Tampereen ammattikorkeakoulu

Verkkosovellus puhelimien elinkaarisovelluksen työnkulun määrittelyyn

Tuukka Juusela

OPINNÄYTETYÖ
Lokakuu 2020

Tietojenkäsittely
Ohjelmistotuotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Ohjelmistotuotanto

JUUSELA, TUUKKA:
Verkkosovellus puhelimien elinkaarisovelluksen työnkulun määrittelyyn

Opinnäytetyö 66 sivua, joista liitteitä 1 sivua
Lokakuu 2020

Tämän opinnäytetyön tarkoituksena oli toteuttaa uudelleen suunniteltu työnkulun hallintajärjestelmä Piceasoft Oy:n puhelimien elinkaarisovellukselle yksisivuisena verkkosovelluksena. Piceasoft Oy:llä oli olemassa aikaisempia toteutuksia työnkulun hallintaan, mutta asiakasvaatimuksista nousi tarve kehittää viimeisintä järjestelmää eteenpäin. Edellinen toteutus oli toteutettu Piceasoft Oy:n asiakkuudenhallintajärjestelmään monisivuisena verkkosovelluksena. Uusi työnkulun hallintajärjestelmä toteutettiin yksisivuisena verkkosovelluksena sen tarjoamien etujen ja aikaisempien hyvien kokemusten vuoksi.

Opinnäytetyössä tutkittiin aluksi, miten yksisivuiset verkkosovellukset toimivat verrattuna monisivuisiin verkkosovelluksiin ja mitä etuja ne tarjoavat. Työssä perehdyttiin tämän jälkeen siihen, miten vanha työnkulun hallintajärjestelmä toimi ja miten toteutettava järjestelmä toimisi. Opinnäytetyössä kuvattiin, miten projektin toteutus eteni ja tutustuttiin toteutetun sovelluksen ominaisuuksiin. Lopuksi opinnäytetyössä perehdyttiin siihen, miten toteutettu yksisivuinen verkkosovellus toimii teknisesti ottaen.

Työnkulun hallintajärjestelmä toteutettiin keväällä 2020 projektina, johon osallistui myös muita Piceasoft Oy:n työntekijöitä. Opinnäytetyöntekijä vastasi verkkosovelluksen toteutuksesta ja muut työntekijät toimivat mm. projektin tietokannan ja elinkaarisovelluksen työnkulun hallinnan toteuttamisen parissa. Uusi työnkulun hallintajärjestelmä julkaistiin 10.6.2020, jonka jälkeen verkkosovellukseen lisättiin vielä lisäominaisuuksia, ja verkkosovelluksen käyttöliittymää paranneltiin.

Asiakasvaatimukset esittänyt asiakas on ottanut julkaisun jälkeen uuden työnkulun hallintajärjestelmän onnistuneesti käyttöönsä. Uudesta järjestelmästä on saatu hyvää palautetta ja sitä on kehuttu helppokäyttöiseksi.

Toteutettu yksisivuinen verkkosovellus vastaa sille asetettuja tavoitteita. Toteutettua verkkosovellusta voidaan kehittää vielä lisää parantelemalla verkkosovelluksessa käytettyä sovellusarkkitehtuuria.

Asiasanat: verkkosovellus, react, sovelluskehitys

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Option of Software Production

JUUSELA, TUUKKA:
Implementation of Web Application for Managing Mobile Device Lifecycle Application's Workflows

Bachelor's thesis 66 pages, appendices 1 pages
October 2020

The purpose of this Bachelor's thesis was to implement redesigned single page web application system for workflow management of Piceasoft Ltd.'s mobile device lifecycle application. Piceasoft had previous versions of managing workflows of the lifecycle application, but new customer requirements brought about the need to redesign the system. The previous implementation of workflow management system had been implemented to Piceasoft Ltd.'s customer management system as multipage web application. The new workflow management system was decided to be implemented as single page web application, because of advantages it brought compared to multipage web applications and previous good experiences with them.

This Bachelor's thesis studied how single page web applications differ from their multipage counterparts and what advantages they bring compared to it. It examined how the old workflow management system worked, how the implemented system would work compared to the old system, how the project went and studied of features of the new workflow management system. The thesis also took an in depth look at how the implemented system works from technical point of view.

The new workflow management system was implemented in the spring of 2020. Other Piceasoft Ltd.'s employees worked on for example on database and life cycle application side of the workflow management system. New workflow management system was released in June, after which couple more features were added and UI improvements made.

The implemented web application meets its requirements. The web application could be improved in the future in terms of its application hierarchy.

Key words: web application, react, software development

SISÄLLYS

ERITYISSANASTO

verkkosovellus	sovellus, jota käytetään verkkoselaimen kautta
yksisivuinen verkkosovellus	sovellus, jonka toiminta keskittyy yhdelle sivulle, jota päivitetään selaimessa
monisivuinen verkkosovellus	sovellus, jonka toiminta on jaoteltu monelle eri sivulle, joita haetaan palvelimelta
Asiakkuudenhallintajärjestelmä	Piceasoft Oy:n järjestelmä, jolla hallinnoidaan PiceaServicesin asiakkuuksia ja asiakkuuksien tuotteita
PMC	Product Management Console. ks. asiakkuudenhallintajärjestelmä
Frontend	käyttäjälle näkyvä osuus sovelluksesta, jota hän käyttää itse. Työnkulun hallintajärjestelmän tapauksessa verkkosovellus, joka on käyttäjän selaimessa
Backend	käyttäjälle näkymätön osuus, jota hän ei käytä suoraan itse. Työnkulun hallintajärjestelmän tapauksessa rajapinnat ja tietokanta, johon käyttäjän selaimessa toimiva frontend ottaa yhteyden
PiceaServices	Piceasoft Oy:n puhelimien elinkaarisovellus
Volume	PiceaServicesissä oleva ominaisuus, jolla voidaan ajaa Verify, Diagnostics ja Erasure samanaikaisesti kymmenille puhelimille

Verify	puhelimien myyntivalmiuden varmistamiseen käytetty sovellus
Diagnostics	puhelimien kunnon varmistamiseen tehty sovellus
Erasure	puhelimien muistin pyyhintään käytetty sovellus
Työnkulun hallintajärjestelmä	asiakkuudenhallintajärjestelmässä toimiva sovellus, jolla voidaan muokata Volumen toimintaa PiceaServicesissä

1 JOHDANTO

Opinnäytetyö tehtiin Piceasoft Oy:lle, joka tuottaa ohjelmistoja puhelimien elinkaaren hallintaan. Tuotteiden ja asiakkaiden hallintaan on kehitetty verkossa toimiva asiakkuudenhallintajärjestelmä, josta voidaan hallinnoida asiakkaiden tuotteita, heidän tietojaan ja muuta asiakkuuksiin liittyvää. Yksi yrityksen tuotteista on Volume, joka on puhelimien massadiagnosointiin ja -puhdistamiseen tehty ohjelmisto. Volume yhdistää yrityksen muut ohjelmat yhdeksi suureksi kokonaisuudeksi, jossa kymmeniä puhelimia voidaan operoida kerralla vaivattomasti. Volumen työnkulku eli operaatiot, jotka puhelimelle suoritetaan, voidaan muokata konfiguraatioiden avulla. Puhelimien operaatioiden suoritusta voidaan muokata valitsemalla mitkä operaatiot puhelimelle suoritetaan, miten mahdollisten testien epäonnistuminen käsitellään ja mitä puhelimia työnkulun määrittely koskee.

Piceasoft Oy:lla on aikaisempia versioita asiakkuudenhallintajärjestelmässään Volumen työnkulun määrittelemisestä, mutta ne ovat osoittautuneet riittämättömiksi. Uusien asiakasvaatimuksien ja -tarpeiden pohjalta suunniteltiin Volumen työnkulku uudelleen. Vaihtoehtoina oli tehdä vanhaan työnkulun hallintajärjestelmään suuria muutoksia tai uusia koko järjestelmä.

Piceasoft Oy päätyi toteuttamaan uuden Volumen työnkulun hallintajärjestelmän asiakkuudenhallintajärjestelmään yksisivuisena verkkosovelluksena. Uuden järjestelmän toteuttamiseen yksisivuisena verkkosovelluksena päädyttiin aikaisempien positiivisten kokemuksien ja yksisivuisten verkkosovelluksien tuomien hyötyjen takia. Päätökseen vaikutti myös opinnäytetyöntekijän aikaisempi kokemus yksisivuisten verkkosovellusten toteutuksesta ja Piceasoft Oy:n suunnitelma tehdä samaan yksisivuiseen verkkosovelluspohjaan muita saman tyyppisiä järjestelmiä.

Yksi työnkulku pitää sisällään kolme eri konfiguraatiota suoritettaville operaatioille. Nämä konfiguraatiot kertovat Piceasoft Oy:n PiceaServices-ohjelmalle, miten eri Volumen operaatiot tulee suorittaa. Konfiguraatioita ja työnkulkuja voi-

daan erikseen muokata työnkulunhallintajärjestelmässä niille tarkoitetuissa näkymissä, missä ne listataan. Molempia voidaan poistaa sekä muokata luonnin jälkeen. Uudet työnkulut ovat heti käytettävissä PiceaServices-ohjelmassa niiden luonnin jälkeen.

Vanha Volumen työnkulun hallintajärjestelmä oli toteutettu monisivuisena verkkosovelluksena, jolloin verkkosovellus toimii monella verkkosivulla selaimessa. Selain hakee tällöin aina uusien verkkosivujen sisällön käyttäjän selaimeen palvelimelta käyttäjän navigoidessa uudelle sivulle.

Yksisivuiset verkkosovellukset mahdollistavat koko sovelluksen toimimisen kokonaisuudessaan vain yhdellä verkkosivulla. Sivun sisältöä muokataan JavaScript-ohjelmalla, joka pystyy muokkaamaan sivun tekstiä, kuvia ja muuta sisältöä ohjelmallisesti selaimessa. Käyttäjän selain hakee siis vain yhden HTML-sivun palvelimelta käyttäjän avatessa verkkosovelluksen, mikä vähentää käytävää datan vaihdon määrää palvelimen ja käyttäjän selaimen välillä. Yksisivuisten verkkosovellusten etuna on työpöytäsovelluksen kaltainen toiminnallisuus, jossa sovellus vastaa käyttäjän syötteisiin melkein heti ja näyttää uuden sisällön nopeasti.

Toteutettu yksisivuinen verkkosovellus toteutettiin React-sovelluskirjastoa käyttäen. React-sovellukset koostuvat tilallisista ja tilattomista komponenteista. Komponentit pitävät sisällään sovelluslogiikkaa sekä piirtologiikan. Tilalliset komponentit pystyvät säilömään ja muokkaamaan tietoja sisällään erillisesti muista komponenteista, kun taas tilattomat komponentit piirtävät itsensä sen perusteella, mitä tietoja niille annetaan.

Volumen työnkulun hallintajärjestelmää varten perustettiin oma React-sovellus, joka pystyy hallinnoimaan myös muita vastaavia hallintajärjestelmiä. Itse työnkulun hallintajärjestelmässä toteutettiin piirrettävät ponnahdusikkunat epätyypillisellä tavalla React-sovellukselle, koska komponenttien tilaa haluttiin käsitellä keskitetysti yhdessä tiedostossa.

Uusi Volumen työnkulun määrittely ideoitiin ja suunniteltiin uusien vaatimuksien perusteella. Työnkulun määrittelyn käyttöliittymä ja toiminnallisuus on hyväksytetty sisäisesti Picesoft Oy:lla ja asiakkailla ennakkoon, jotta uusi versio työnkulun määrittelystä vastaa kaikkiin tarvittaviin osa-alueisiin.

Projektin toteuttamisesta vastasi Piceasoft Oy:n eri tiimien jäsenistä koottu ryhmä, jossa opinnäytetyöntekijä oli mukana palvelintiimin jäsenenä. Piceasoft Oy:n muut vanhemmat työntekijät vastasivat tietokannan, rajapintojen sekä elinkaarisovelluksessa toimivan työnkulun toimivuudesta. Käyttöliittymät suunnitteli Piceasoft Oy:n UX-suunnittelija.

Projektin alkaessa pidettiin etäpalaveri, jossa suunniteltiin projektin alustava toteutus. Projektin aikana pidettiin seurantapalavereita, joissa keskusteltiin työn etenemisestä ja erinäisistä työhön liittyvistä kysymyksistä. Loppuvaiheessa työnkulun hallintajärjestelmää myös esiteltiin ennakkoon asiakkaalle, joka antoi positiivista palautetta. Julkaisun jälkeen verkkosovellukseen päätettiin lisätä vielä käyttöliittymäparannuksia ja asiakkaan pyytämiä lisäominaisuuksia.

Toteutettu verkkosovellus on otettu käyttöön asiakkaille onnistuneesti julkaisun jälkeen.

Opinnäytetyön teoriaosuudessa tutkitaan, mitä eroja yksisivusisilla verkkosovelluksilla on monisivuisiin verkkosovelluksiin verrattuna. Teoriaosuus selventää, miksi yksisivuisten verkkosovelluksien suosio on nousussa ja mitä etuja ne tarjoavat verrattuna monisivuisiin. Yksisivuisten verkkosovellusten edut olivat yhtenä perusteena, miksi Piceasoft Oy päätti toteuttaa uuden työnkulun hallintajärjestelmän yksisivuisena verkkosovelluksena.

Työnkulun hallintajärjestelmä -kappaleessa tutustutaan uuden järjestelmän tarpeisiin ja syihin, miksi uusi järjestelmä päätettiin toteuttaa. Lopuksi kappaleessa tutustutaan toteutetun järjestelmän ominaisuuksiin.

Tekninen toteutus -kappaleessa tutustutaan lyhyesti asiakkuudenhallintajärjestelmän ympäristöön, mihin uusi työnkulunhallintajärjestelmä on toteutettu. Kappaleessa selvennetään yksisivuisen verkkosovelluksen toteuttamiseen käytetyn

sovelluskirjasto Reactin toimintaperiaatteita lyhyesti. Kappaleessa tutustutaan lopuksi teknisestä näkökulmasta siihen, miten sovelluksen eri osat toteutettiin.

2 YKSISIVUISET VERKKOSOVELLUKSET

2.1 Mikä on yksisivuinen verkkosovellus?

2.1.1 Verkkosovellukset ja niiden tyypit

TEPA-termipankki määrittelee (TEPA-termipankki n.d) verkkosovelluksen tietokoneen selaimessa käytettäväksi sovellukseksi. Sovelluksen TEPA-termipankki määrittelee käyttäjän käytettäväksi ohjelmaksi, joka toteuttaa tietyn tehtävän. Tärkein ero verkkosovelluksen ja tavallisen tietokonesovelluksen tai puhelimen sovelluksen välillä onkin siis se, että sovellusta käytetään verkkoselaimen kautta. Tämä tarkoittaa sitä, että käyttäjän ei tarvitse erikseen ladata ja asentaa laitteelleen sovellusta pystyäkseen käyttämään sitä. Kaikki sovelluksen toiminnallisuus tapahtuu siis käyttäjän verkkoselaimessa, ilman erillistä ohjelmistoa.

Verkkosovellus saattaa olla monelle tuntemattomampi termi, mutta termi ”verkkosivu” on mahdollisesti monelle tuttu. Verkkosivun ja verkkosovelluksen ero on hämärä, ja monet määrittelevät ne eri tavoin. Mary MacPherson (2019) on määritellyt verkkosovelluksen ja verkkosivun eroksi sen, että verkkosivujen sisältö ei päivity dynaamisesti. MacPherson (2019) selventää eroja vielä kuvaamalla, että verkkosivut ovat yksisuuntaisia tiedonlähteitä, jotka eivät mahdollista vuorovaikutusta käyttäjien ja sivun välillä.

