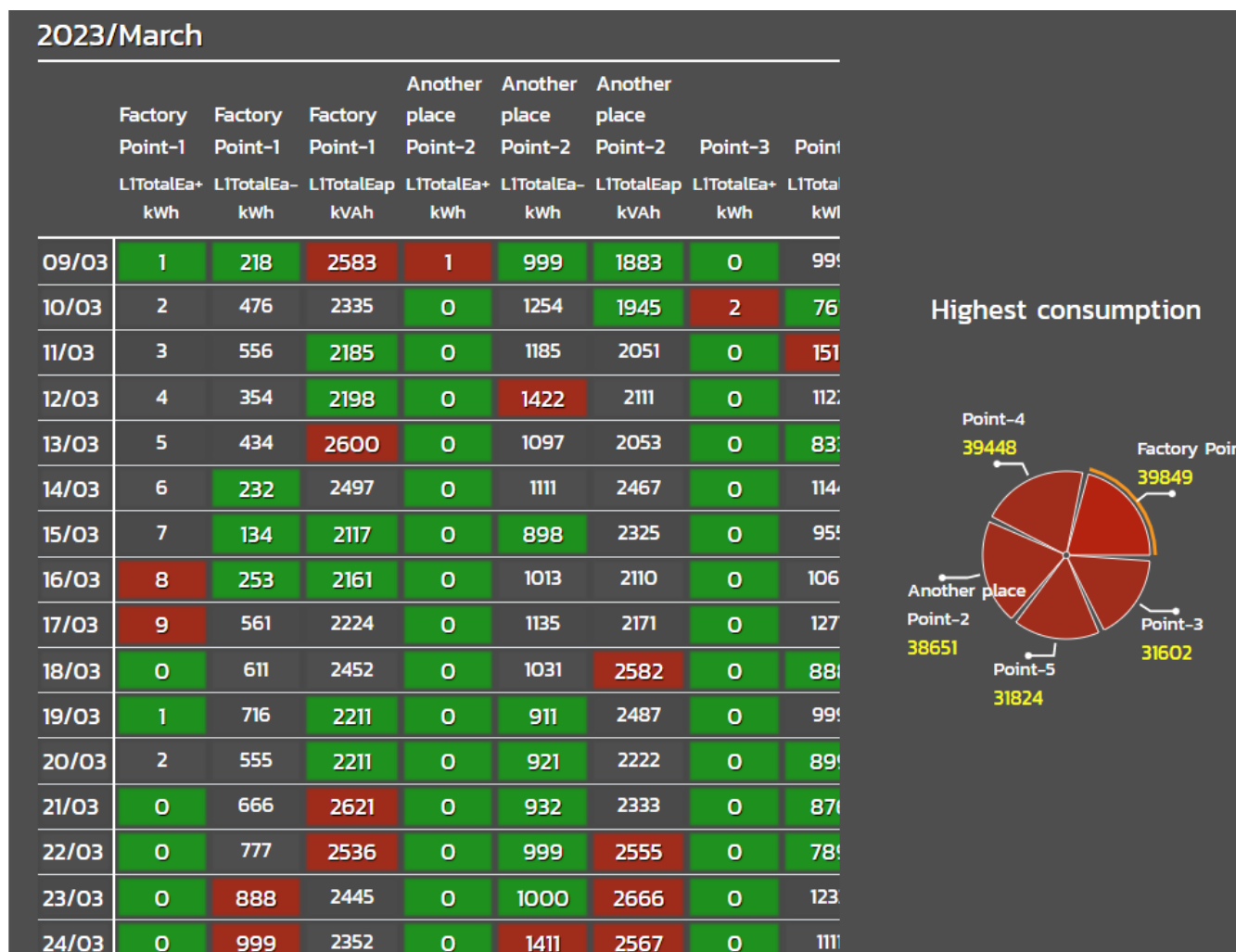
Kuvassa 50 näkyy, miten energiaraporttinäkymä taipuu epätasaisiin näyttökokoihin. Kuvassa 50 ympyräkaavio on pienellä näytöllä, jonka kuvasuhde on noin 1:1. Kuvassa 50 ympyräkaavio on paljon pienempi, kuin kuvassa 48, ja lisäksi sen otsikkotekstit menevät enemmän piirakoiden päälle niiden koittaessa mahtua ruudulle. Kuvasta on hieman vaikea hahmottaa, että raportti on todellakin pienellä näytöllä ja tämän vuoksi myös ympyrän todellista kokoa on kuvasta vaikea hahmottaa. Hahmottamista saattaa helpottaa, jos vertaat kuvien 48 ja 50 ympyräkaavioiden tekstejä, sillä molemmissa kuvissa fonttikoko on sama. Kuvassa 50 samankokoinen fontti on paljon suurempi suhteessa ympyrän kokoon, kuin kuvassa 48. Ympyräkaavio siis muuttaa kokoaan toimivasti suhteessa näytön kokoon.



Kuva 48. Ensimmäinen valmis versio energiaraporttinäkymästä.



Kuva 49. Ensimmäinen valmis versio energiaraporttinäkymästä tablet-laitteella.



Kuva 50. Ensimmäinen valmis versio energiaraporttinäkymästä pienellä näytöllä.

Viikkoanalyysi 8

Päällimmäisenä viikosta jäi mieleen viimeisten päivien loppurutistus. Tuntui, että sain enemmän aikaiseksi kahden päivän aikana, kuin toisinaan saan aikaiseksi viikon aikana. Tämä johtui siitä, että tiiminvetäjän suunnalta tuli painetta saada energiaraporttinäkymän ensimmäinen versio valmiiksi tämän kehitysjakson aikana. Ilman asetettua takarajaa, olisin todennäköisesti jatkanut toimintojen lisäämistä energiaraporttinäkymään.

Olisin halunnut jatkokehittää muun muassa taulukon heatmap-toimintoja eteenpäin. Olisin halunnut käyttäjän voivan valita, värjääkö heatmap-toiminto jokaiselta sarakkeelta suurimmat arvot, vai suurimmat arvot koko taulukosta. Lisäksi raportin tallennusominaisuus tietokoneelle PDF-muotoisena

jäi täysin alkutekijöihin ja tallennettuun PDF-dokumenttiin ei sisälly muun muassa raportin ympyrä-kaaviota laisinkaan. Kuvassa 51 näkyy, millaisena raportti tulostuu tällä hetkellä, kun käyttäjä painaa "Download PDF"-painiketta.