Oikeusministeriön omistama oikeudellisen aineiston julkinen palvelu Finlex on hyvä esimerkki verkkosivusta. Finlexissä käyttäjä voi mm. selata lainsäädäntöön liittyviä asioita, katsoa hallituksen esityksiä tai hakea Finlexin ainestoista haluamaansa tietoa. Käyttäjällä ei ole siis mitään vuorovaikusta sivun sisältöön. Hän ei pysty tuottamaan sisältöä tai vaikuttamaan siihen, eli sivu toimii vain tietoportaalina käyttäjälle. Monen todennäköisesti tuntema ja käyttämä Facebook osuu TEPA-termipankin määritelmään hyvin verkkosovelluksesta. Facebookin ominaisuuksiin kuuluu mm. käyttäjien profiilit, joissa he voivat jakaa kiinnostuksiaan, kuviaan tai tehdä päivityksiä elämästään ja tarkastella muiden profiileja ja päivityksiä. Facebookissa käyttäjä voi siis tehdä erinäisiä toimintoja sivustolla, kuten kuvien tai päivitysten lisäämistä, mikä tekee Facebookista verkkosovelluksen.

Amelia Staff (2019) kertoo, että verkkosivutyyppejä on siis kahden tyyppisiä: staattisia ja dynaamisia. Staattisissa sivuissa sisältö ei muutu ja se on jokaiselle käyttäjälle samanlainen. Dynaamisissa sivuissa sisältöä voidaan muuttaa ja sivulla pystytään mahdollistamaan käyttäjäinteraktio. Dynaamisen sivun sisältöä voidaan muuttaa joko palvelimella tai käyttäjän selaimessa. (Staff 2019.) Verkkosovellusten toimintaperiaate perustuukin dynaamisiin sivuihin, joiden sisältöä voidaan muokata tarvittaessa käyttäjille.

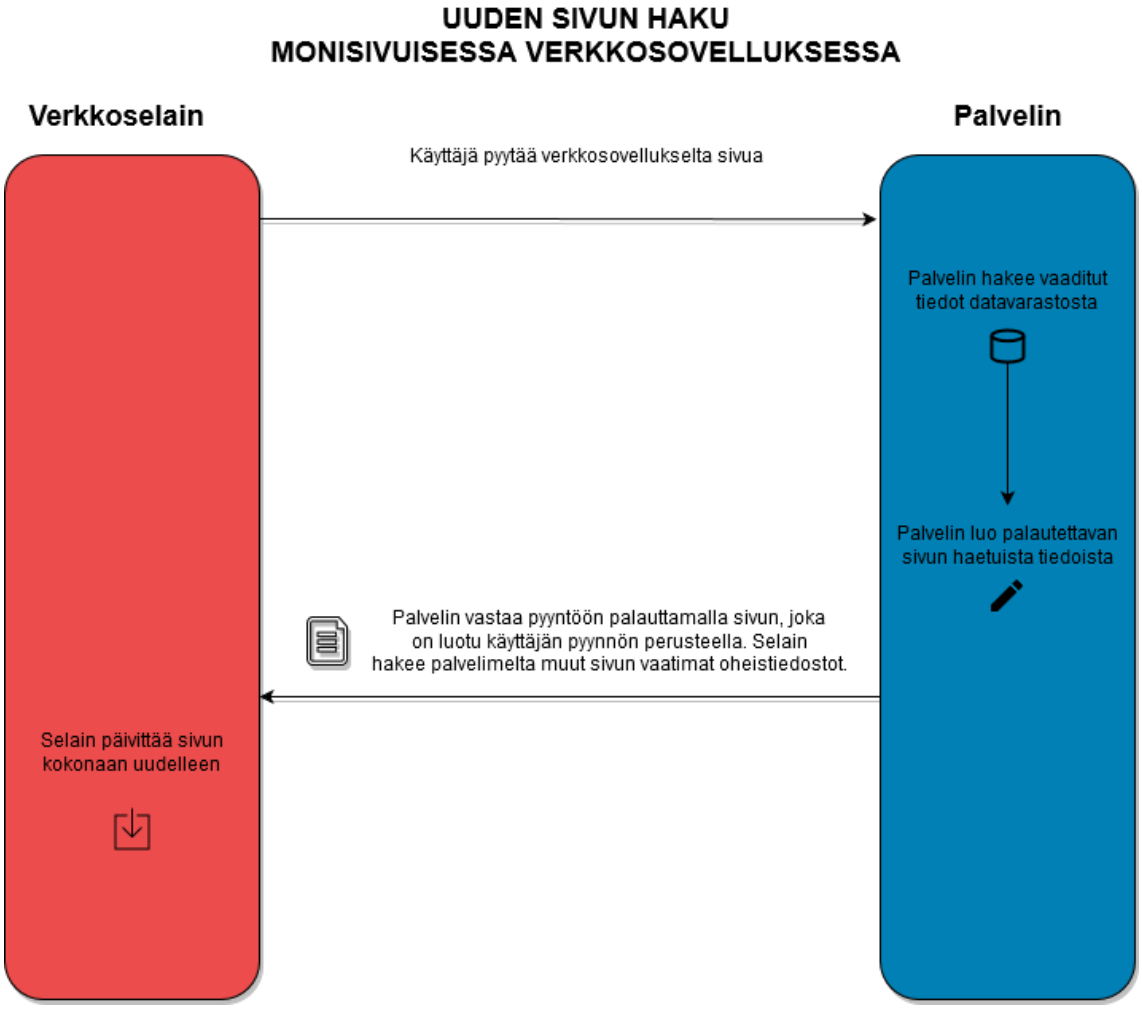
Verkkosivutyyppien lisäksi (staattiset ja dynaamiset) on olemassa myös erilaisia verkkosovellustyyppejä. Jason Bengtson (2019) kertoo kahdesta sovellustyypeistä, joita on kahdenlaisia: monisivuisia (MPA – multi page application, perinteiset verkkosovellus) ja yksisivuisia (SPA – single page application). Verkkosovellusten nimistä voidaan päätellä myös niiden ero. Yksisivuiset verkkosivut on suunniteltu toimimaan vain yhdellä sivulla, jossa sivun sisältöä päivitetään selaimessa. Monisivuisten verkkosovelluksien toiminnallisuus ja sisältö puolestaan jakautuu monelle eri sivulle yhden sivun sijaan. (Bengtson 2019, 28.)

2.1.2 Monisivuiset verkkosovellukset

Scott Emmit (2015) kertoo monisivuisten verkkosovellusten toimintaperiaatteesta. Käyttäjän pyytäessä uutta sivua hän lähettää palvelimelle pyynnön sivusta. Palvelin käsittelee pyynnön ja hakee vaadittavan tiedot pyynnön täyttämiseen, luo palautettavan sivun mallinnusjärjestelmällä ja palauttaa sivun käyttäjälle (tai vaihtoehtoisesti palauttaa staattisen HTML-sivun, joka on kaikille samanlainen). Verkkoselaimen saadessa uuden sivun se piirtää sen vanhan tilalle. Jokainen käyttäjän pyyntö uudesta sivusta vaatii pyynnön kulkemisen käyttäjän verkkoselaimesta palvelimelle ja takaisin sekä uuden sivun piirtämisen selaimeen. (Emmit 2015.) Monisivuisten verkkosovellusten piirtologiikka on siis palvelimella, jossa palvelin päättää käyttäjän pyynnön perusteella mikä sivu käyttäjälle lähetetään ja mitä sivulla näytetään. Tällöin monisivuiset verkkosovellukset toimivat pääosin palvelimella ja suurin osa niiden logiikasta on myös siellä.

Kuviossa 1 kuvataan tarkemmin, miten monisivuisessa verkkosovelluksessa käyttäjä saa uuden sisällön selaimeensa. Käyttäjän selaimen pyytäessä uutta sisältöä palvelimelta selain hakee palvelimelta uuden sivun. Palvelin käsittelee

pyynnön, hakee vaaditut tiedot ja rakentaa haetuista tiedoista palautettavan sivun. Selain päivittää sivun kokonaan uuteen, joka saatiin vastauksena pyyntöön palvelimelta.



KUVIO 1. Yksinkertaistettu kuvio siitä, miten monisivuisessa verkkosovelluksessa uusi sisältö haetaan käyttäjälle. Kuvio on luotu perustuen Emmit (2015) kirjan kuvaukseen.

Haettavan sivun lisäksi käyttäjän selain hakee yleensä myös CSS-tyylitiedostoja ja JavaScript-ohjelmatiedostoja. CSS-tyylitiedostojen avulla pystytään muuttamaan sivun sisällön ulkonäköä. CSS pystyy määrittelemään, mikä esimerkiksi on tekstin väri tai koko, elementtien välissä olevan tyhjän tilan määrä jne (MDN web docs 2020 CSS basics). JavaScript-ohjelmatiedostojen avulla puolestaan luodaan sivulle toiminnallisuus, kuten sivujen sisällön muuttaminen dynaamisesti tai palvelimelle pyyntöjen lähettäminen (MDN web docs 2020 JavaScript Introduction). CSS- ja JavaScript-tiedostot ovat tärkeä osa verkkosovelluksia, sillä ne

mahdollistavat verkkosivujen ulkonäön ja toiminnallisuuden. Käyttäjän verkkoselain osaa hakea sivun vaatimat CSS-, JavaScript- ja muut tiedostot automaattisesti sivussa ilmoitettujen tietojen perusteella.

Monisivuisissa verkkosovelluksissa ilmenee ongelma, jossa uuden sisällön saaminen selaimeen ei ole niin nopeaa kuin voisi olla. Kun käyttäjä navigoi uudelle sivulle, palvelimen täytyy lähettää käyttäjälle sivu ja sen oheistiedostot. Selaimen täytyy tällöin päivittää ja ladata koko sivu ja sen tiedostot uudelleen, kun käyttäjä pyytää uuden sivun palvelimelta.

2.1.3 Yksisivuiset verkkosovellukset

Yksisivuiset verkkosovellukset tarjoavat ratkaisun monisivuisten verkkosovelluksien ongelmakohtiin. Michael Mikowksin ja Josh Powellin mukaan yksisivuisten verkkosovellukset tarjoavat työpöytäsovelluksien kaltaisen välittömyyden käyttäjän tekemiin toimintoihin. Ne poistavat käyttäjältä odotusajan uusien sivujen ja tiedostojen hakemiselta ja pystyvät vastaamaan käyttäjän toimintoihin käytännössä heti. (Mikowski & Powell 2013.)

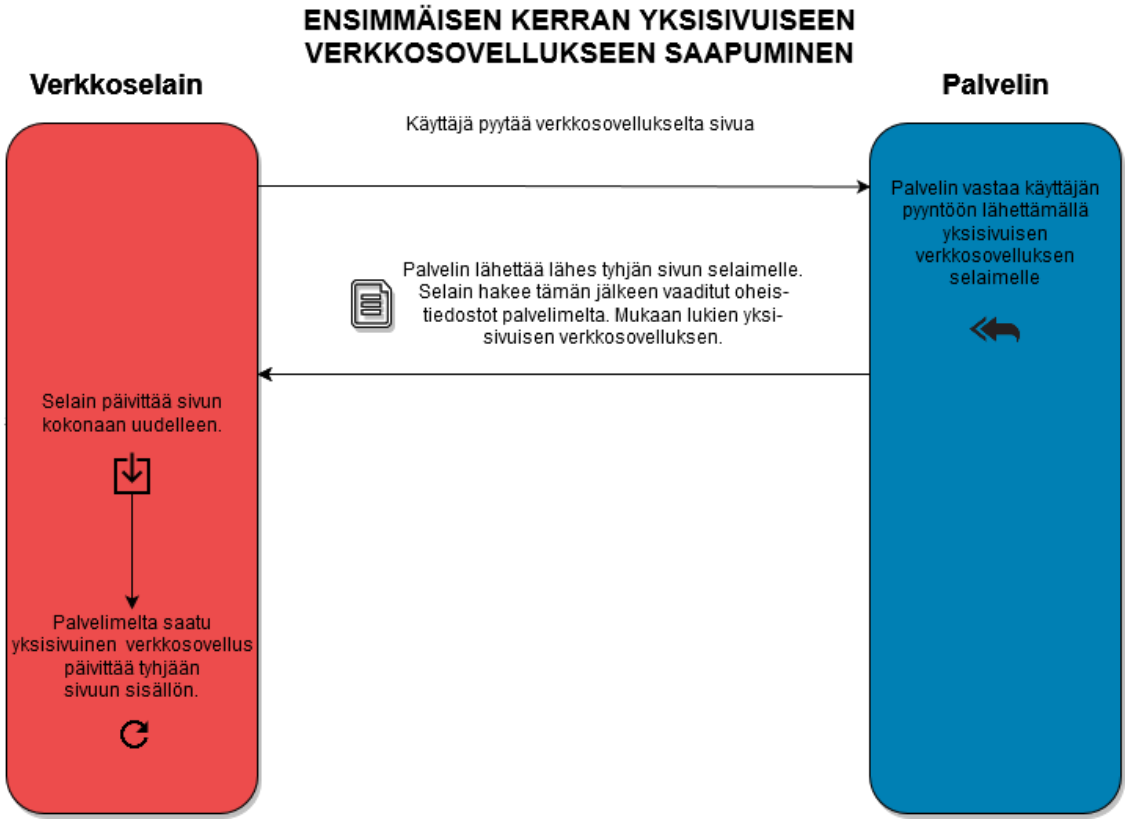
Yksisivuisten verkkosovellusten suurimpana erona monisivuiseen on se, missä käyttäjälle uuden sisällön piirtävä logiikka on. Yksisivuisissa verkkosovelluksissa piirtologiikka on pääosin käyttäjän selaimella toimivassa yksisivuisessa verkkosovelluksessa (Emmit 2015). Tällöin sivun sisällön rakentaminen ja mitä sivulla näytetään on siirretty palvelimelta käyttäjän selaimeen pääasiallisesti.

Eli erona on siis, että kun monisivuisessa verkkosovelluksessa palvelin päättää mikä sivu selaimelle lähetetään, yksisivuisessa verkkosovelluksessa käyttäjälle näytettävän näkymän päättää selaimessa toimiva yksisivuinen verkkosovellus. Emmit (2015) kertoo, että ensimmäisen sivulatauksen latauksen jälkeen sivun ei tarvitse hakea palvelimelta enää uusia tiedostoja, vaan uuden sisällön piirtäminen ja päivittäminen tapahtuu selaimessa JavaScriptin avulla. Uuden sisällön näyttäminen käyttäjälle tapahtuu hakemalla palvelimelta uuden näkymän vaatimat tiedot ja piirtämällä se käyttäjän selaimessa toimivan verkkosovelluksen avulla. (Emmit 2015). Monisivuisessa verkkosovelluksessa sivut haetaan palvelimelta aina uudelleen, kun käyttäjä haluaa uuden sivun. Tällöin käyttäjä joutuu

odottamaan, että pyyntö uudesta sivusta otetaan vastaan, piirretään, lähetetään takaisin ja että selain päivittää sivun kokonaan uuteen. Tähän verrattuna yksisivuisessa sovelluksessa selain pyytää vain uuden näkymän muodostamiseen vaaditut tiedot palvelimelta ja piirtää uuden näkymän itse.

Emmit (2015) kuvaa, miten yksisivuisessa verkkosovelluksen logiikka toimii. Ensimmäisen kerran yksisivuiseen sovellukseen saavuttaessa palvelin lähettää monisivuisen sovelluksen tapaan sivun selaimeen. Tavallisesta monisivuisen verkkosovelluksen sivuista poiketen lähetetty sivu on käytännössä yleensä tyhjä. Sivulla saattaa olla yksi tyhjä elementti, joka on merkitty jollakin tietyllä id-arvolla. Tähän tyhjään elementtiin yksisivuinen verkkosovellus päivittää käyttäjälle verkkosovelluksen sisällön. Ohjelma käyttää selaimen DOM-rajapintaa uusien näkymien näyttämiseen käyttäjälle selaimessa. Tällöin selaimen ei tarvitse päivittää sivua kokonaan uudelleen, koska ohjelma pystyy päivittämään tarvittavat ominaisuudet dynaamisesti. (Emmit 2015.)

Kuviossa 2 kuvataan, miten yksisivuinen verkkosovellus käyttäytyy, kun käyttäjä saapuu ensimmäisen kerran sivustolle. Selain pyytää palvelimelta sivua, johon palvelin vastaa lähes tyhjällä sivulla. Selain hakee myös saadun sivun oheistiedostot, jossa on mukana yksisivuinen verkkosovellus. Verkkosovelluksen latauduttua se päivittää sisällön näkyviin käyttäjälle.



KUVIO 2. Yksinkertaistettu kuvio, miten yksisivuinen verkkosovellus ladataan ensimmäisen kerran käyttäjän selaimeen. Kuviossa on luotu perustuen Emmit (2015) kirjan kuvaukseen.

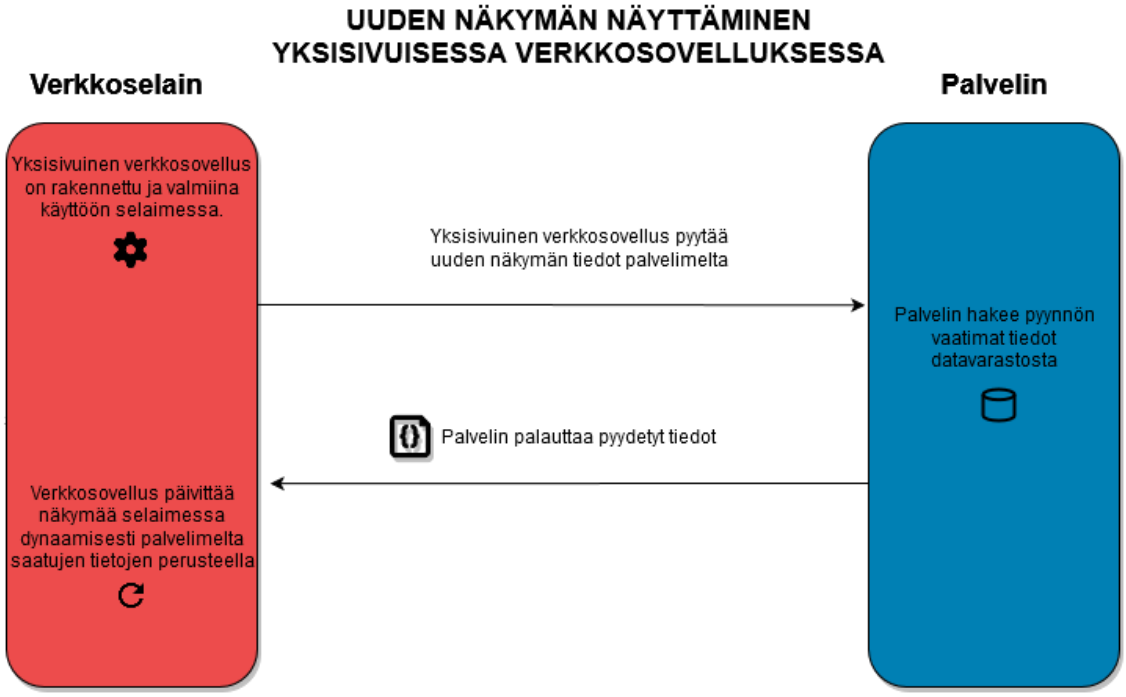
Yksisivuisissa verkkosovelluksissa olevaa sovelluslogiikkaa ja datan käsittelyä on siirretty mahdollisimman paljon palvelimelta käyttäjän selaimessa toimivaan verkkosovellukseen (Emmit 2015). Vaikka sovelluksen logiikkaa ja datan käsittelyä on siirretty palvelimelta selaimelle, tarvitsee selain kuitenkin palvelinta eri tarkoituksiin, vaikka se ei käytä palvelinta enää uusien sivujen hakemiseen tai muodostamiseen ensimmäisen latauksen jälkeen.

Emmit (2015) selittää, miten yksisivuiset verkkosovellukset käyttävät palvelinta. Selaimessa toimiva sovellus käyttää palvelinta tietojen hakemiseen ja tallentamiseen pysyväistalletuksesta. Tietoja voidaan hakea palvelimelta pysyväistalletuksesta joko hakemalla valmiiksi piirrettyjä käyttöliittymän komponentteja palvelimelta tai hakemalla uuden näkymän vaatimaa dataa. Haetut käyttöliittymäkomponentit voidaan piirtää dynaamisesti selaimeen ilman sivun päivitystä tai haetun datan avulla sovellus voi piirtää sivulle uuden näkymän. Tällöin käyttäjän ei tarvitse odottaa selaimen päivittymistä kokonaan uuteen sivuun. Tämän lisäksi verk-

koselaimessa toimiva sovellus lähettää pyyntöjä tallettaa tai poistaa tietoja pysyväistalletuksesta. (Emmit 2015.) Tietojen pysyväistalletuksen lisäksi palvelinta voidaan käyttää käyttäjän autentikointiin ja yksisivuisen verkkosovelluksen lähettämien tietojen validointiin (Mikowski & Powell, 2013).

Kummassakin tapauksessa palvelimen kanssa vaihdetun tiedon määrä vähenee, koska palvelimen ei tarvitse lähettää kokonaan uuden sivun tietoja, vaan se pystyy välittämään tarvitut tiedot joko osina tai pelkästään datana, jonka selaimessa toimiva sovellus päivittää sivuun. Kuviossa 3 selitetään, miten yksisivuiset verkkosovellukset näyttävät uuden näkymän käyttäjälle.

Kuviossa 3 kuvataan, miten selaimessa valmiina oleva yksisivuinen verkkosovellus näyttää uuden sisällön käyttäjälle. Verkkosovelluksen ollessa valmiina käyttäjän selaimessa sillä on kaikki työkalut uusien näkymien luomiseen ja näyttämiseen. Sovellus lähettää pyynnön palvelimelle tiedoista, joita se tarvitsee uuden näkymän näyttämiseen. Palvelin hakee datavarastosta vaaditut tiedot, jotka se palauttaa selaimelle. Selaimessa toimiva verkkosovellus osaa päivittää dynaamisesti näkymän käyttäjän sivulle ilman kokonaan uuden sivun päivittämistä.



KUVIO 3. Yksinkertaistettu kuvio, miten käyttäjän selaimessa toimiva yksisivuinen verkkosovellus hakee uuden näkymän näyttämiseen vaaditut tiedot palvelimelta. Kuvio on luotu perustuen Emmit (2015) kirjan kuvaukseen.

Verkkosovellusten ei kuitenkaan ole pakko olla joko yksisivuisia tai monisivuisia. Sovelluksen tyyliksi voidaan myös valita hybridi, jossa yhdistellään molempia tyylejä. Hybridiverkkosovelluksissa osa sivuista on luotu monisivuisella ja osa yksisivuisella tekniikalla tai yhdistelemällä näitä tekniikoita samalla sivulla. (Bengtson 2019, 29.) Yksi vaihtoehdoista on upottaa monimutkaisemmat ominaisuudet yksisivuisina verkkosovelluksina osaksi monisivuista verkkosivurakennetta. Tällöin yksinkertaiset verkkosivukokonaisuudet voidaan toteuttaa helposti mallinnusjärjestelmällä tai staattisilla sivuilla ja monimutkaiset verkkosovellukset voidaan toteuttaa omina alakokonaisuuksinaan. Näin pystytään yhdistelemään molempien sovelluksien ominaisuuksia.

Monisivuiset verkkosovellukset pystyvät myös muuttamaan sisältöään käyttäjän selaimessa dynaamisesti tarvittaessa verkkoselaimen tarjoaman DOM-rajapinnan (Document Object Model) kautta JavaScriptillä. DOM:in kautta pystytään vaihtamaan HTML-sivun elementtien ulkonäköä, toiminnallisuutta tai muuttamaan elementti kokonaan toiseen (MDN web docs 2020 Introduction to the DOM). Tätä varten on tehty kirjastoja, jotka helpottavat työtä. Yksi esimerkki on jQuery, joka mainostaa itseään HTML dokumentin käsittelemistä helpottava kirjastona (jQuery n.d). DOM-rajapinnan ja kirjastojen avulla sivulle pystytään lisäämään toiminnallisuutta, kuten ponnahdusikkunoiden näyttämistä, datan esittämistä ja niin edelleen. Vaikka monisivuisen verkkosovelluksen HTML-sivun elementtejä muutetaan lennosta käyttäjän sivulle, sen pääsääntöinen uuden sisällön näyttäminen tapahtuu hakemalla uusi sivu palvelimelta.

2.2 Mitä etuja yksisivuinen verkkosovellus tuo?

Yksisivuiset verkkosovellukset peittoavat monisivuiset käyttäjäkokemuksessa nopeudellaan, koska se näyttää uuden sisällön nopeammin käyttäjälleen (Mikowski & Powell 2013). Kuten edellisessä kappaleessa mainittiin, sisällön piirtäminen tapahtuu käyttäjän verkkoselaimessa palvelimen sijaan, joten uusi sisältö pystytään näyttämään käyttäjälle nopeasti. Monisivuiset verkkosovellukset hakevat uuden sisällön palvelimelta ja verkkoselain joutuu päivittämään koko sivun

tätä varten. Tähän voi kulua useita sekunteja huonoissa olosuhteissa, kun yksisivuiset pystyvät näyttämään uuden sisällön melkein heti (Mikowski & Powell 2013).

Käyttäjän tekemiin toimintoihin vastaaminen tapahtuu myös nopeammin yksisivuisissa verkkosovelluksissa. Osa verkkosovelluksen logiikasta ja datan käsittelystä on siirretty palvelimelta käyttäjän selaimeen, jolloin verkkoselain pystyy vastaamaan käyttäjän toimintoihin nopeasti. (Mikowski & Powell 2013.) Monisivuisessa verkkosovelluksessa käyttäjän täytyy odottaa pyynnön lähettämistä, siihen vastaamista ja uuden sivun piirtämistä. Yksisivuiset verkkosovellukset voivat vain paikallisesti päättää käyttäjien toimien mukaisesti mitä piirretään. Yksisivuisten verkkosovellusten täytyy kuitenkin odottaa näkymän piirtämiseen tarvitut tiedot palvelimelta.

Palvelimen kanssa vaihdetun tiedon määrä on pienempi, mikä vähentää odotusaikaa. Ensimmäisen latauksen jälkeen yksisivuinen verkkosovellus ei lataa uusia sivuja palvelimelta. Yksisivuinen verkkosovellus tarvitsee vain tiedot uuden näkymän muodostamiseen palvelimelta. Tällöin verkkoselaimen ei tarvitse päivittää kokonaan uutta sivua, vaan yksisivuinen verkkosovellus pystyy päivittämään nopeasti näkymäänsä palvelimelta saatujen tietojen mukaisesti. (Emmit 2015.)

Yksisivuisessa verkkosovelluksessa on myös etuja verrattuna tietokoneella tai mobiililaitteella toimivaan sovellukseen.

Verkkoselaimet kuuluvat maailman käytetyimpien ohjelmistojen joukkoon. Tämä mahdollistaa sen, että verkkosovellusta pystyy käyttämään mahdollisimman moni helposti. Sovelluksen käyttämiseen vaaditaan vain nykyaikainen verkkoselain ja verkkoyhteys, jotka löytyvät tietokoneista puhelimiin ja tabletteihin. Koska verkkosovellukset toimivat selaimissa ei tällöin sovellusta tarvitse asentaa kuten normaaleja työpöytäsovelluksia. (Mikowski & Powell 2013.) Tämä pätee myös mobiilisovelluksiin, jotka tulee asentaa sovelluskaupasta. Tällöin käyttäjille voidaan tarjota sovellus, jota käyttäjät pystyvät käyttämään navigoimalla vain verkkoselaimella sovelluksen osoitteeseen. Tällöin käyttäjän ei tarvitse ladata edes haluttua sovellusta, vaan sovellukseen pääsee käsiksi heti.

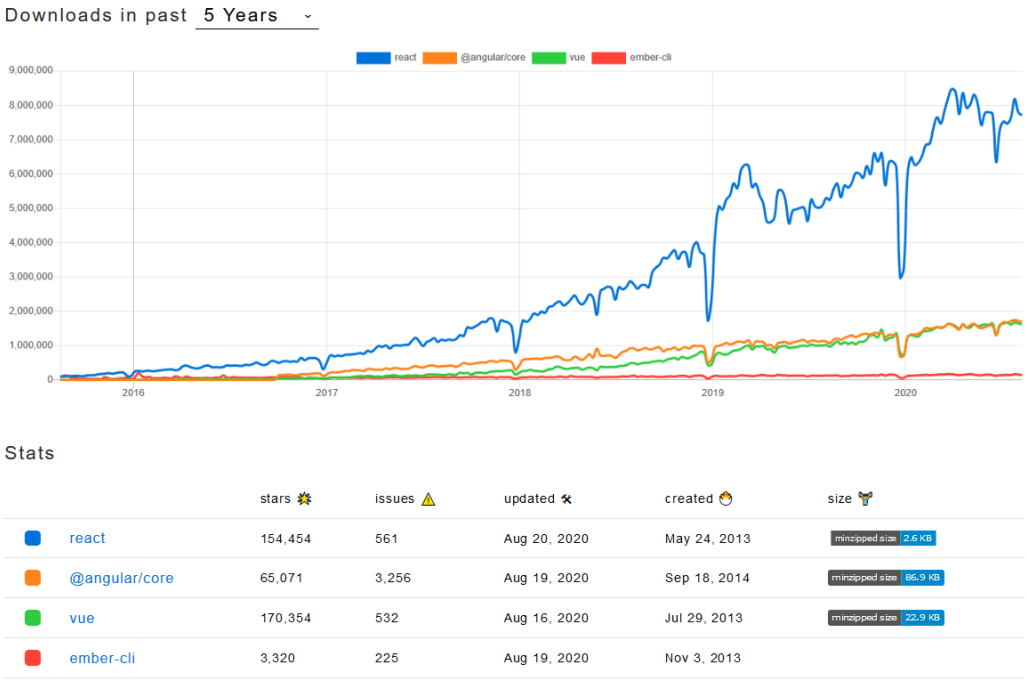
Tämän lisäksi verkkosovelluksia voidaan päivittää helposti verrattuna työpöytäsovelluksiin (Mikowski & Powell 2013). Käyttäjän verkkoselain hakee palvelimelta yksisivuisen verkkosovelluksen aina uudestaan, kun hän navigoi verkkosovellukseen. Kun palvelimelle päivitetään uusin versio yksisivuisesta verkkosovelluksesta, käyttäjät saavat sen heti käyttöönsä. Työpöytäsovelluksissa yleensä uusi versio täytyy aina asentaa tai asennuttaa käyttäjille erikseen (Mikowski & Powell 2013). Sama pätee myös mobiilisovelluksiin, joissa päivitykset tehdään sovelluskaupasta.