	Factory Point-1 L1Total Ea+ kWh	Factory Point-1 L1Total Ea- kWh	Factory Point-1 L1Total Eap kVAh	Another place Point-2 L1Total IEa+ kWh	Another place Point-2 L1Total IEa- kWh	Another place Point-2 L1Total Eap kVAh	Point-3 L1Total Ea+ kWh	Point-3 L1Total Ea- kWh	Point-3 L1Total Eap kVAh	Point-4 L1Total Ea+ kWh	Point-4 L1Total Ea- kWh	Point-4 L1Total Eap kVAh	Point-5 L1Total Ea+ kWh	Point-5 L1Total Ea- kWh	Point-5 L1Total Eap kVAh
09/03	1	218	2583	1	999	1883	0	999	2013	111	1298	2099	2044	632	2000
10/03	2	476	2335	0	1254	1945	2	761	2142	222	1387	2188	2155	742	2199
11/03	3	556	2185	0	1185	2051	0	1511	1954	333	1476	2277	1966	1052	1988
12/03	4	354	2198	0	1422	2111	0	1122	1865	444	1565	2366	1877	1116	1877
13/03	5	434	2600	0	1097	2053	0	833	2227	555	2154	2455	2288	877	2266
14/03	6	232	2497	0	1111	2467	0	1144	1582	666	1943	2644	1899	1117	1855
15/03	7	134	2117	0	898	2325	0	955	2196	777	1832	2733	2111	927	2144
16/03	8	253	2161	0	1013	2110	0	1066	1923	888	1921	2922	1922	1077	1933
17/03	9	561	2224	0	1135	2171	0	1277	1764	991	2513	2011	1733	1253	1722
18/03	0	611	2452	0	1031	2582	0	888	1533	112	1724	2122	1544	823	1522
19/03	1	716	2211	0	911	2487	0	999	1723	223	1677	2233	1755	933	1711
20/03	2	555	2211	0	921	2222	0	899	1723	334	1524	2233	1755	933	1711

Kuva 51. Energiaraportin tallennus PDF-dokumenttina.

Tämä on muistutus minulle siitä, miksi määräajat/takarajat (eng. deadlines) ovat hyödyllisiä. Ne "pakottavat" sinut saamaan asioita eteenpäin ja valmiiksi. Itselleni määräaika tuo tietynlaista kiireen tuntua ja sitä kautta energiaa suorittaa tehtävä kyseiseen määräaikaan mennessä. Lisäksi, jos mietimme kehittämääni energiaraaporttia, niin määräaika pakotti minut tekemään päätöksiä siitä, mitä ominaisuuksia leikkaan pois tästä ensimmäisestä versiosta, ja mitkä ominaisuudet koitan saada valmiiksi määräaikaan mennessä.

Journalisti Christopher Cox tutkii määräaikojen tuottamaa positiivista vaikutusta työn tekemiseen kirjassaan "The Deadline Effect". Hän puhuu muun muassa siitä, kuinka jollain tasolla tuntuu väärältä rajoittaa luovaa työtä tekevää henkilöä aikamääräellä sen sijaan, että luovalle henkilölle annettaisiin täysi vapaus ajallisesti tuottaa paras mahdollinen lopputulos. Mutta se, mitä hän on ha-

vainnut tutkiessaan aihetta ja siitä tehtyjä tutkimuksia on, että kun henkilölle annetaan järkevä aikamääre tehtävän suorittamiseksi, henkilö saavuttaa luovan ja aikaan saavan mielentilansa nopeammin. Ilman aikamäärettä henkilö on taipuvainen viivyttämään ja joutuu tekemään enemmän töitä saavuttaakseen tämän mielentilan. (Christopher 8.9.2021, 15:58 – 17:20.)

Ajanhallintavalmentaja ja Real Life E ajanhallintavalmennusyrityksen toimitusjohtaja Elizabeth Grace Saunders kertoo aikamääreiden olevan tärkeitä työn priorisoinnin kannalta. Kun tehtävälle on asetettu konkreettinen aikamääre ja osaat arvioida siihen vaadittavan työmäärän, pystyt helpommin kieltäytymään uusista tehtävistä, jotka hidastaisivat meneillä olevaa tehtävääsi. Aikamääreet voivat antaa sinulle virtaa ja tarvittavaa keskittymistä saattaa työ loppuun sovitussa ajassa. (Grace Saunders 2016.)

Pidän myös työnhakupalvelu Indeedin editoritiimin kirjoituksesta ”Why Deadlines Are Important”, sillä moni kohta heidän tekstistänsä tuntui tukevan niitä tunteita, joita energiaraportin valmiiksi saamiselle asetettu aikamääre minulle aiheutti. He kertovat aikamääreen auttavan priorisoimaan ne asiat, jotka ovat tärkeää saada valmiiksi ja lisäksi päästämään irti perfektionismista, johon itsekkin syyllistyn front end -kehityksessä. He myös tuovat esiin sen, että ilman aikamäärettä tehtävän valmiiksi saattaminen kestää usein pidempään, kuin aikamääreen kanssa. (Indeed Editorial Team 2022.)

Meillä on käytössä ketterän kehityksen mukaiset kehitysjaksot (eng. sprint), ja kehitysjaksolle asetamme aina itsellemme tehtävät, jotka olisi syytä saada valmiiksi kehitysjakson aikana. Jos minulla ei ole kuitenkaan konkreettisempaa painetta saada tehtävää valmiiksi, saattaa tehtävä usein venyä ja siirtyä seuraavalle kehitysjaksolle. Tämä johtuu usein muun muassa siitä, että kehittäessäni esimerkiksi jotain uutta toimintoa tai näkymää sovellukseen, haluan tehdä siitä mahdollisimman hyvän.

Ohjelmoidessa käyttöliittymää tuntuu, että et ole koskaan valmis ja löydät aina jotain pientä parannettavaa, johon käyttää aikaasi. Jos minulle annetaan konkreettisempi paine ja aikamääre saattaa tehtävä valmiiksi, lopetan loputtoman funktioiden ja käyttöliittymän ulkoasun parantamisen ja pusken itseni tuottamaan toimivan ja kelvollisen lopputuloksen. Pystyn paremmin fokusoimaan itseni työstämään kokonaisvaltaista ratkaisua, jossa kaikki toimii, mutta kaikkea voisi vielä jatkokehittää. Ilman aikamäärettä saatan ajautua hiomaan jotain yksityiskohtaa täydelliseksi, ennen kuin voin siirtyä seuraavaan yksityiskohtaan, joka tulee saada täydelliseksi, ennen kuin voin siirtyä seuraavaan yksityiskohtaan ja niin edelleen.

3.9 Seurantaviikko 9

Tiistai 11.4.2023

Päivä oli poikkeuksellisen palaveritäyteinen. Suoritimme ohjelmistokehitystiimin kesken kehitysjaksosuunnittelupalaverin (eng. sprint planning), eli edellisen kehitysjakson läpikäynnin ja alkavan kehitysjakson suunnittelun. Esittelin työstämäni energiaraporttinäkymän tiimille ja demosin sen toiminnallisuudet läpi. Tiedossa olevat bugit olin kirjannut ylös Azure Devopsin tehtävätaululle, jossa siis ylläpidämme tilannekuvaa kunkin tiimin jäsenen tehtävistä, sovelluksen bugeista, tulevista ominaisuuksista ynnä muusta.