2.3 Miten yksisivuinen verkkosovellus toteutetaan?

Yksisivuisten verkkosovelluksien toteutuksessa käytetään yleensä jotakin kirjastoa, joka auttaa niin oikean näkymän päättämisessä kuin sen piirtämisessäkin (Emmit 2015). Kirjastoja yksisivuisten verkkosovellusten tekemiseen on useita erilaisia. Rafael Carvalho (2017) mainitsee artikkelissaan suosituimmiksi sovelluskirjastoiksi seuraavat:

- Angular
- React.js
- Vue.js
- Meteor
- Ember
- Aurelia

Vertailemalla NPM-paketinhallintajärjestelmän asennuslukuja suosituimmista yksisivuisten verkkosovellusten kirjastoista saamme kuvan niiden suosiosta (kuvio 4).



KUVIO 4. Kuvio näyttää eri asennuksien määrän NPM-paketinhallintajärjestelmässä (npm trends 2020). Vertailuun on otettu mukaan kirjastoihin kuuluvista paketeista eniten asennettu.

Kolme kirjastoa erottuu selvästi joukosta: React, Angular ja Vue. React- ja Angluar-kirjastojen takana ovat suuret teknologiayritykset: React-kirjastolla Facebook (React 2020) ja Angular-kirjastolla Google (Angular 2020). Vue-kirjaston alkuperäinen kehittäjä on entinen Googlen työntekijä Evan You (You 2020).

Yksisivuisten verkkosovellusten kehittämiseen löytyy monta erilaista kirjastoa, joiden toimintaperiaatteet eroavat toisistaan. Osalla kirjastoista on takanaan suuria yrityksiä, jotka todennäköisesti vaikuttavat niiden suosioon. Mahdollisten kirjastojen määrä helpottaa kehittäjän työtä, koska se tarjoaa mahdollisuuden valita mieleisensä kirjasto kehitykseen.

2.4 Piceasoft asiakkuudenhallintajärjestelmä

Opinnäytetyö on toteutettu olemassa olevaan Piceasoft Oy:n asiakkuudenhallintajärjestelmään, joka on hybridiverkkosovellus. Toteutettava verkkosovellus on osana kokonaisuutta, jossa osa sivuista on tehty monisivuisina ja osa yksisivuisina verkkosovelluksina. Tällöin yksinkertaisimmat sivut on pystytty toteuttamaan EJS-mallinnusjärjestelmän avulla ja monimutkaisemmat alikokonaisuudet (kuten

toteutettu opinnäytetyö) on tehty yksisivuisina verkkosovelluksina käyttäen React-kirjastoa.

Yksisivuiset verkkosovellukset toimivat alikokonaisuutena suuremmassa järjestelmässä. Asiakkuudenhallintajärjestelmän palvelimessa toimivan sovelluksen ympäristönä toimii Node.js ja palvelinkirjastona Express.

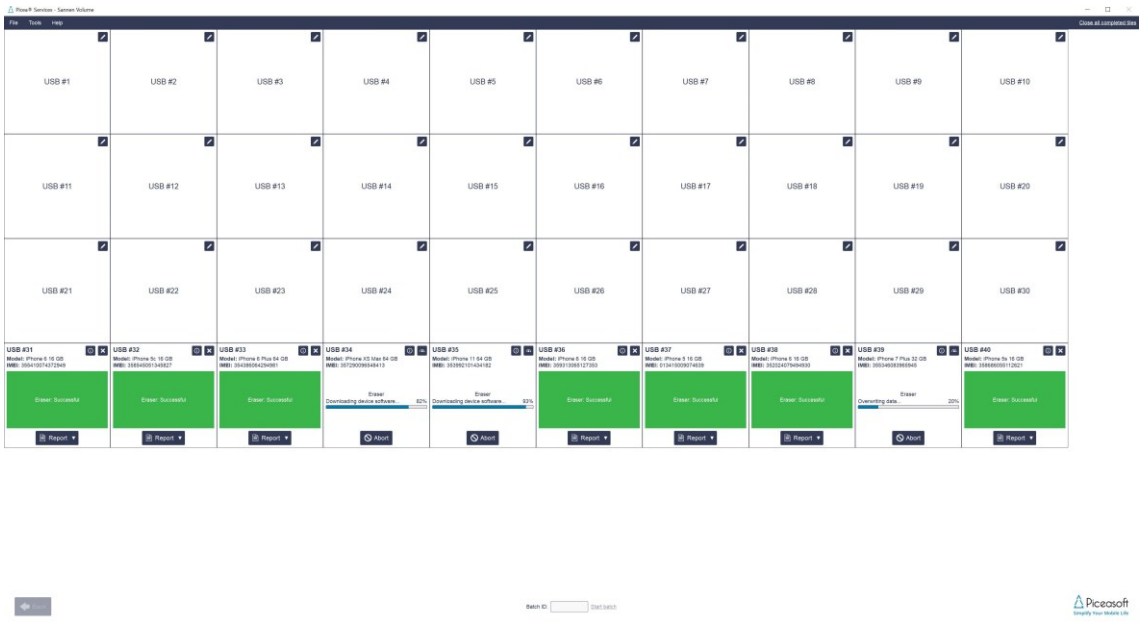
3 TYÖNKULUN HALLINTAJÄRJESTELMÄ

3.1 Tarve

Opinnäytetyö tehtiin Piceasoft Oy:lle. Toimialanaan yritys tuottaa ohjelmistoja puhelimien elinkaarten hallintaan. Tuotekatalogiin kuuluu PiceaServices, joka tarjoaa palveluita puhelimien sisällön vaihdosta puhelimesta toiseen, puhelimen myyntivalmiuden varmistamiseen, puhelimen diagnostiikkaan, puhelimien muistin pyyhkimiseen ja puhelimien myyntihinnan arvioimiseen. PiceaServices toimii käyttäjien tietokoneella, jossa he voivat yhdistää puhelimia tietokoneeseen ajaakseen palvelua. Yksi palveluista on Volume, joka sitoo kolme näistä operaatioista yhteen:

- Verify selvittää onko tuote myyntivalmis. Automatisoitujen testien avulla ohjelma pystyy tarkistamaan esimerkiksi, ettei käyttäjällä ole jäänyt puhelimeen aktiivisia tilejä, SIM-kortteja tai ettei puhelinta ole ilmoitettu varastetuksi
- Diagnosticsin avulla puhelimelle pystytään teettämään laaja-alaisesti diagnostiikkatestejä. Diagnostics suorittaa käyttäjän avulla testejä esimerkiksi puhelimen kameran kunnon varmistamiseksi, näytön kuolleiden pikseleiden havaitsemiseksi ja puhelimien nappien toiminnallisuuden varmistamiseksi
- Eraserin avulla puhelimen muisti pystytään pyyhkimään luotettavasti, jotta puhelin voidaan kierrättää uudelleen tietoturvallisesti.

Näitä operaatioita voidaan ajaa puhelimella erikseen muista operaatioista. Volume sitoo nämä kaikki operaatiot yhden operaation alle, jossa voidaan käsitellä kymmeniä puhelimia samanaikaisesti. Volume on suunnattu kierrättäjien kaltaisille toimijoille, joilla on tarve käsitellä mahdollisimman monta puhelinta nopeasti. Kuvassa 1 on nähtävillä Volume-operaation suoritus PiceaServicesissä.

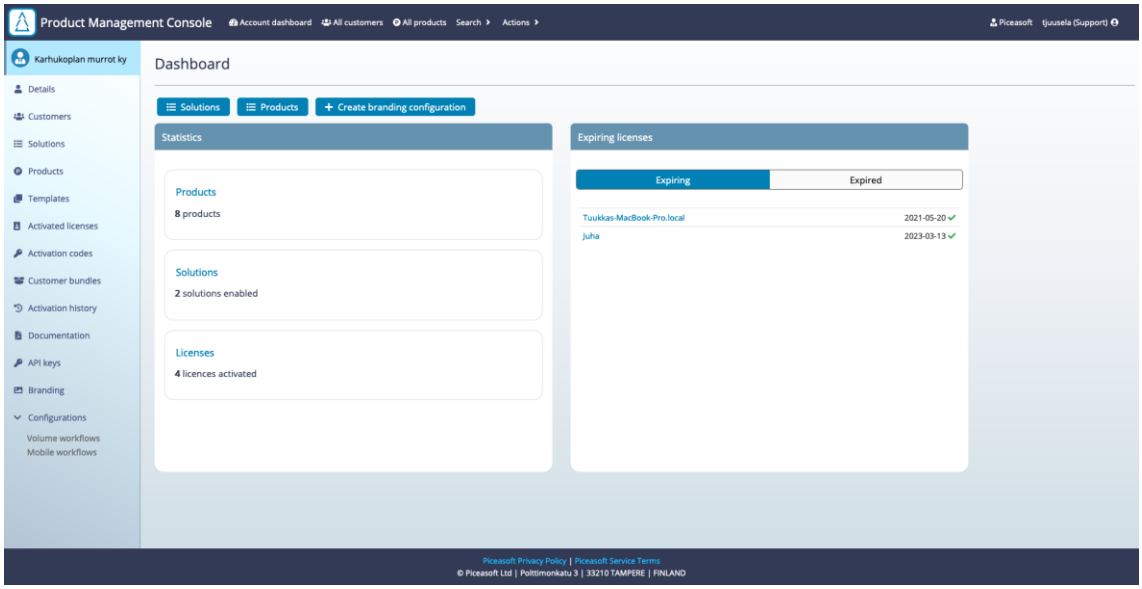


KUVA 1. Kuva Volume-operaation suorituksesta PiceaServicesissä. Kuvan ottamishetkellä laitteille suoritetaan Erasure-operaatiota.

Piceasoft Oy:n tuotteiden ympärille on myös syntynyt tuoteperhe tukemaan muiden tuotteiden toimintaa. Esimerkkeinä näistä on raportointijärjestelmä ja asiakkuudenhallintaohjelma. Asiakkuudenhallintaohjelmalla nimensä mukaisesti hallitaan asiakkaiden käyttäjiä ja tuotteiden ominaisuuksia. Asiakkuudenhallintaohjelman virallinen nimi on Product Management Console eli tuttavallisesti PMC (PMC:n asiakkuuden kojelauta näkyvissä kuvassa 2). PMC syntyi tarpeesta luoda kattava kokonaisuus asiakkaiden hallintaa varten vanhan asiakkuudenhallintajärjestelmän tilalle moderneilla tekniikoilla. Virallisesti uusi asiakkuudenhallintajärjestelmä julkaistiin kesällä 2019, jonka jälkeen PMC:hen on tuotu lisää ominaisuuksia.

Yksi julkaisun jälkeen lisätyistä ominaisuuksista on asiakkaan Volume-tuotteen työnkulun määrittely asiakkuudenhallintajärjestelmässä kaikille asiakkaan käytössä oleville tietokoneille. Ennen käyttäjät joutuivat paikallisesti määrittelemään jokaiselle tietokoneelle oman Volumen määrityksensä. Tätä ongelmaa varten alettiin kehittämään ratkaisua, jolla Volumen määrityksen saisi kaikille asiakkaan tai tuotteen käyttämille järjestelmille. Ensimmäinen ratkaisu oli Configuration API, jolla asiakkaat pystyvät luomaan oman rajapinnan Volumen ominaisuuksien määrittelyyn. Rajapinnan tuli palauttaa JSON-tiedosto, jonka Volume haki käyttäjän sovelluksen käynnistyessä. Tämä vaati asiakkaalta teknistä osaamista ja

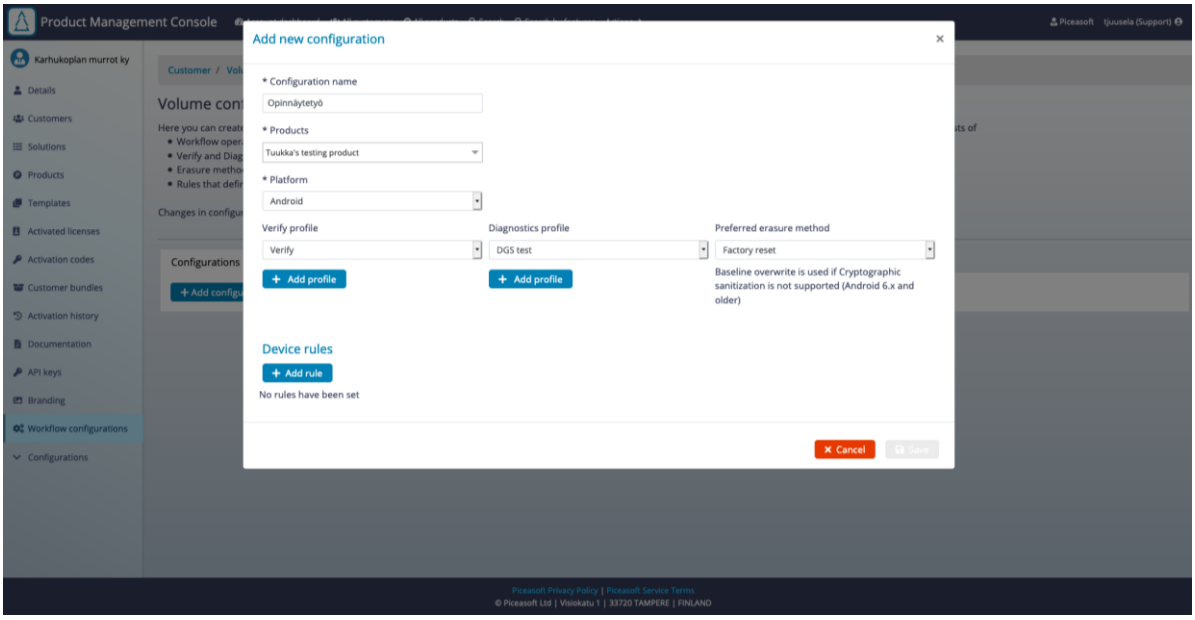
infrastruktuuria, minkä takia päätettiin luoda asiakkuudenhallintajärjestelmään graafinen käyttöliittymä helpottamaan tehtävää.



KUVA 2. Kuvakaappaus asiakkuudenhallintajärjestelmä PMC:n asiakkuuden kojelaudasta. Vasemmassa alakulmassa näkyy Volumen työnkulun hallintajärjestelmän pääsyyn käytetty navigaatioelementti ”Volume workflows”.

Ratkaisuna tähän syntyi asiakkuudenhallintajärjestelmään edellinen Volumen työnkuluhallintajärjestelmä, jossa asiakkaat pystyivät hallitsemaan Volumen työnkulkua. Virallisesti asiakkuudenhallintajärjestelmässä Volumen työnkulunhallintajärjestelmää kutsutaan Volume workflow’ksi. Järjestelmä mahdollisti määrittelyjen luomisen alijärjestelmäkohtaisesti: Erasuren muistinpyyhintämetodi, Verifyn ja Diagnostiikan testit. Volumen alijärjestelmäkonfiguraatiot pystyttiin nitomaan yhteen luomalla uusi Volumen työnkulku eli ”workflow”. Uuden työnkulun luontiin käytetty ponnahdusikkuna kuvassa 3.

Järjestelmästä pystyttiin myös hallitsemaan Volumen työnkulun laitesuodattimia, jotka määräävät millä puhelimille operaatiot voidaan ajaa. Työnkulusta oli valittava vähintään puhelinalusta, jolla työnkulku pystyttäisiin ajamaan. Tämän jälkeen laitesuodattimista voitiin määrittää myös työnkulku ajettavaksi vain tietyn valmistajan laitteilla tai tietyillä valmistajan puhelimilla. Alustan perusteella valittiin myös työnkulussa käytetty Erasuren muistinpyyhintämetodi. Työnkulun alustavalinnan vuoksi työnkulkua pystyttiin käyttämään vain yhdellä alustalla kerrallaan, mikä rajoitti työnkulun ketteryyttä.



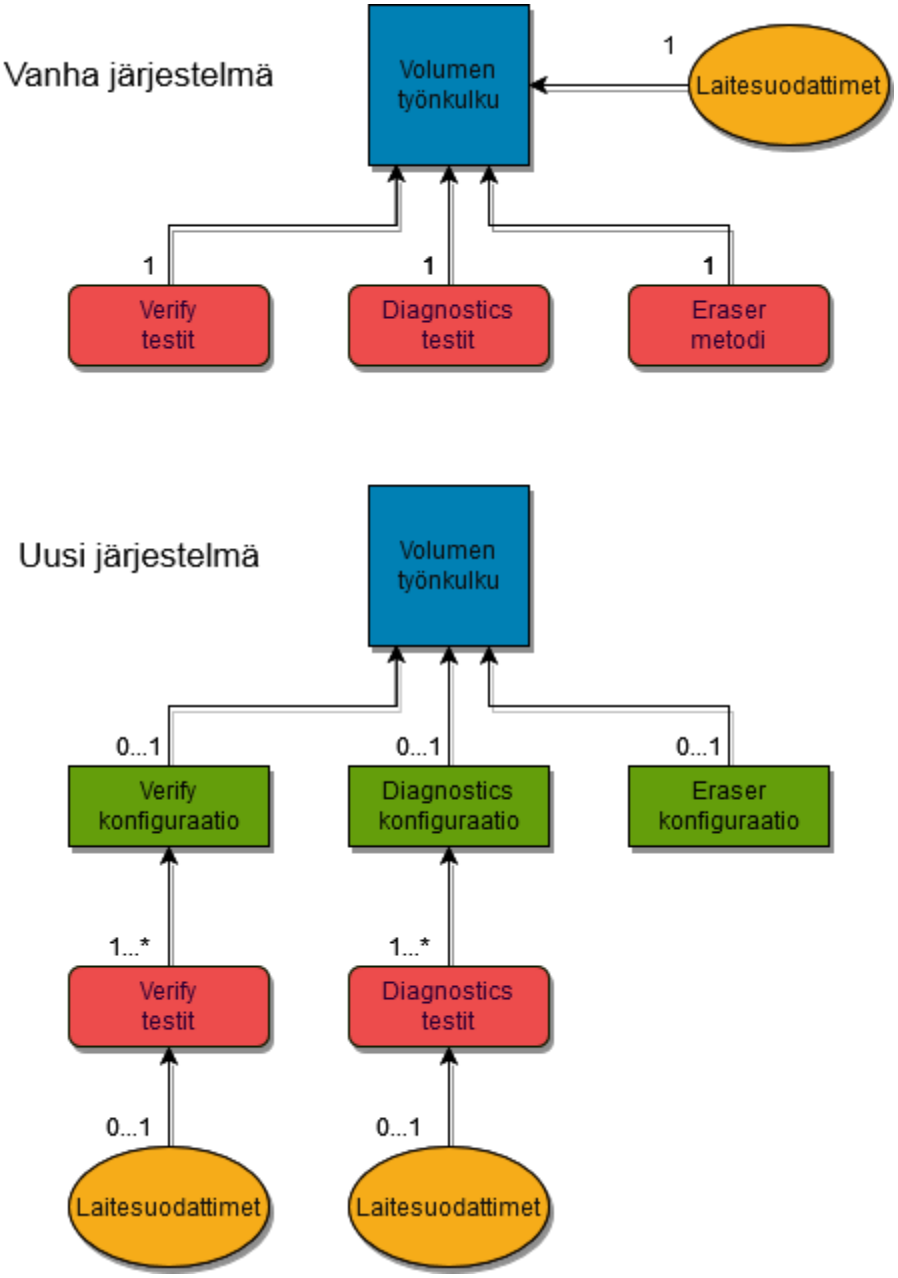
KUVA 3. Kuvakaappaus vanhasta Volumen työnkulun hallintajärjestelmästä, jossa on auki uuden työnkulun luontiin käytetty ponnahdusikkuna.

Asiakkailla, jotka käyttivät tätä ominaisuutta, oli kuitenkin lisätoiveita työnkulun luontiin liittyen. Yksi tärkeimmistä lisätoiveista oli mahdollisuus luoda monta ”sääntöä” Verifyn ja Diagnosticsin konfiguraatioihin, jotka liitetään yhteen Volumen työnkulkuun. Säännöt olivat edellisen järjestelmän Verifyn ja Diagnosticsin alijärjestelmäkonfiguraatioita, joissa pystytään valitsemaan, mitkä testit puhelimelle ajetaan. Sääntöihin pystytään myös luomaan sääntökohtaisia laitesuodattimia usealle alustalle kerrallaan. Tällöin esimerkiksi yksi Verify-konfiguraatio kykenee samaan aikaan pitämään sisällään iOS- ja Android-laitteiden erilliset testit samassa konfiguraatiossa. Erasure-konfiguraatioon tuli lisätä mahdollisuus määrittää muistinpyyhintämetodit kaikkiin ajettavissa oleviin alustoihin samanaikaisesti.

Erona vanhan ja suunnitellun järjestelmän välillä oli siis:

- Verify- ja Diagnostics-konfiguraatiot sisältävät sääntöjä, joissa on määritelty, mitkä testit ajetaan. Sääntöihin pystytään määrittämään laitesuodattimet alusta-, valmistaja- ja laitekohtaisesti. Myös useamman alustan määrittäminen yhteen sääntöön on mahdollista
- Erasure-konfiguraatioon pystytään määrittelemään muistin pyyhintämetodit samanaikaisesti eri alustoille.

Järjestelmien erot näkee myös kuviossa 5, johon on kuvattu molempien järjestelmien rakenne.



KUVIO 5. Vanhan ja uuden Volumen työnkulun hallintajärjestelmän dataraken-teet.

Uudet ominaisuusvaatimukset edellyttivät vanhalta Volume workflows-järjestelmältä suuren määrän työtä tietokannaltaan, tietokannan rajapinnalta sekä selainkäyttöliittymältään. Tämän vuoksi tuli tehdä päätös, ryhdytäänkö vanhaa järjestelmää muokkaamaan vai tehdäänkö Volumen työnkulku uudelleen lisäten tuki muiden tuotteiden konfiguraatioiden luomiseen ja tehden järjestelmä yksisivuisena verkkosovelluksena monisivuisen sijaan.

Järjestelmä päädyttiin tekemään yksisivuisena sovelluksena. Yksisivuisena verkkosovelluksena toteuttamiseen päädyttiin aikaisempien positiivisten kokemuksien ansiosta ja yksisivuisten verkkosovelluksien tuomien hyötyjen takia. Päätökseen vaikutti myös opinnäytetyöntekijän aikaisempi kokemus yksisivuisten verkkosovellusten toteutuksesta ja Piceasoft Oy:n suunnitelma tehdä samaan yksisivuiseen verkkosovelluspohjaan muita saman tyyppisiä järjestelmiä. Projekti kuului opinnäytetyöntekijän tavanomaisiin työtehtäviin, mutta siitä päätettiin tehdä samalla hänen opinnäytetyönsä.

Varsinaisen työnkulunhallintajärjestelmän toteutuksen lisäksi opinnäytetyö pohjusti useita tulevia projekteja. Opinnäytetyöntekijä suunnitteli ja toteutti yksisivuisen verkkosovelluksen pohjan. Opinnäytetyön Diagnosticsin, Verifyn ja Erasuren konfiguraatioiden sekä testitapausten valinta ja laitesuodattimien teko tuli tehdä uudelleen käytettäväksi eri komponentteihin, jotta tulevat sovellukset pystyvät käyttämään niitä.

Yksi suunnitelluista sovelluksista kehitettiinkin samanaikaisesti Volume Workflow’n kanssa. Mobile Workflowsin vastaa Piceasoft Oy:n OnTheFly-tuotteen työnkulun määrityksestä. Mobile Workflows käyttää Volume Workflow:ta varten tehtyä yksisivuisen verkkosovelluksen pohjaa, navigointia sekä Verifyn sekä Diagnosticsin testitapausten valintaa varten tehtyjä komponentteja.

3.2 Projektin kulku

Projekti alkoi 1. päivänä huhtikuuta 2020 etätapaamisena, jossa olivat kaikki palvelintiimin Volumen työnkulun uuden hallintajärjestelmän kehitykseen liittyvät henkilöt sekä Piceasoft Oy:n CTO (chief tecnical officer). Ennen tapaamista henkilöille oli lähetetty projektinhallintaohjelmisto Jiraan tehty määrittely, jossa kuvaillaan uuden ominaisuuden tarpeita ja ominaisuuksia. Tapaamisessa selitettiin projektin tarkoitus kaikille ja mietittiin tekniseen toteutukseen liittyviä yksityiskohtia. Tapaamisessa jaoteltiin projektin vastuualueet. Aktiivisessa ohjelmistonkehityksessä oli kokonaisuudessaan neljä henkilöä.

- PiceaServices-ohjelmoija, joka vastasi Volumen työnkulun määrittelyn toteutuksesta työpöytäohjelmistossa. Tekijänä toimi PC-tiimin jäsen
- Tietokanta- ja backend-ohjelmoija, joka vastasi tietokannan toteutuksesta sekä rajapinnasta. Työnkulun hallintajärjestelmä ottaa yhteyden tähän rajapintaan tallettaakseen tietoja pysyväistalletukseen
- Backend-ohjelmoija, joka vastasi PiceaServicesin ja tietokannan välisestä rajapinnasta. Tekijänä toimi palvelintiimin jäsen
- Frontend-ohjelmoija, joka vastaa PMC:hen tehtävästä käyttöliittymästä, jota käytetään Volumen työnkulun määrittelyn tekoon. Opinnäytetyöntekijä toimi tässä roolissa osana palvelintiimiä.

Ennen tapaamista Piceasoft Oy:n UX-suunnittelija oli tehnyt luonnoksen toteutettavista käyttöliittymistä PMC:hen ja PiceaServicesiin. Tapaamisessa käytiin tehdyt luonnokset läpi, jonka jälkeen tapaaminen päätettiin ja projektin kehitystyö alkoi.

Suomessa 2020 maaliskuussa levinneen COVID-19 -taudin takia valtion toimesta oli määrätty etätyösuositus. Piceasoft Oy:n oli projektin alkaessa suositusten mukaisesti siirtynyt etätöihin pois toimistolta, joka pakotti koko projektin toteutuksen tapahtumaan etätyönä. Tämä todennäköisesti hidasti projektin etenemistä, koska työntekijöiden oli samalla totuttava uuteen etätyöskentelyyn. Erityisesti etätyö vaikeutti opinnäytetyöntekijän työskentelyä, koska asiakkuudenhallintajärjestelmä oli kokonaan uusi ja perehdytys jäi etätyöskentelyn takia vähälle.

Ensimmäisestä tapaamisesta lähtien projektin tietokantaa, palvelimessa pyörivää ns. backendiä ja käyttäjän selaimessa toimivaa ns. frontendiä ryhdyttiin kehittämään samanaikaisesti. Samanaikainen kehitys aiheutti kuitenkin hidasteita projektin frontendin kehitykseen. Sillä aikaa, kun projektin backendiä vielä kehitettiin, frontend joutui keskittymään käyttöliittymän koodaukseen. Tällöin frontend ei pystynyt suunnittelemaan tarkasti, miten tulevat työnkulun määritykseen käytettävät rajapinnat tulisivat toimimaan ja tallentamaan dataa. Tämä kuitenkin oli vain pieni hidaste, koska projektin perustana toimivaa sovellusta ja työnkulun hallintajärjestelmän käyttöliittymää pystyttiin edistämään sillä aikaa.

Ensimmäisessä seurantapalaverissa 14.4.2020 oli paikalla työnkulkua tekevät tiimit esimiehineen palvelintiimistä sekä työpöytäsovellustiimistä, UX-suunnittelija, asiakasrajapinnassa toimiva testaaja, CTO ja tuotteistuksen johtaja (Product Director). Palaverissa keskusteltiin työn edistymisestä ja mahdollisista ongelmista. Pääkohtina palaverissa olivat frontendin kannalta, miten työpöytäsovelluksessa käytettävät testit saadaan frontendiin ja mitkä Diagnostics-osion testit pystyttäisiin parametrisoimaan. Ensimmäisessä palaverissa frontendistä oli esiteltävänä vain käyttöliittymä ilman yhteyksiä backendiin.

Toinen seurantapalaveri pidettiin 6.5.2020, jolloin frontendistä oli esiteltävänä verkkosovelluksen päätoiminnallisuus paikallisesti (työnkulun ja konfiguraatioiden luonti ja editointi). Kutsuja konfiguraatioiden ja työnkulkujen tallettamiseen, muokkaamiseen tai poistamiseen palvelun backendiin ei kuitenkaan vielä oltu tehty. Palaverissa käsiteltiin jälleen työn edistymistä sekä mahdollisia ongelmia. Suurimpia palaverin kysymyksiä olivat seuraavat:

- Miten laitesuodattimet toteutettaisiin frontendiin: dropdown-valikkona vai käyttäjän syötteenä. Asiassa päädyttiin käyttäjän syötteeseen, koska dropdown-valikon ylläpitäminen kaikista käyttöjärjestelmäversioista eri alustoilla katsottiin liian suureksi työmääräksi hyötyihin nähden.
- Milloin frontendiä pystyttäisiin esittelemään asiakkaalle, joka oli esittänyt toiveita edellisen työnkulun määrittelyn parantamiseen. Asiassa päädyttiin päivämäärään 18.5.2020, jota ennen pidettäisiin 14.5.2020 sisäinen palaveri.

Kolmannessa seurantapalaverissa 14.5.2020 työnkulun määrityksen verkkokäyttöliittymästä oli valmiina MVP-versio (minimum viable product eli ensimmäinen toimiva versio tuotteesta minimiominaisuuksineen), jota asiakkaalle pystyttäisiin esittelemään. Tähän mennessä projektin backend oli valmis ja työpöytäsovellukseen toteutettava toiminto oli melkein valmis. Palaverissa käytiin läpi verkkokäyttöliittymää, jota esiteltäisiin 18.5.2020 ja mietittiin mahdollisia muutoksia tehtäväksi ennen esittelyä. Verkkokäyttöliittymään päädyttiin tekemään pari muutosta ennen asiakastapaamista.

- Diagnostics-konfiguraation testeistä tuli poistaa tuleva AI powered -testi, jolla ei ollut vielä toiminnallisuutta Volumessa.
- Tyhjien Diagnostics-testisettien näkyvistä poistaminen.