Kehitysjaksosuunnittelupalaverin jälkeen korjasin muutaman bugin energiaraporttinäkymästä ja hyppäsin mukaan seuraavaan kokoukseen, jossa kävimme läpi tilannetta sovelluksen testaukseen liittyen. Testausta suunnittelee ja rakentaa ulkopuolinen ohjelmointiyritys, eli he kävivät meille läpi, mitä ovat saaneet aikaseksi ja mitä ongelmia he ovat kohdanneet. Pidin tapaamista mielenkiintoisena, vaikka palaverissa käytiin läpi ainoastaan back endin koodin testausta.

Tämän jälkeen oli seuraava tapaaminen, jossa toinen ulkopuolinen ohjelmointiyritys, jonka kanssa olemme myös yhteistyössä, saapui toimistolle keskustelemaan muun muassa sovelluksen tietokantapäivityksestä. Päivän päätteeksi oli vielä yrityksemme oma kuukausipalaveri.

Vaikka päivän aikana olisi ollut ehkä joitakin tunteja aikaa ohjelmoida, oli aika kuitenkin niin katkokanista palaverien välissä, että en kyennyt saamaan paljoa aikaiseksi. Minulle on hankala siirtyä sormia napsauttamalla ohjelmoimaan monimutkaista sovellusta ja palauttaa mieleen, mitä minun oli tarkoitus ohjelmoida, jos olen juuri tehnyt jotain aivan muuta. Tämä on ilmeisesti kuitenkin melko normaalia, sillä esimerkiksi tiimimme sisällä muut henkilöt tuntevat kanssani samoin.

Keskiviikko 12.4.2023

Kollegani pidempään työstämä SVG-kuva graafista käyttöliittymäosuutta varten oli valmis. Käytin päivän kuvan korjaamiseen, kuten asioiden uudelleen nimeämiseen ja tiettyjen elementtien pieniin muokkauksiin. Kollega oli upottanut SVG-kuvan sisään ikonikuvia PNG-muotoisena, joka tarkoitti, että en pysty koodilla pääsemään käsiksi näiden ikonikuvien sisältämiin viivoihin ja väreihin. Poistin upotetut PNG-kuvat ja piirsin itse niiden tilalle vektorigrafiikkaa hyödyntäen vastaavat kuvat. Erona oli siis se, että minun kuvani olivat oikeasti vektorigrafiikkaa, eikä upotettu PNG-kuva vektorigrafiikan seassa. Tämä oli tärkeää siksi, että pääsen koodissa käsiksi kaikkiin <path>, <rect> sekä muihin vektorigrafiikkaelementteihin ja muokkaamaan näiden elementtien väriä, kokoa, paksuutta ynnä muuta.

Kollega oli osittain tehnyt turhaa työtä, mutta se on täysin ymmärrettävää, sillä hän ei tiedä, kuinka elementtejä voidaan tyyllitellä ja manipuloida ja minkälaisia rajoituksia tai haasteita siihen voi liittyä. Graafisen käyttöliittymäosuuden SVG-kuvan piirtäminen ja suunnittelu oli sellainen tehtävä, että siihen tarvittiin minun tietojani juuri front end -puolen logiikkaan liittyen.

Viikkoanalyysi 9

12.4 keskiviikon jälkeen minulle tuli mieleen, että olisin voinut toimia itse järkevämmiin vektorigrafiikka-asioihin liittyen. Tulin piirtäneeksi uudelleen samoja ikonikuvia, kuin kollega oli jo piirtänyt. Tajusin jälkepäin, että kollegalla oli varmaan tallessa jossain hänen piirtämänsä versiot ja ne olisi ehkä voitu tuoda SVG-kuvaan suoraan vektoridatana. Toisin sanoen, todennäköisesti tein myös itse turhaa työtä piirtäessäni kuvat uudelleen.

4 Pohdinta

Seurantajakson raportoinnin aikana syvensin osaamistani ja tietojani sovelluskehitystyöstä sekä omiin työtehtäviini liittyvistä termeistä, tekniikoista ja aiheista. Kirjoittaessa päivä- ja viikkoraportteja jouduin jäsentämään ja pohtimaan työni haasteita eri tavalla, kuin tavallisesti. Vaikka termit ja tekniikat kuten esimerkiksi IIot (industrial Internet of things), ketterä kehitys ja WebSocket olivat minulle tuttuja, niiden auki selittäminen lukijalle ja niistä kirjoittaminen selkeytti niiden merkitystä samalla myös minulle.

Sovelluskehittäjänä olen mielestäni tämän opinnäytetyön aikana kasvanut suurimmaksi osaksi vain front end -puolen kehityksessä, mutta toivoisin tulevaisuudessa voivani kasvattaa osaamistani kokonaisvaltaisesta ohjelmistokehityksestä. Tämänhetkinen vastuualueellani oleva front end -puolen suuri työmäärä yhdistettynä siihen, että olen tiimini ainoa front end -kehittäjä tarkoittaa, että en todennäköisesti tule back end -puolta kehittämään tämänhetkisessä työympäristössä ainakaan lähitulevaisuudessa. Mikäli en työn puitteissa pysty back end -ohjelmointitaitojani kehittämään, täytyy minun mahdollisesti edistää näitä taitoja vapaa-ajalla ja omissa projekteissa.

Olin myös ennen tätä opinnäytettä ja opinnäytetyön tekoprosessin alussa pohtinut, kuinka voisin paremmin analysoida kirjoittamaani koodia, sillä tiimissäni ei ole asiantuntijatasoista front end -kehittäjää ja esimerkiksi tiiminvetäjämme front end -puolen taidot ovat minua heikommat. Eli lähelläni ei ole henkilöä, joka voisi opastaa minua vaativammissa front end -puolen ohjelmointiasioissa. Oli mielenkiintoista huomata, kuinka ChatGPT-tekoäly muodostui tämän opinnäytetyön aikana jonkinasteiseksi koodimentoriksi minulle, sillä ChatGPT-tekoälyltä pystyin saamaan välitöntä palautetta kirjoittamastani koodista.

ChatGPT-tekoäly nousikin yhdeksi keskeiseksi aihealueeksi viikkoraporttien aikana. Olen tämän opinnäytetyön tekemisen aikana saanut suuren määrän omakohtaista kokemusta ChatGPT-tekoälystä, mutta hakenut myös Internetistä tietoa siihen liittyen. Olen huomannut oman työtehokkuuteni kasvaneen ChatGPT-tekoälyn myötä, sillä sen avulla vastausten etsiminen ohjelmointiaiheisiin kysymyksiini on ollut nopeampaa, kuin Googlen hakukoneen avulla, jota tyypillisesti olen käyttänyt ja käytän toki edelleen.