Asiakkaalle esiteltiin 18.5.2020 tapaamisessa verkkokäyttöliittymän toiminnallisuutta sekä erikseen työpöytäsovelluksen toimintaa. Asiakkaan edustajat pitivät uutta Volumen työnkulun hallintajärjestelmän uutta versioita heille toimivana ja parannuksena edellisestä versioista. Asiakkaan edustajat antoivat muutosehdotuksia verkkokäyttöliittymään, mutta annetut muutosehdotukset olivat jo työn alla. Palaverin päällimmäinen kysymys oli, milloin he pääsevät testaamaan ja tutustumaan tarkemmin uuteen työnkulun hallintajärjestelmään ja milloin se julkaistaan virallisesti. Julkaisupäivämääräksi arviotiin kesäkuun alkua, jos yllätyksiä tai suuria muutosehdotuksia ei tulisi.

28.5.2020 asiakkaan edustajille pidettiin asiakasrajapinnassa toimivan testaajan toimesta tarkempi esittely uudesta Volumen työnkulun hallintajärjestelmästä. Tapaamisessa käytiin läpi uusi työnkulun hallintajärjestelmä tarkemmin ja asiakkaan edustajat pääsivät testaamaan uutta ominaisuutta. Tapaamisessa saatiin hyvää palautetta toteutetusta ominaisuudesta.

Uusi Volumen työnkulun hallintajärjestelmä julkaistiin käyttäjille 10.6.2020.

Volumen työnkulun hallintajärjestelmän työstäminen ei kuitenkaan päättynyt tähän. Virallisen julkaisun jälkeen hallintajärjestelmään päätettiin tehdä vielä lisäparannuksia ja -ominaisuuksia.

UX-suunnittelijan sekä asiakasrajapinnassa toimivan testaajan kanssa pidettiin palaveri 9.6.2020, jossa käytiin läpi työnkulun hallintajärjestelmän käyttöliittymään tehtäviä parannuksia. Käyttöliittymästä löydettiin parannuskohteita 12 kappaletta. Suurin osa ehdotuksista oli pieniä käyttöliittymään tehtäviä parannuksia, kuten painikkeiden sävyjen vaihtamisia, tekstikentän muuttamisia pienemmäksi tai käyttöliittymän mittasuhteiden muutoksia. Palaverissa tuli ilmi kuitenkin kaksi suurta muutosehdotusta.

- Automaattinen konfiguraation ja säännön nimen täyttö sekä nimikenttään kursorin kohdistus. Työnkulun hallintajärjestelmän käyttöliittymärakenteen vuoksi käyttäjät avaavat konfiguraatioita ja työnkulun määrityksiä tehdessään monta ponnahdusikkunaa, joihin käyttäjän tulee täyttää konfiguraatioiden ja sääntöjen nimiä. Tehtävä muutos vähentää käyttäjältä aikaa tekstikenttien täyttämiseen ja niihin huomion kiinnittämiseen sekä klikkaamiseen.
- Toinen muutosehdotus oli laitesuodattimien lisäykseen käytetyn ikkunan parantelu. Muutosehdotuksessa valitut laitteet näytettäisiin pudotusvalikkojen alla, jolloin käyttäjä pysyy paremmin selvillä lisätyistä suodattimista.

Käyttöliittymäparannuksien lisäksi Volumen työnkulun hallintajärjestelmään päätettiin lisätä vielä kaksi lisäominaisuutta, joista toinen oli asiakkaan pyytämä.

- Asiakkaan pyytämä lisäominaisuus oli laitesuodattimien tuonti CSV-tiedostosta työnkulun hallintajärjestelmän Verify- ja Diagnostics-sääntöihin. Tällä tavoin asiakas pystyi hallitsemaan laitteitaan taulukkolaskentaohjelmassa ja tuomaan tarvittaessa laitteet verkkosovellukseen.
- Toinen lisäominaisuus oli lisätä Diagnostics-säännön näytönkosketustestiin lisävaihtoehtoja, joista käyttäjä voi valita millaisella kuviolla näytönkosketustesti suoritettaisiin.

Opinnäytetyön laajuus rajattiin näihin lisäominaisuuksiin sekä parannuksiin, koska ne tulivat heti julkaisun jälkeen.

3.3 Tilastoja verkkosovelluksen toteutuksesta

Työn päävaihe kesti 70 päivää. Ensimmäinen projektin työpäivä oli 1.4.2020 ja viimeinen päävaiheen työpäivä oli 10.6.2020, jolloin työnkulun hallintajärjestelmä julkaistiin ensimmäisen kerran. Työaikaa tällä ajanjaksolla käytettiin Volumen työnkulun hallintajärjestelmään noin 245 tuntia.

Työhön lisättäviä parannuksia ja lisäominaisuuksia tehtiin aikajaksolla 11.6.2020 – 22.6.2020. Tällä aikajaksolla työaikaa kertyi noin 30 tuntia työnkulun hallintajärjestelmään.

Yhteensä tuntityöaikaa kertyi palavereineen opinnäytetyöntekijän osalta pelkästään työnkulun hallintajärjestelmän tekoon 275 tuntia.

Verkkosovelluksen osalta Volumen työnkulun hallintajärjestelmä jakautui 32 tiedostoon. Työnkulun hallintajärjestelmän päätiedosto oli suurin, 843 riviä. Pienin tiedosto oli 5 riviä, jossa oli sovelluksessa käytettäviä vakiomuuttujia. Pienin ohjelmistoa sisältävä tiedosto oli 8 riviä. Koodirivejä opinnäytetyössä oli 4755, jolloin tiedoston keskimääräinen pituus oli noin 149 riviä.

Keskimääräinen tunnissa syntynyt koodimäärä oli noin 17,3 riviä.

3.4 Volumen työnkulun hallintajärjestelmän toiminnallisuus

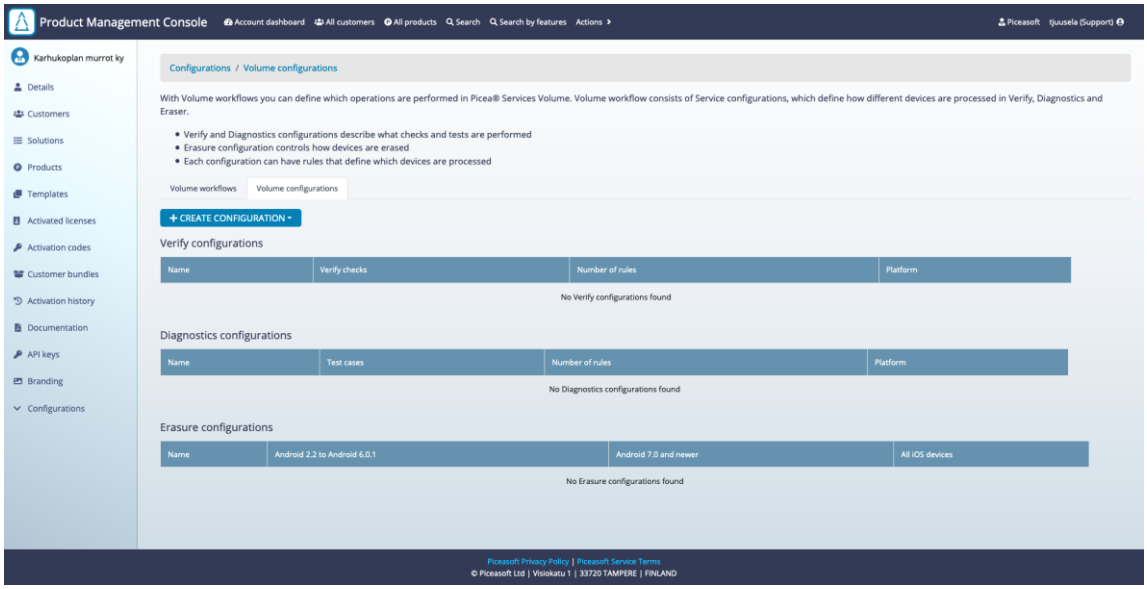
3.4.1 Päänäkymä

Verkkosovelluksen toiminnallisuutta pystytään ymmärtämään paremmin, jos seurataan käyttäjän matkaa työnkulun hallintajärjestelmässä ja uuden työnkulun luonnissa.

Asiakkuudenhallintajärjestelmä PMC:ssä Volumen työnkulunhallintajärjestelmän virallinen nimi on Volume workflows. Uuden Volumen työnkulun luonti alkaa painamalla PMC:n navigointiin käytetystä sivupalkista kohtaa ”Configurations”, josta löytyy ”Volume workflows”. Volumen työnkulun hallintajärjestelmä koostuu kahdesta alinäkymästä:

- Tehtyjen Volumen työnkulkujen listaukseen käytetty Volume workflows -alinäkymä
- Tehtyjen Verify- Diagnostics- ja Erasure-konfiguraatioiden listaukseen (Volume configurations) käytetty alinäkymä. Volumen työnkulku koostuu näistä konfiguraatioista, jotka näkyvät listassa.

Näistä alinäkymistä (konfiguraationäkymä kuvassa 4) käyttäjä voi hallinnoida asiakkaan työnkulkuja ja konfiguraatioita.

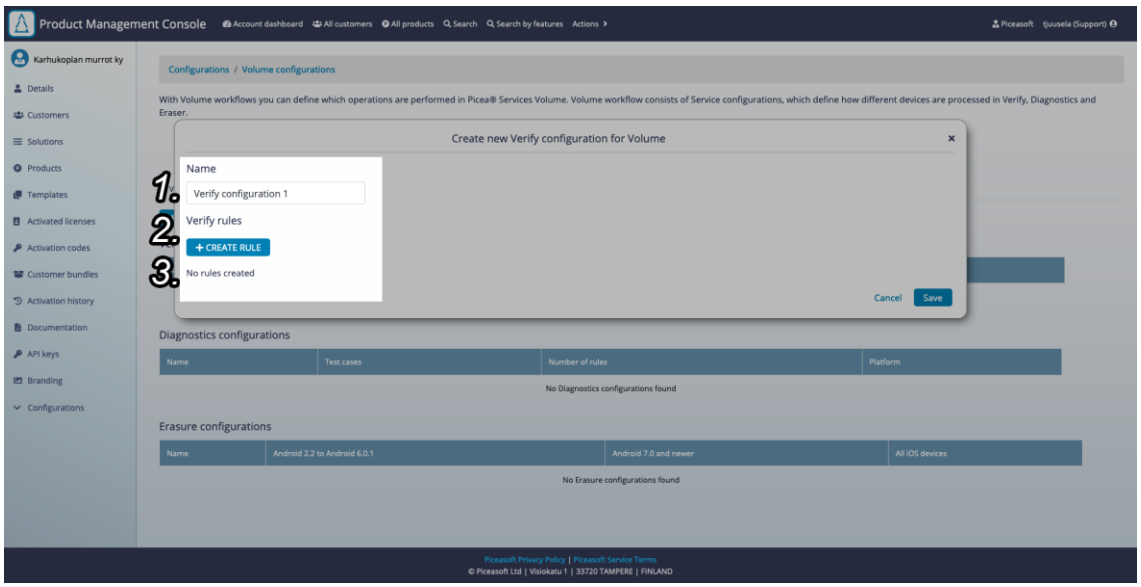


KUVA 4. Kuvakaappaus työnkulun hallintajärjestelmän konfiguraatiot-näkymästä, jossa ei ole konfiguraatioita luotuna. ”Create Configuration” -napin päällä on näkymien vaihtamiseen käytetyt välilehtipainikkeet.

Yksi luotu Volumen työnkulku vaatii vähintään yhden Verify-, Diagnostics- tai Erasure-konfiguraation, jota käytetään työnkulussa. Käyttäjä voi luoda uuden konfiguraation painamalla Volume konfiguraatiot -sivulta pudotusvalikon avaavaa painiketta ”Create configuration” (painike näkyvissä kuvassa 4). Painamalla nappia käyttäjälle aukeaa pudotusvalikko, josta hän voi valita, minkä konfiguraation hän haluaa luoda.

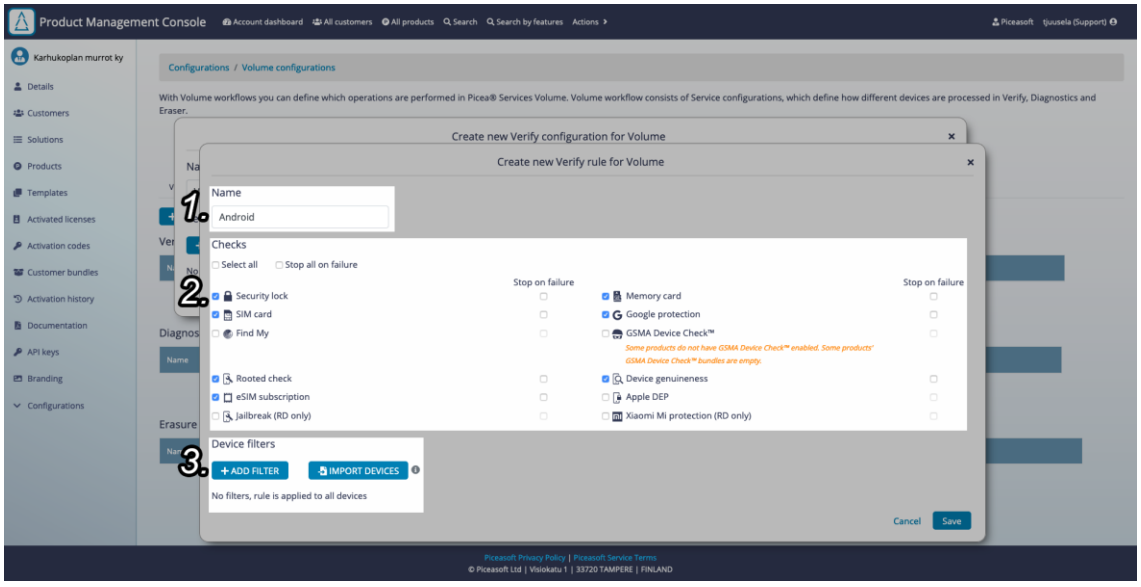
3.4.2 Verify-konfiguraatio luonti

Painaessa ”Create configuration” -pudotusvalikosta Verify-valintaa avautuu näytölle ponnahdusikkuna konfiguraation luomiseen (kuva 5). Ponnahdusikkunassa käyttäjä voi syöttää konfiguraation nimen sekä luoda uuden säännön konfiguraatioon. Jotta konfiguraation voi tallentaa, käyttäjän tulee luoda siihen vähintään yksi sääntö.



KUVA 5. Kuvakaappaus työnkulun hallintajärjestelmän Verify-konfiguraation luonnista. 1: Konfiguraation nimikenttä. 2. Uuden säännön luontiin käytetty painike. 3: Konfiguraation säännöt, jos niitä on luotu.