YouTube-kanavalla ”bri does things” on hyvä videoesimerkki ChatGPT-tekoälyn hyödyntämisestä tiedon ja oppimateriaalin haussa. Videolla Bri näyttää, mitä hän kysyy ChatGPT-tekoälyltä ja videossa näkyy myös tekoälyn vastaukset. Bri pyytää kirjoja, videoita, podcasteja ja harjoituksia Python-koodikielen opiskeluun ja ChatGPT tarjoaa hyvin kattavan listan opiskelumateriaalia Brille. Tärkeäksi huomioksi Bri nostaa sen, että jos hän lähtisi itse manuaalisesti etsimään yhtä kattavaa materiaalilistaa, häneltä kuluisi siihen vähintäänkin useita tunteja aikaa. Lisäksi ChatGPT-tekoälylle

voi antaa tarvittaessa tarkempia ohjeita tiedonhakuun liittyen. Esimerkkinä tästä Bri nostaa esiin sen, että voit esimerkiksi pyytää ChatGPT-tekoälyä tarjoamaan vain sellaista materiaalia, joka on ilmaista ja vapaasti käytettävissä, mikäli et ole valmis maksamaan opiskelumateriaalistasi. (bri does things 20.4.2023, 03:05 – 04:00).

Koen, että tulevaisuudessa ChatGPT-tekoälyn ja sen kaltaisten työkalujen hyödyntäminen yleisesti työelämässä tulee olemaan tärkeää. ChatGPT-tekoälyn kaltaisia tekoälyjä ollaan jo sisällyttämässä yrityksissä tyypillisesti käytössä oleviin muihin työkaluihin, joten uskon, että tavanomainen toimistotyöntekijä ei pääse tätä muutosta karkuun. Muun muassa Google ja Microsoft ovat julkistaneet omat tekoälynsä, Microsoft 365 Copilotin sekä Google Bardin, joiden on tarkoitus mullistaa tammamme työskennellä. (Spataro 16.3.2023; Pichai 6.2.2023.)

Microsoftin Copilot työskentelee rinnallasi ja on sisällytetty jokaiseen tyypillisesti käytettyyn Microsoftin applikaatioon kuten Word, Excel, Teams, PowerPoint ja Outlook. Copilot pääsee tarvittaessa käsiksi kaikkeen työssä hyödyntämääsi dataan kuten kalenterimerkintöihisi, palaveritietoihisi, muistiinpanoihisi ja yhteystietoihisi. Copilot voi esimerkiksi tiivistää sinun käymäsi videopalaverit, sähköpostikeskustelut ja muistiinpanot haluamastasi projektista tai aihealueesta yhdeksi selkeäksi dokumentiksi tai vaikka PowerPoint-esitykseksi. Microsoft painottaa, että käyttäjä on kuitenkin se, joka tekee lopullisen tuotoksen muokkaamalla Copilotilta saatua luonnosta haluamaansa muotoon. Tämä voi tarkoittaa esimerkiksi satunnaisten lauseiden muokkaamista vastaamaan enemmän käyttäjän omaa persoonaa, kun käyttäjä on pyytänyt Copilotia luomaan hänelle PowerPoint-esityksen jostain aiheesta. (Spataro 16.3.2023.)

Googlen Bardin on sen sijaan tarkoitus muun muassa antaa Googlen hakukenttään esitettyihin kysymyksiin suoraan vastaus, eli käyttäjän ei tarvitse välttämättä lähteä etsimään vastausta linkkien takaa (Pichai 6.2.2023). Google Bard hyödyntää Googlen omaa kehitystä olevaa PaLM2-kielimallia ja kykenee esimerkiksi kääntämään ohjelmakoodia toiselle kielelle, kuten Python koodia JavaScript koodiksi, Java koodia C# koodiksi ja niin edelleen (Google Research 10.5.2023, 00:45 – 01:00).

Mielestäni ChatGPT, Microsoft Copilot, Google Bard ja muut vastaavat tekoälyt ovat työkaluja, joita käyttämällä sovelluskehittäjän tuottavuus tulee nousemaan, ja tämän olen huomannut omalla kohdalla. 7 vuotta sovelluskehitystyötä tehnyt ja YouTube-sisältöä tuottava Travis on sitä mieltä, että kokeneen sovelluskehittäjän on ehdottomasti alettava hyödyntämään ChatGPT-tekoälyä työkaluna työssään, mikäli ei vielä näin tee. Hän tuo esiin sen, että tekoäly ei itsessään tule viemään sinulta työpaikkaa, mutta toinen sovelluskehittäjä, joka osaa hyödyntää tekoälyä tehokkaasti tulee viemään sinulta työpaikan. Travisin mielestä ChatGPT-tekoäly auttaa sinua tuottamaan parempaa ja selkeämpää koodia nopeammin. Sen sijaan vasta ohjelmointia opettelevan ihmisen ei pidä antaa

ChatGPT-tekoälyn kirjoittaa koodia puolestaan. Travis toteaa, että mikäli et ymmärrä ChatGPT-tekoälyn sinulle antamaa koodia ja et muun muassa kykene tekemään siihen muutoksia, ei sinun tulisi käyttää sitä tähän tarkoitukseen. Sen sijaan ohjelmointia opettelevalla aloittelijalla hyvä tapa hyödyntää ChatGPT-tekoälyä olisi kysyä apua koodin ymmärtämiseen ja pyytää ChatGPT-tekoälyä selittämään auki jonkin koodipätkän toiminnallisuus. (Travis Media 19.3.2023, 00:41 – 03:31.)

Olen Travisin kanssa samaa mieltä siitä, että ohjelmointiymmärryksen ollessa vielä aloittelijan tasolla, voi sovelluskehittäjä tehdä oppimisilleen haittaa hyödyntämällä ChatGPT-tekoälyä liiaksi. Tällä tarkoitan nimenomaan sitä, että pyydät ChatGPT-tekoälyä kirjoittamaan koodia puolestasi ymmärtämättä, mikä on ChatGPT-tekoälyn tarjoaman koodin logiikka ja miksi se toimii tai ei toimi. Sama pätee mielestäni lähes mihin tahansa asiaan: Jos toinen tekee kaiken puolestasi, et itse opi.

Itse taistelen tätä vastaan haastamalla aika ajoin itseni kirjoittamaan koodia täysin puhtaalta pöydältä ja kieltämällä itseltäni valmiin koodin kopioimisen ja erilaiset ohjelmointia nopeuttavat valmiiksi kirjoitetut koodipohjat (eng. template). Tämän kaltaiset harjoitukset virkistävät muistiani muun muassa käyttämäni ohjelmointikielen syntaksista eli kirjoitusasusta ja luovat mielestäni minulle vahvempaa osaamisen pohjaa.

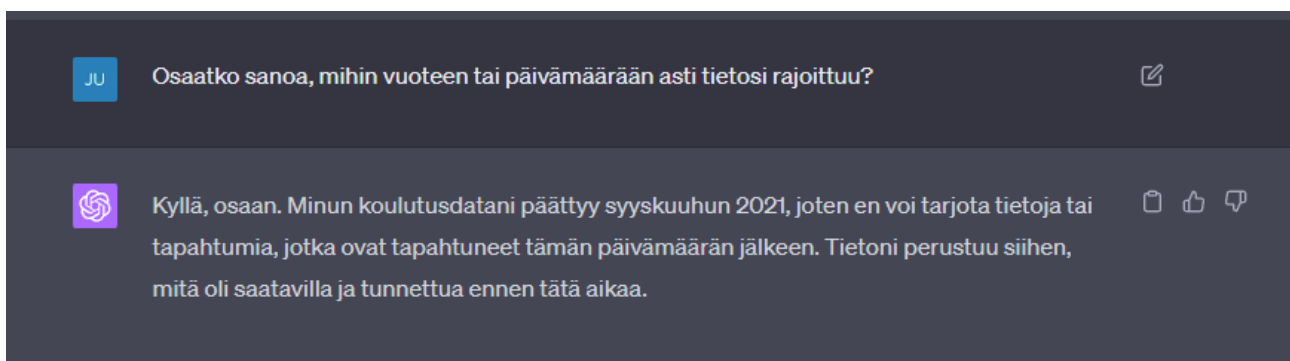
Suurimmaksi osaksi en kuitenkaan näe mitään syytä olla kopioimatta valmista koodia tai olla käyttämättä valmiiksi kirjoitettuja koodipohjia nopeuttaaksesi työskentelyäsi, mikäli kykenet tarvittaessa tekemään saman lopputuloksen itsenäisesti. Samalla tavalla ajattelen ChatGPT-tekoälyn käytöstä: Mikäli kykenisit kirjoittamaan saman koodin kuin tekoäly, mutta sinulla menisi aikaa useita tunteja, kun taas tekoälyltä menee tähän joitakin sekunteja, miksi et hyödyntäisi tätä työkalua? Sinun tehtäväksesi jää tekoälyn luoman koodin läpikäyminen, ymmärtäminen ja muokkaaminen. Työskentele tehokkaammin ja älykkäästi sen sijaan, että tekisit asiat hankalamman kautta. Englanniksi tämän voisi sanoa: "Work smarter, not harder".

Uskon näiden edellä mainittujen tekoälyjen tai niistä vielä kehittyneempien variaatioiden muokkaavan lähitulevaisuudessa tapojamme työskennellä ja siksi myös oman kehityksen kannalta haluan pysyä perässä näistä teknologioista ja työkaluista. On tärkeä myös ymmärtää, että ChatGPT-tekoälyllä on rajansa ja heikkoutensa ja ChatGPT-tekoälyä hyödyntävän henkilön olisi syytä tiedostaa tämä. Yksi olennainen heikkous on, että ainakin toistaiseksi ChatGPT-tekoälyn koulutukseen käytetty data päättyy vuoden 2021 syyskuuhun. Tämä tarkoittaa luonnollisesti sitä, että et voi olettaa ChatGPT-tekoälyn antavan sinulle oikeita tai järkeviä vastauksia kysymyksiin tuoreista aiheista kuten esimerkiksi viimeaikaiset uutiset, uudet tutkimukset tai uudet teknologiat.

Tieto siitä, että ChatGPT-tekoälyn tiedot loppuvat vuoden 2021 syyskuuhun ei ole suoraan nähtävillä sivuston käyttöliittymästä, jonka kautta käyttäjä tavallisesti keskustelee ChatGPT-tekoälyn

kanssa. Tämä voi mielestäni johtaa helposti tilanteisiin, jossa käyttäjä ei tunnista käyttämänsä tekoälyn rajoitteita ja sitä, mihin työkalua voi käyttää ja minkälaisiin kysymyksiin tekoäly ei kykene vastaamaan järkevästi. Tekoälyyn kriittisemmin suhtautuva kirjailija ja mediapersoonaa Adam Conover antaa YouTube-videossaan tästä esimerkin, jossa käyttäjä kysyy Bing-hakukoneelta, milloin on "Avatar 2"-elokuvan seuraava näytös. Microsoftin Bing-hakukoneeseen on lisätty ChatGPT-4-tekoäly, joka tarjoaa käyttäjälle vastauksia (Mehdi 14.3.2023). Bing-hakukone eli käytännössä tässä tapauksessa ChatGPT-4 kertoo, että "Avatar 2"-elokuva julkaistaan vasta tulevaisuudessa vuonna 2022. Käyttäjä luonnollisesti hämmäntyi, sillä elämme kuitenkin vuotta 2023. (Adam Conover 31.3.2023, 11:20 – 11:30.)

Mielestäni tämän kaltaisten rajoitteiden esiin tuominen selkeämmin käyttäjälle ennaltaehkäisisi näiden työkalujen vääränlaista käyttöä. Tieto siitä, että ChatGPT-tekoälyn harjoitusdata rajoittuu vuoden 2021 syyskuuhun löytyy muun muassa OpenAI dokumentaatiosta (OpenAI). Käyttäjä voi myös kysyä tämän kaltaista tietoa ChatGPT-tekoälystä ChatGPT-tekoälyltä itseltään, kuten on tehnyt kuvassa 52.



Kuva 52. ChatGPT-tekoäly vastaa kysymykseen koskien sen omaa koulutusdataa.

Kaiken kaikkiaan tämä opinnäytetyöprosessi on kehittänyt minua niin tekstintuottajana kuin sovel-luskehittäjänä. Tästä on hyvä lähteä eteenpäin ja aion seurata aktiivisesti, miten tämä ala tulee ke-hittymään ja minkälaisia muutoksia esimerkiksi tekoälyn kehitys tälle alalle mahdollisesti tuo.

Lähteet

Advolodkin N. 27.1.2022. React Testing Course for Beginners – Code and Test 3 Apps. Video.

Katsottavissa: <https://www.youtube.com/watch?v=8vfQ6SWBZ-U>. Katsottu 25.2.2023.

Apexcharts 2023. Heartmap Charts. Luettavissa: <https://apexcharts.com/react-chart-demos/heatmap-charts/>. Luettu: 7.3.2023.

Arunghosh 2022. react-grid-heatmap. Luettavissa: <https://github.com/arunghosh/react-grid-heatmap>. Luettu: 7.3.2023.

Beck, K., Beedle, M., Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D. 2021. Manifesto for Agile Software Development. Luettavissa: <https://agilemanifesto.org/>. Luettu 18.3.2023.

Bhandari A. 4.12.2015. Importance of to-do list in Web-Developers day to day life. Luettavissa: <https://www.sjinnovation.com/todo-list-as-a-web-developer>. Luettu: 5.3.2023.

Bootstrap team. Get started with Bootstrap. Luettavissa: <https://getbootstrap.com/docs/5.3/getting-started/introduction/>. Luettu: 10.3.2023.

Bri does things 20.4.2023. How to use ChatGPT to easily learn any skill you want. Video. Katsottavissa: <https://www.youtube.com/watch?v=MnDudvCyWpc>. Katsottu 15.4.2023.

Christopher C. 8.9.2023. The Deadline Effect with Christopher Cox Book Party. Video. Katsottavissa: <https://www.youtube.com/watch?v=XCitnLdyB8Y>. Katsottu: 9.4.2023.