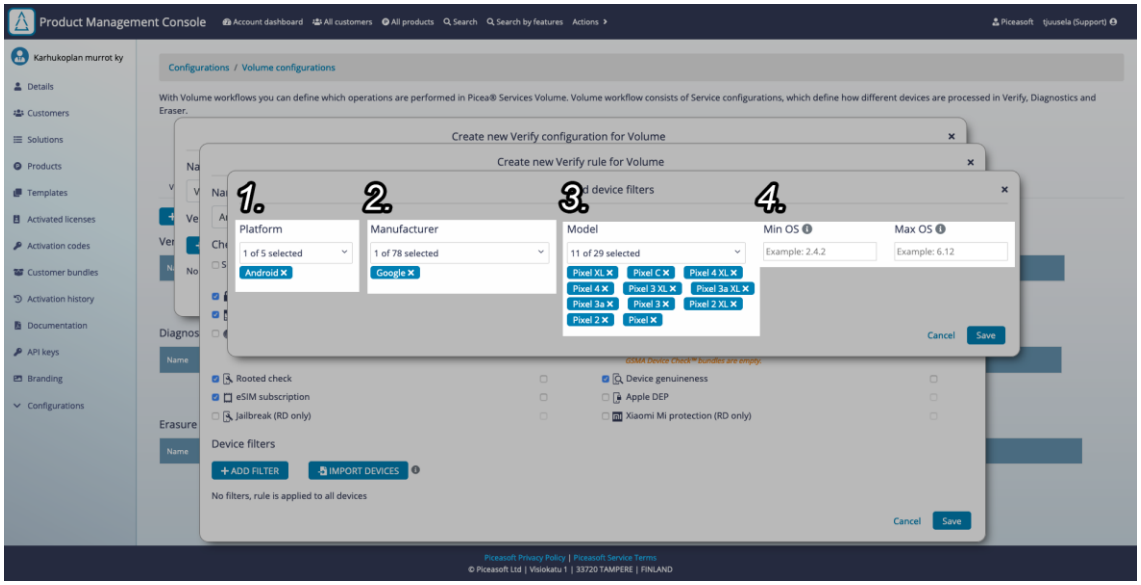
Luodakseen uuden säännön käyttäjä painaa ”Create rule” -painiketta, josta hänen eteensä aukeaa uuden Verify-säännön (kuva 6) luontiin käytetty ponnahdusikkuna. Ikkunasta käyttäjä löytää Verify-säännön testit, joita suoritetaan puhelimelle. Käyttäjä voi valita haluamansa testit painamalla testin nimen vieressä olevaa valintaruutua. Testeistä voidaan myös valita, keskeytetäänkö testien suorittaminen, jos valitun testin tulos on negatiivinen. GSMA Device Check™ -testin kohdalla käyttäjälle näytetään varoitus, jos hänen tuotteillaan tai käyttäjällä ei ole testiin vaadittuja ominaisuuksia päällä tai käyttökertoja jäljellä.



KUVA 6. Kuvakaappaus työnkulun hallintajärjestelmän Verify-säännön luonnista. 1: Säännön nimikenttä. 2: Säännön testitapaukset. 3: Säännön laitesuodattimet ja niiden lisäykseen käytetyt painikkeet.

Seuraavaksi seurataan esimerkkikäyttäjän etenemistä työnkulunhallintajärjestelmässä.

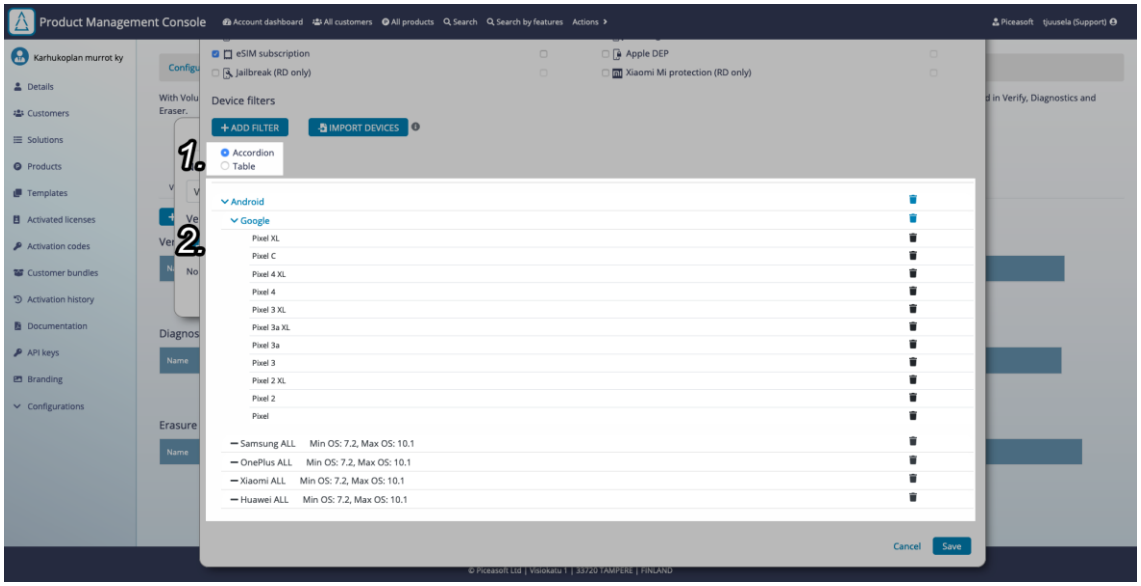
Käyttäjä haluaa luoda Android-laitteille säännön, jossa määritellään mitä testejä ajetaan. Jotta säännön testit ajettaisiin vain Android-laitteilla, sääntöön on lisättävä laitesuodatin. Laitesuodattimien (lisäykseen käytetty ponnahdusikkuna kuvassa 7) avulla määritellään, mille puhelimelle tai puhelinalustoille testit suoritetaan. Jos suodattimia ei lisätä sääntöön, säännössä määritellyt testit suoritetaan kaikilla mahdollisilla puhelimilla. Suodattimien lisäys tapahtuu painamalla ”Add filter” -painiketta, josta käyttäjälle avautuu uusi ponnahdusikkuna laitteiden lisäykseen.



KUVA 7. Kuvakaappaus työnkulun hallintajärjestelmän laitesuodattimien lisäyksestä Verify-sääntöön. 1: Laitesuodattimen alusta-pudotusvalikko (iPhone, Android…). 2: Laitesuodattimien valmistaja-pudotusvalikko. 3: Laitesuodattimien puhelinmalli-pudotusvalikko. 4: Minimi- ja maksimikäyttöjärjestelmäkentät.

Käyttäjä päättää, että hän haluaa testinsä ajettavan Googlen Pixel -puhelimilla, jolloin hän valitsee pudotusvalikoista alustaksi Androidin, valmistajaksi Googlen ja puhelimiksi kaikki Googlen Pixel-mallit. Käyttäjä voi myös lisätä suodattimiin

minimi- ja maksimikäyttöjärjestelmäversion, jolla testit ajetaan. Lisäyksen jälkeen laitesuodattimet näkyvät säännön alaosassa.



KUVA 8. Kuvakaappaus työnkulun hallintajärjestelmän Verify-säännön laitesuodattimista haitarinäkymänä. 1: Haitari- ja listanäkymän vaihtamiseen käytetyt painikkeet. 2: Sääntöön lisätyt laitesuodattimet.

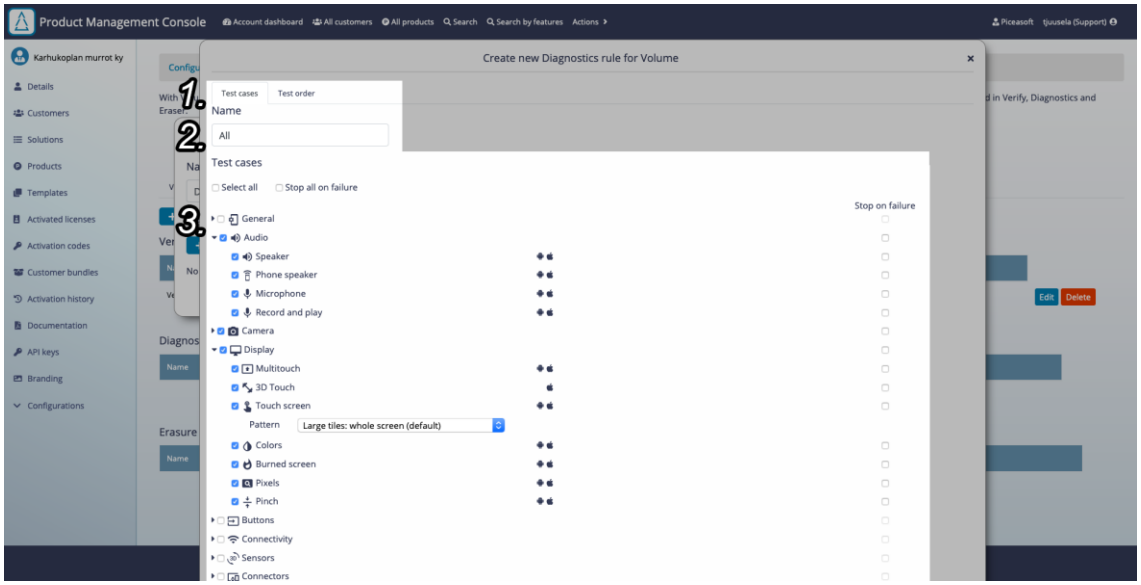
Käyttäjä päätti lisäksi, että säännön testit ajettaisiin myös kaikilla laitteilla, joiden valmistaja on joko Samsung, OnePlus, Xiaomi tai Huawei ja joiden Android-versio on 7.2:n ja 10.1:n välillä. Käyttäjä voi tarkastella säännön laitesuodattimia kahdella eri tavalla: haitari- tai listanäkymässä (kuva 8). Haitarinäkymässä puhelimet on ryhmitelty alustojen ja valmistajien alle ja listanäkymässä kaikki suodattimet näytetään yksi kerrallaan listana. Tämän jälkeen käyttäjä tallentaa säännön ja päättää tehdä samantyyppisen säännön iPhone-puhelimille, minkä jälkeen hän tallentaa konfiguraation.

3.4.3 Diagnostics-konfiguraation luonti

Jatketaan esimerkkikäyttäjän etenemisen seuraamista työnkulunhallintajärjestelmässä.

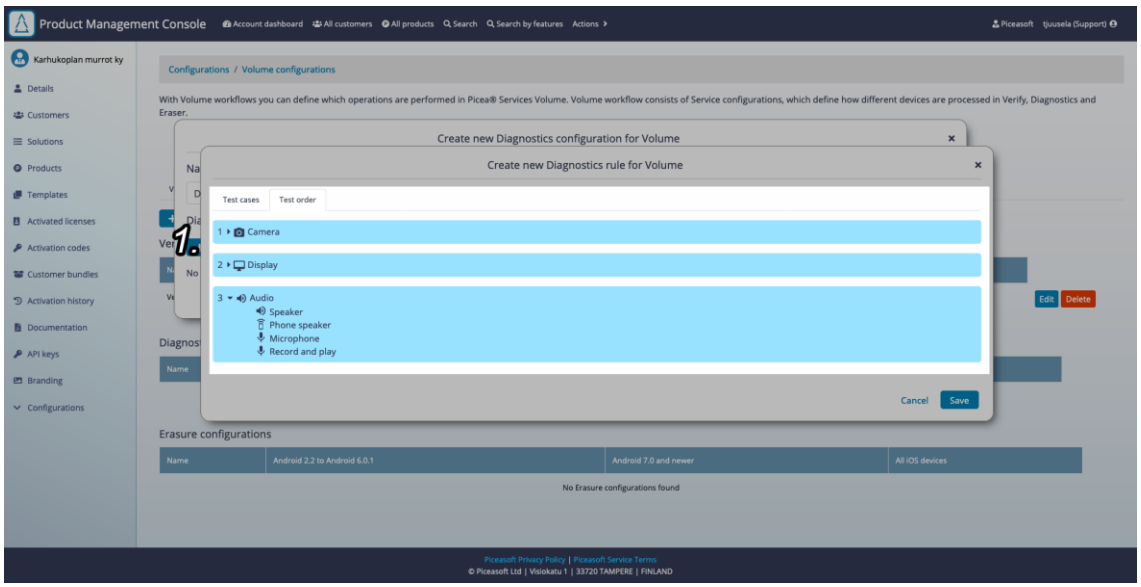
Käyttäjä haluaa, että puhelimelle tehdään kaikki mahdolliset operaatiot, joten seuraavaksi hän tekee uuden Diagnostics-konfiguraation. Diagnostics-konfigu-

raation luonti toimii samalla tavalla kuin Verify-konfiguraation, jolloin yksi konfiguraatio pitää sisällään monta erillistä sääntöä ja säännöille pystytään lisäämään laitesuodattimia. Diagnostics-konfiguraatioiden luonti eroaa Verify-konfiguraatioiden luonnista konfiguraatioiden sääntöjen testeissä. Diagnostics-sääntöjen valintaan käytetty ponnahdusikkuna on näkyvissä kuvassa 9.



KUVA 9. Kuvakaappaus työnkulun hallintajärjestelmän Diagnostics-säännön testien valinnasta. 1. Diagnostiikan testitapausten ja testisettien järjestyksen välillä vaihtamiseen käytetyt välilehtipainikkeet. 2: Säännön nimikenttä. 3: Diagnostiikan testitapaukset. Kosketusnäyttötestin (Touch screen) lisäparametri näkyvissä kuvassa.

Diagnostics-säännössä (kuva 9) on Verify-konfiguraatioiden tapaan testejä, mutta testit on jaettu ns. setteihin. Setit ovat testien ”ryhmiä”, jotka auttavat käyttäjää hahmottamaan testien kategoriat. Kahdessa testissä on lisäksi mahdollisuus antaa lisäparametrejä: puhelimen kosketusnäyttötestin kuvion valinta sekä akkutestin kesto. Testejä voidaan Verify-konfiguraation tapaan valita yksi kerrallaan tai käyttäjä voi valita setin, joka asettaa kaikki setin testit päälle. Diagnostics-säännössä testisettien järjestystä voidaan myös muuttaa (kuva 10). Järjestyksen muuttaminen tapahtuu menemällä ponnahdusikkunan yläreunassa olevaan välilehteen ”Test order”. Käyttäjä näkee täällä valitsemansa testisetit, joiden testejä hän on lisännyt sääntöönsä. Käyttäjä voi raahaamalla testisettien nimiä muuttaa testien suoritusjärjestystä.



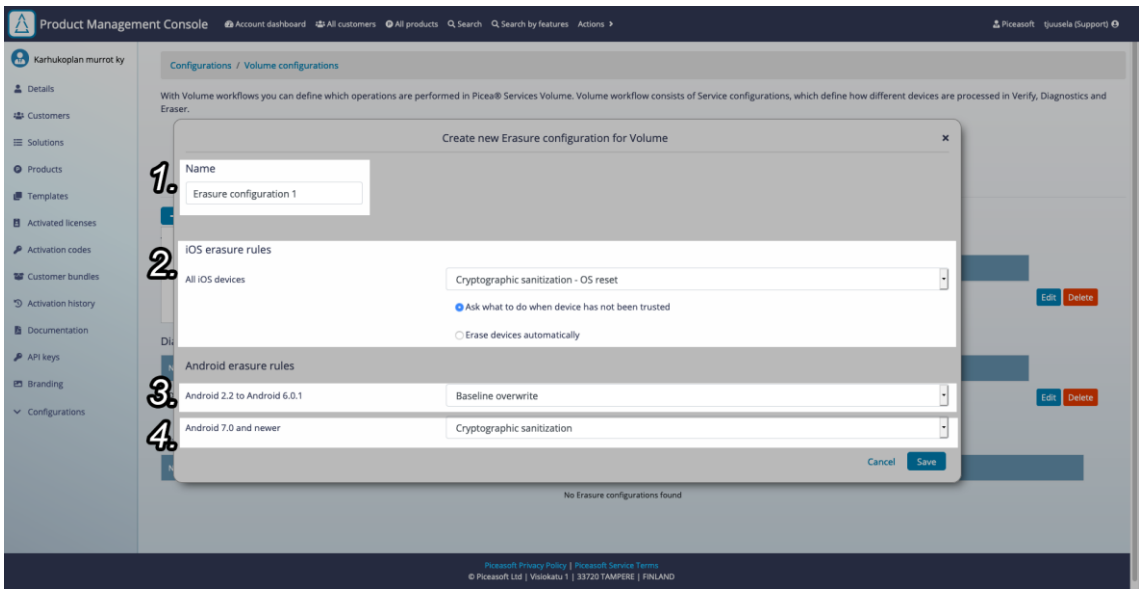
KUVA 10. Kuvakaappaus työnkulun hallintajärjestelmän Diagnostics-sääntöjen testisettien järjestyksen muuttamisesta. 1: Diagnostiikassa aktiivisena olevat testisetit. Avaamalla testisetin voidaan tarkistaa mitkä testit ovat päällä. Settien suoritusjärjestystä voidaan muuttaa raahaamalla.