Cosima M. 10.5.2022. Magical SVG Techniques. Luettavissa: <https://www.smashingmagazine.com/2022/05/magical-svg-techniques/>. Luettu: 10.4.2023.

Coyier C. 18.2.2014. How SVG Line Animation Works. Luettavissa: <https://css-tricks.com/svg-line-animation-works/>. Luettu: 31.3.2023.

Developedbyed. Learn To Build An SVG Animation With CSS. Video. Katsottavissa: <https://www.youtube.com/watch?v=gWai7fYp9PY>. Katsottu: 3.3.2023.

GeoPandas developers 2022. geopandas. GeoDataFrame. Luettavissa: <https://geopandas.org/en/stable/docs/reference/api/geopandas.GeoDataFrame.html>. Luettu: 12.3.2023.

Google Research 10.5.2023. Introducing PaLM 2, Google's next generation large language model | Research Bytes. Video. Katsottavissa: <https://www.youtube.com/watch?v=yAANQypgOo8>. Katsottu: 17.5.2023.

Grace Saunders E. 18.3.2016. If You Dread Deadlines, You're Thinking About Them All Wrong. Luettavissa: <https://hbr.org/2016/03/if-you-dread-deadlines-youre-thinking-about-them-all-wrong>. Luettu: 9.4.2023.

Greenberg R. 10.10.2019. Naming Convention — 9 Basic Rules for any Piece of Code. Luettavissa: <https://medium.com/wix-engineering/naming-convention-8-basic-rules-for-any-piece-of-code-c4c5f65b0c09>. Luettu: 12.3.2023.

Gupta D. 2018. Performance-Analysis. Luettavissa: <https://github.com/dg92/Performance-Analysis-JS#readme>. Luettu: 19.2.2023.

Halfon L. 5.2.2019. Beginner's guide for naming things in your code. Luettavissa: <https://blog.ta-boola.com/beginners-guide-for-naming-things-in-your-code/>. Luettu: 12.3.2023.

Hamedani M. 13.1.2023. ChatGPT Tutorial for Developers - 38 Ways to 10x Your Productivity. Video. Katsottavissa: <https://www.youtube.com/watch?v=sTeoEFzVNSc>. Katsottu 25.3.2023.

Indeed Editorial Team 25.6.2022. Why Deadlines Are Important. Luettavissa: <https://www.indeed.com/career-advice/career-development/why-are-deadlines-important>. Luettu: 9.4.2023.

Jain A. 25.1.2021. Naming Conventions 101 for developers. Luettavissa: <https://medium.com/swlh/naming-conventions-101-for-developers-8997bb96fd60>. Luettu: 12.3.2023.

Karakaya H. 2022. How to Convert HTML to PDF Using React. Luettavissa: <https://pspdfkit.com/blog/2022/how-to-convert-html-to-pdf-using-react/>. Luettu: 23.3.2023.

MDN Web Docs 2023a. Fills and Strokes. Luettavissa: https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial/Fills_and_Strokes. Luettu: 21.2.2023.

MDN Web Docs 2023b. Window.innerHeight. Luettavissa: <https://developer.mozilla.org/en-US/docs/Web/API/Window/innerHeight>. Luettu: 3.3.2023.

MDN Web Docs 2023c. Array.from(). Luettavissa: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/from. Luettu: 8.3.2023.

MDN Web Docs 2023d. Set. Luettavissa: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set. Luettu: 8.3.2023.

MDN Web Docs 2023e. Array.prototype.flatMap(). Luettavissa: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/flatMap. Luettu: 8.3.2023.

MDN Web Docs 2023f. rgb(). Luettavissa: https://developer.mozilla.org/en-US/docs/Web/CSS/color_value/rgb. Luettu: 10.3.2023.

MDN Web Docs 2023g. The WebSocket API (WebSockets). Luettavissa: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API. Luettu: 13.3.2023.

MDN Web Docs 2023h. z-index. Luettavissa: <https://developer.mozilla.org/en-US/docs/Web/CSS/z-index>. Luettu: 31.3.2023.

MDN Web Docs 2023i. SVG Tutorial. Luettavissa: <https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial>. Luettu: 10.4.2023.

MDN Web Docs 2023j. SVG API. Luettavissa: https://developer.mozilla.org/en-US/docs/Web/API/SVG_API#svg_element_interfaces. Luettu: 10.4.2023.

MDN Web Docs 2023k. SVG Tutorial. Luettavissa: <https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial>. Luettu: 10.4.2023.

MDN Web Docs 2023L. Community for a better web. Luettavissa: <https://developer.mozilla.org/en-US/community>. Luettu: 15.4.2023.

Mehdi 14.3.2023. Confirmed: the new Bing runs on OpenAI's GPT-4. Luettavissa: https://blogs.bing.com/search/march_2023/Confirmed-the-new-Bing-runs-on-OpenAI%E2%80%99s-GPT-4. Luettu: 18.5.2023.

Meta pen Source 2023. Step 1: Break the UI into a component hierarchy. Luettavissa: <https://react.dev/learn/thinking-in-react>. Luettu: 21.3.2023.

Meta Platforms 2023a. Jest is a delightful JavaScript Testing Framework with a focus on simplicity. Luettavissa: <https://jestjs.io/>. Luettu: 20.2.2023.

Meta Platforms 2023b. Using Matchers. Luettavissa: <https://jestjs.io/docs/using-matchers/>. Luettu: 20.2.2023.

Microsoft 10.11.2022. What is Azure DevOps? Luettavissa: <https://learn.microsoft.com/en-us/azure/devops/user-guide/what-is-azure-devops?view=azure-devops>. Luettu: 13.3.2023.

Microsoft 9.10.2020. Introduction to SignalR. Luettavissa <https://learn.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr>. Luettu: 14.3.2023.

Microsoft 2023. What is edge computing? Luettavissa: <https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-edge-computing/?cdn=disable>. Luettu: 20.3.2023.

Microsoft 2023b. What is IoT? Luettavissa: <https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-iot/#overview>. Luettu: 20.3.2023.

Mind Tools Content Team. To-Do Lists. Luettavissa: <https://www.mindtools.com/aug8p7g/to-do-lists>. Luettu: 5.3.2023.

Nivo. HeatMap. Luettavissa: <https://nivo.rocks/heatmap/>. Luettu: 7.3.2023.

OpenAI. GPT-3.5. Luettavissa: <https://platform.openai.com/docs/models/gpt-3-5>. Luettu: 18.5.2023.

OpenAI 30.11.2022. Introducing ChatGPT. Luettavissa: <https://openai.com/blog/chatgpt>. Luettu: 15.4.2023.

Pichai, S. 6.2.2023. An important next step on our AI journey. Luettavissa: <https://blog.google/technology/ai/bard-google-ai-search-updates/>. Luettu: 10.5.2023.

Pluralsight 2023. The technology workforce development company. Luettavissa: <https://www.pluralsight.com/about>. Luettu 23.3.2023.

Pujari V. 27.5.2020. A Guide for Naming Things in Programming. Luettavissa: <https://leवलup.gitconnected.com/a-guide-for-naming-things-in-programming-2dc2d74879f8>. Luettu: 12.3.2023.

React-Bootstrap. Rebuilt with React. Luettavissa: <https://react-bootstrap.github.io/>. Luettu: 10.3.2023.

React-Bootstrap b. Grid system. Luettavissa: <https://react-bootstrap.github.io/layout/grid/>. Luettu: 15.3.2023.

React-Bootstrap c. Tabbed components. Luettavissa: <https://react-bootstrap.github.io/components/tabs/>. Luettu: 23.3.2023.

React-Redux 2022. useSelector(). Luettavissa: <https://react-redux.js.org/api/hooks>. Luettu: 2.3.2023.

Redux-toolkit 2022. createSlice. Luettavissa: <https://redux-toolkit.js.org/api/createSlice>. Luettu: 3.3.2023.

Recharts Group 2023. Recharts. Luettavissa: <https://recharts.org/en-US>. Luettu: 21.3.2023.

Recharts Group 2023. PieChartWithCustomizedLabel. Luettavissa: <https://recharts.org/en-US/examples/PieChartWithCustomizedLabel>. Luettu: 4.4.2023.

Red Hat 2022. What is a CI/CD pipeline? Luettavissa: <https://www.redhat.com/en/topics/devops/what-cicd-pipeline>. Luettu 25.2.2023.

Seibert C. 24.12.2022. Is ChatGPT actually useful for coding? Video. Katsottavissa: <https://www.youtube.com/watch?v=B3yuK2XHmvM>. Katsottu: 25.3.2023.

Spataro, J. 16.3.2023. Introducing Microsoft 365 Copilot – your copilot for work. Luettavissa: <https://blogs.microsoft.com/blog/2023/03/16/introducing-microsoft-365-copilot-your-copilot-for-work/>. Luettu: 10.5.2023.

Stackoverflow. Who we are. Luettavissa: <https://stackoverflow.co/>. Luettu: 15.4.2023.

Tarleton S. 2023. Use map() instead of for() loops. Luettavissa: <https://blog.bitsrc.io/please-use-map-instead-of-for-loops-5a2f54f088c8>. Luettu: 19.2.2023.

Techsolutionstuff 25.7.2022. How To Convert HTML To PDF Using JavaScript. Luettavissa: <https://dev.to/techsolutionstuff/how-to-convert-html-to-pdf-using-javascript-1a3p>. Luettu: 23.3.2023.

TestingXperts 2022. What is Functional Testing? Luettavissa: <https://www.testingxperts.com/blog/functional-testing>. Luettu: 5.3.2023.

Traverse Media 30.8.2021. React Testing Crash Course. Video. Katsottavissa: <https://www.youtube.com/watch?v=OVNjst09xM>. Katsottu: 20.2.2023.

Travis Media 19.3.2023. REAL Uses of ChatGPT As A Developer | 12 Practical Examples. Video. Katsottavissa: <https://www.youtube.com/watch?v=k59FJ3NzD7s>. Katsottu: 17.5.2023.

Tsui T. 17.1.2023. Coding with chatGPT. Luettavissa: <https://medium.com/@tanyamarleytsui/coding-with-chatgpt-b50ab3fcb45f>. Luettu: 12.3.2023.

Uiwjs 2022. react-heat-map. Luettavissa: <https://github.com/uiwjs/react-heat-map>. Luettu 7.3.2023.

Worldometer. Current World Population. Luettavissa: <https://www.worldometers.info/world-population/>. Luettu: 15.4.2023.

Liitteet

Liite 1. Terminologia

Azure DevOps

Microsoftin ylläpitämä ja tarjoama pilvipalvelu, joka sisältää joukon erilaisia työkaluja sovelluskehityksen tarpeisiin tai projektihallinnan tarpeisiin (Microsoft 10.11.2022)

Back end

Se osa sovelluksen koodista, jonka kanssa käyttäjä ei ole suoraan tekemisissä. Back end -koodi huolehtii muun muassa sovelluksen tietokantatoiminnoista, kuten tiedon tallentaminen tietokantaan ja tiedon haku tietokannasta. Back end -koodi käsittelee käyttäjän käyttöliittymän kautta lähettämiä pyyntöjä ja palauttaa niihin vastauksia. Kirjoitetaan usein myös yhteen ”backend”, vaikka tämä ei olekaan Suomessa kieliopillisesti oikein.

Bootstrap

Bootstrap on avoimen lähdekoodin CSS- ja JavaScript-kehys, joka sisältää valmiiksi rakennettuja komponentteja ja ominaisuuksia verkkosivujen tai web-sovellusten kehitykseen ja tyyllittelyyn. Bootstrapin valmiilla komponenteilla voi rakentaa responsiivisia käyttöliittymiä vaivattomasti. Responsiivinen käyttöliittymä tarkoittaa käyttöliittymän mukautumista eri päätelaitteille, kuten mobiililaitteille tai tietokoneille.

ChatGPT-tekoäly

OpenAI yrityksen luoma tekoäly, joka perustuu suurten kielimallien (eng. large language model) hyödyntämiseen. GPT tulee sanoista ”Generative pre-trained transformer”, joka on tietyn tyyppinen kielimalli. (ByteByteGo 24.4.2023, 00:35 – 00:46.) ChatGPT-tekoäly on älykäs keskustelubotti, jonka kanssa henkilö voi kommunikoida tavalliseen tapaan ja ChatGPT-tekoäly tarjoaa henkilölle älykkäitä vastauksia lähestulkoon mistä tahansa aiheesta, johon voisit löytää vastauksen Internetistä. (OpenAI 30.11.2022.)

CSS

Tulee sanoista Cascading Style Sheet ja on front end -ohjelmoinnissa käytetty kieli sovelluksen tai verkkosivujen tyyllittelyyn. Tyyllittelyllä tarkoitetaan esimerkiksi fontin kokomuutoksia, elementtien värimuutoksia tai yksinkertaisia animaatioita.

Front end

Se osa sovelluksen tai verkkosivujen koodista, joka tuottaa käyttäjän nähtäville visuaaliset elementit kuten painikkeet, ja jonka kautta käyttäjä käyttää sovellusta tai verkkosivuja. Front end sisältää usein HTML-, CSS- sekä JavaScript-kielen koodia, joiden avulla käyttöliittymä luodaan. Front end voi sisältää logiikkaa ja erilaista laskentaa, kuten Back end, mutta raskas logiikka Front endin puolella on sovellukselle kuormittavampaa, kuin sen sijoittaminen Back end -puolelle. Kirjoitetaan usein myös yhteen ”frontend”, vaikka tämä ei olekaan Suomessa kieliopillisesti oikein.

Funktio

Ohjelmoidessa funktiolla viitataan koodilohkoon, joka suorittaa jonkin sille asetetun toiminnon ja tarvittaessa palauttaa jonkin vastauksen suoritettuaan tämän toiminnon. Esimerkiksi voisimme luoda funktion nimeltä ”laske”, ja antaa tälle tehtäväksi laskea $x + y$ ja palauttaa meille näiden lukujen summa. Voimme tämän jälkeen käyttää funktiota ”laske” laskeaksemme mitkä tahansa kaksi lukua yhteen.

Get-pyyntö

Front endin ja back endin välisessä kommunikaatiossa käytetty tiedonsiirtometodi, jolla haetaan tietoa käyttöliittymän käyttöön palvelimelta (back end).

HTML

”Hyper Text Markup Language” eli HTML on verkkosivujen rakentamiseen käytetty kieli. HTML avulla elementit, tekstit, painikkeet luodaan ja näitä HTML kielen avulla luotuja elementtejä tyypillisesti sommitellaan ja tyylitellään CSS-kielen avulla.

JavaScript

Ohjelmointikieli, jota käytetään tyypillisesti Front end ohjelmoinnissa yhdessä HTML- ja CSS-kielen kanssa. HTML luo elementtejä, CSS tyylittelee elementit ja JavaScriptillä luomme logiikan painikkeille ja sovelluksen käyttöliittymän toiminnoille.

Jest

JavaScript-testauskehys, jonka avulla voidaan luoda ohjelmatestausta sovelluksen front endille. Testauksella pyritään varmistamaan muun muassa sovelluksen toimivuutta.

JSON

”JavaScript Object Notation” eli lyhennettynä JSON on tapa tallentaa ja liikuttaa dataa tietynlaisessa muodossa. JSON on aina tekstiä, joten sitä voidaan käsitellä lähtökohtaisesti kaikilla ohjelmointikielillä ja JSON-muotoinen data on ihmisille helppolukuista.

Kehys

Englanniksi käytetään termiä ”framework”. Kehys on oikeastaan joukko erilaisia työkaluja, valmiiksi kirjoitettuja toimintoja ja koodia, joita sovelluskehittäjä voi hyödyntää nopeuttaakseen ja helpottaakseen työskentelyään. Esimerkiksi ”React” on JavaScript-kehys, jota käytetään rakentamaan sovelluksen käyttöliittymiä. Samanlaiseen lopputulokseen pääsee myös ilman kehyksiä, mutta tällöin joutuu käyttämään tyypillisesti enemmän aikaa, kuin kehystä hyödyntäessä.

Laiteyksikkö

Tässä opinnäytetyössä laiteyksiköllä viitataan joukkoon erilaisia laitteita, jotka yhdessä muodostavat kokonaisuuden, jota kutsun laiteyksiköksi. Teollisuuden laiteyksikköön voisi sisältyä esimerkiksi kaasuanalysaattori, näytteen viilentäjä, erilaisia antureita ja teollisuustietokone.

Lähdekoodi

Sovelluksen ohjelmakoodi tekstimuotoisena. Kun puhutaan avoimen lähdekoodin sovelluksesta, tarkoitetaan sovellusta, jonka koko ohjelmakoodi on vapaasti saatavilla ja täten käyttäjä voi tarpeen tullen tehdä itse muutoksia sovelluksen lähdekoodiin. Esimerkiksi käyttöjärjestelmä Linux on avoimen lähdekoodin käyttöjärjestelmä ja Linuxista on saatavilla valtava määrä erilaisia versioita erilaisten käyttäjäkuntien kehittämänä.

Objekti

Käytetään myös termiä Olio. Objekti on keskeinen käsite ohjelmoinnissa. Sillä tarkoitetaan käytännössä sitä, että joukko erilaisia ominaisuuksia ja toimintoja sidotaan yhteen yhden objektin alle. Esimerkkinä voimme kuvitella auton olevan objekti, joka sisältää joukon ominaisuuksia kuten punainen väri ja kesärenkaat ja joukon toimintoja kuten kaasu ja jarru. JavaScript-objekti on tyypillisesti joukko erilaisia tallennettuja arvoja.

Ohjelmistokehittäjä

Henkilö, joka suunnittelee ja kehittää ohjelmia sekä sovelluksia. Ohjelmistokehittäjällä voi olla oma erikoisosaamisensa tai tehtävänsä, kuten sovelluksen front end tai back end -kehitys. Ohjelmistokehittäjä hallitsee tyypillisesti useita erilaisia ohjelmistokehityksen teknologioita ja mahdollisesti myös ohjelmointikieliä.

Rajapinta

Ohjelmistokehityksessä rajapinnalla tyypillisesti tarkoitetaan tapaa, jolla määritetään, kuinka sovelluksen back end ja front end kommunikoivat keskenään. Rajapinta on siis ikään kuin portti, jonka kautta voidaan tehdä tiedonsiirtoa. Ohjelman rajapinnat voivat olla myös julkisia, jolloin sovelluskehittäjät ympäri maailman pystyvät tekemään pyyntöjä kyseiseen rajapintaan. Yleensä rajapinta kuitenkin suojataan jonkin tasoisella käyttäjätodennuksella (eng. authentication), jotta kuka tahansa ei kykene hakemaan tietoa rajapinnan kautta.

Refaktorointi

Refaktoroinnilla tarkoitetaan olemassa olevan koodin muokkaamista tarkoituksena parantaa sen luettavuutta ja/tai ylläpidettävyyttä ja/tai suorituskykyä.

React

React on avoimen lähdekoodin JavaScript-kirjasto, joka on suunniteltu käyttöliittymien rakentamisen helpottamiseksi.

Redux

Redux on avoimen lähdekoodin JavaScript-kirjasto sovelluksen tilan hallintaan.

Sovelluskehitys

Prosessi, jossa pyritään luomaan ohjelma tai sovellus. Sovelluksia ja ohjelmia on nykyään erilaisille laitteille, kuten tietokoneille ja älypuhelimille. Riippumatta päätelaitteesta sovelluksen kehittäminen kuuluu aina sovelluskehityksen piiriin.

SVG

Tulee sanoista "Scalable Vector Graphics" ja tarkoittaa vektorigrafiikkaa. Vektorigrafiikalla luodun kuvan etu tavanomaiseen kuvaan verrattuna on se, että vektorigrafiikalla luodun kuvan kokoa voidaan huoletta pienentää tai suurentaa aiheuttamatta kuvaan vääristymiä. Kun tavanomaista kuvaa suurennetaan, kuva yleensä pikselöityy eli muuttuu epätarkaksi. SVG-kuva pysyy aina tarkkana.

Web-kehitys

Web-kehitys viittaa verkkosivujen tai verkkosovelluksien kehittämiseen. Mikäli Web-kehittäjä kehittää sovellusta, on hän myös sovelluskehittäjä. Web-kehitys on siis sovelluskehityksen alakategoria.

Liite 2. Viikkoanalyysi 1 Testaus

Testauksen koodi - <https://www.measurethat.net/Benchmarks/Show/23614/0/filtervastaanmap>

Testauksen tulokset - <https://www.measurethat.net/Benchmarks/ListResults/23614>