Käyttäjä ei lisää Diagnostics-sääntöönsä laitesuodattimia, jolloin sääntö suoritetaan kaikilla laitteilla. Tämän jälkeen käyttäjä tallentaa säännön ja konfiguraation.

3.4.4 Erasure-konfiguraation luonti

Jatketaan esimerkkikäyttäjän etenemisen seuraamista työnkulunhallintajärjestelmässä.

Käyttäjällä on enää jäljellä Erasure-konfiguraation luonti (kuva 11). Erasure-konfiguraation luonti eroaa huomattavasti kahdesta edellisestä. Erasure-konfiguraatioissa valitaan vain eri alustojen muistin pyyhintään käytetyt metodit. Pyyhintämetodien valinta on jaettu kolmeen eri kategoriaan: Android 2.2 – Android 6.0.1, Android 7 – uudemmat ja viimeisenä iOS-laitteet. iOS-laitteissa muistinpyyhintämetodin ollessa ”Cryptographic sanitization – OS Reset” käyttäjälle annetaan vaihtoehto, miten edetään, jos pyyhittävä laite ei ole hyväksynyt ajettavaa ohjelmistoa.

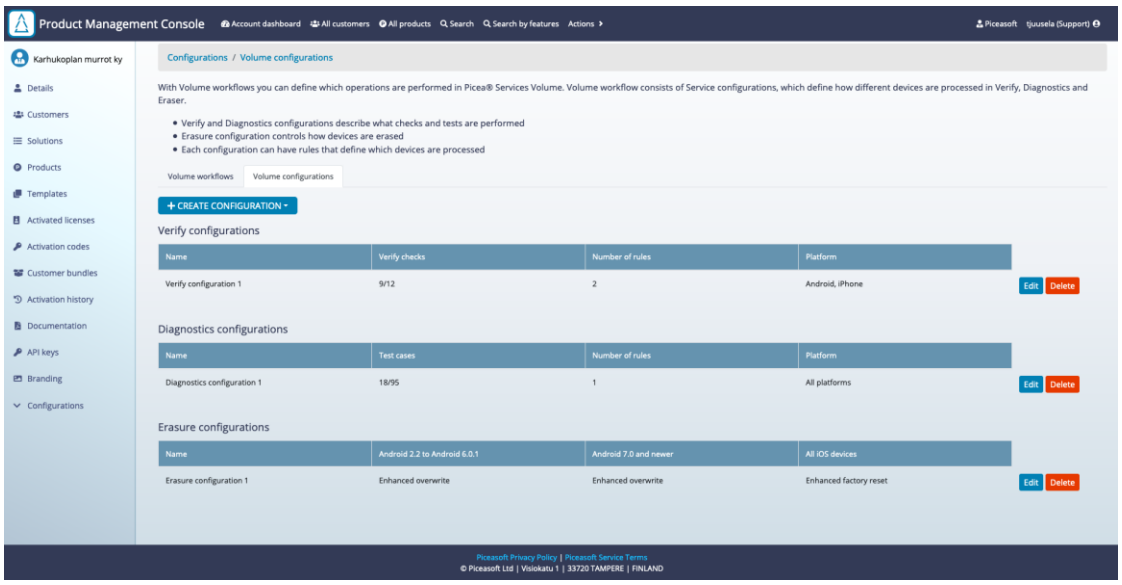


KUVA 11. Kuvakaappaus työnkulun hallintajärjestelmän Erasure-konfiguraation luonnista. 1: Konfiguraation nimikenttä. 2: iOS-laitteiden muistinpyyhintään käytetty metodi. 3: Android 2.2-6.0.1 -laitteiden muistinpyyhintään käytetty metodi. 4: Android 7-uudemmat -laitteiden muistinpyyhintään käytetty metodi.

Erasure-konfiguraatioissa ei ole sääntöjä tai laitesuodattimia, jolloin käyttäjä tallettaa Erasure-konfiguraation.

3.4.5 Volumen konfiguraatiot -näkymä

Erasure-konfiguraation jälkeen käyttäjä on luonut kaikki haluamansa konfiguraatiot, jotka hän aikoo liittää Volumen työnkulkuunsa. Käyttäjä voi tarkastella luomiaan konfiguraatioita päänäkymän Volumen konfiguraatiot -näkymästä (kuva 12), jossa konfiguraatiot ovat listattuina omien operaatioiden mukaan. Konfiguraatioista näytetään listoissa tärkeimmät tiedot, kuten Verify'ssä ja Diagnostics'ssä olevien testien määrät, Verifyn ja Diagnosticsin alustat ja Erasuren pyyhintämetodit.



KUVA 12. Kuvakaappaus työnkulun hallintajärjestelmän konfiguraatiolistasta.

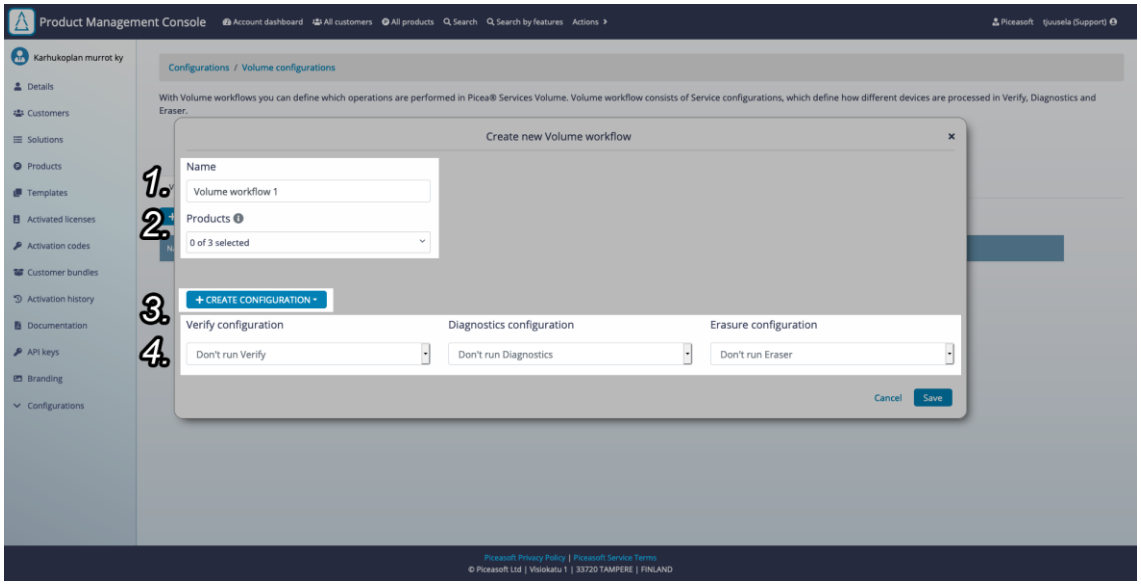
Volume konfiguraatiot -näkymästä käyttäjä voi halutessaan myös muokata tai poistaa konfiguraation. Muokkaaminen tapahtuu painamalla konfiguraation perässä olevaa ”Edit”-nappia, josta käyttäjälle avautuu samanlainen ikkuna kuin konfiguraation luonnissa, mutta muokkaamista varten. Konfiguraation poistaminen tapahtuu ”Delete”-nappia painamalla, jonka jälkeen käyttäjältä varmistetaan vielä ponnahdusikkunan avulla, että hän varmasti haluaa poistaa konfiguraation.

3.4.6 Volumen työnkulun luonti

Jatketaan esimerkkikäyttäjän etenemisen seuraamista työnkulunhallintajärjestelmässä.

Käyttäjä on luonut kaikki konfiguraatiot, joita hän aikoo käyttää työnkulussaan. Uuden työnkulun luonti (kuva 13) alkaa Volumen työnkulut -näkymästä, jossa on nappi ”Create Workflow” (kuvassa 14). Nappia painaessa käyttäjälle avautuu uusi ponnahdusikkuna. Käyttäjä voi määrittää työnkululle ikkunassa seuraavat asiat:

- nimen, joka näkyy PiceaServices -työpöytäsovelluksessa Volumen työnkulkua valittaessa.
- tuotteet, joilla kyseinen työnkulku on käytettävissä.
- Verify-, Diagnostics- ja Erasure-konfiguraatiot, joita käytetään työnkulussa.



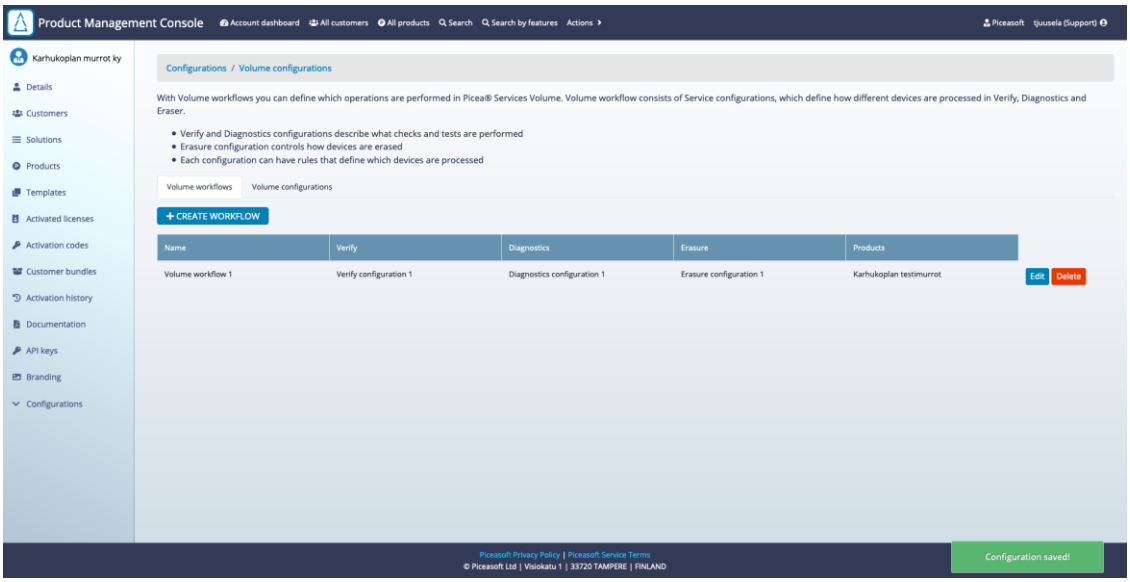
KUVA 13. Kuvakaappaus työnkulun hallintajärjestelmän työnkulun luonnista. 1: Työnkulun nimi. 2: Työnkulun tuotteet. 3: Konfiguraatioiden lisäämiseen käytetty painike. 4: Verify-, Diagnostics- ja Erasure-konfiguraatioiden valitsemiseen käytetyt pudotusvalikot.

Käyttäjä voi valita työnkulkuun vain tuotteita, joista on asetettu Volumen Configuration API käyttöön. Työnkulkua käyttävien tuotteiden valinnan jälkeen käyttäjä valitsee, mitä konfiguraatioita työnkulku käyttää. Konfiguraatioiden valinta tapahtuu pudotusvalikoista. Pudotusvalikoissa voidaan jättää pois konfiguraatio, jolloin kyseistä operaatiota ei ajeta Volumessa. Yhdessä työnkulussa tulee kuitenkin olla vähintään yksi konfiguraatio päällä, jotta konfiguraatio voidaan luoda. Ponnahdusikkunasta voidaan myös luoda konfiguraatioita painamalla ”Create Configuration” -nappia, josta avautuu halutun konfiguraation luontiin tarkoitettu ponnahdusikkuna.

Kun käyttäjä on tallettanut työnkulun, hän pystyy käyttämään sitä PiceaServices -työpöytäsovelluksessa, josta hän voi valita mitä työnkulkua Volume käyttää.

3.4.7 Volumen työnkulut -näkymä

Käyttäjä voi hallita luomiaan työnkulkuja Volume työnkulut -näkymästä (kuva 14). Näkymässä käyttäjä näkee kaikki asiakkuuden työnkulut ja voi halutessaan poistaa tai muokata niitä listasta löytyvistä painikkeista. Listasta löytyvät myös tiedot työnkulkuun kuuluvista konfiguraatioista.



KUVA 14. Kuvakaappaus työnkulun hallintajärjestelmän työnkulkujen listauksesta. Oikeassa alakulmassa näkyvissä ilmoitus uuden työnkulun luonnin onnistumisesta.

3.5 Lisäominaisuudet

Alkuperäisten ominaisuuksien, jotka kuvattiin kappaleessa 3.4, Volumen työnkulun hallintajärjestelmään toteutettiin lisäksi kaksi lisäominaisuutta ensimmäisen julkaisun jälkeen.

3.5.1 Laitesuodattimien tuonti CSV-tiedostosta

Ensimmäinen lisäominaisuus on laitesuodattimien tuonti CSV-tiedostosta Verify- ja Diagnostics-konfiguraatioiden sääntöihin (lisäyksen avaava painike näkyvissä kuvassa 6). Tämä lisäominaisuus lisättiin asiakkaan toiveesta ja se mahdollistaa laitesuodattimien hallinnan asiakkaille itsenäisesti omissa järjestelmissään. CSV-

tiedostojen avulla käyttäjät voivat tuoda tuhansia laitteita järjestelmään hetkessä käyttämällä seuraavaa formaattia:

*alusta, *valmistaja, *malli, *minimikäyttöjärjestelmäversio, *maksimikäyttöjärjestelmäversio*

tai

*alusta, *valmistaja, *malli*

Tähdillä merkityt kohdat voidaan korvata *-merkillä, jolloin laitesuodattimiin tulkitaan, että kaikki kyseisen valmistajan, mallin tai käyttöjärjestelmäversion puhelimet kuuluvat suodattimeen. Jos minimi- tai maksimikäyttöjärjestelmäversiot ovat jätetty riviltä pois, tai ne on merkitty tähdillä, sovellus jättää kyseisen käyttöjärjestelmäversio-suodatuksen pois.

Laitesuodattimet lisätään Diagnostics- ja Verify sääntöjen ”Import devices” -painikkeesta. Painike avaa tiedoston valintaan käytetyn ikkunan, josta voidaan valita vain CSV-tiedostoja. Tiedoston valinnan jälkeen verkkosovellus lukee ja käsittelee tiedoston sisällön. Jos tiedoston sisältö on validi, laitteet lisätään säännön suodattimiin ja jos ei, käyttäjälle näytetään virheilmoitus.

3.5.2 Diagnostics-säännön kosketusnäyttötestin lisäparametrisointi

Toinen lisätty ominaisuus on Diagnostics-säännön kosketusnäyttötestin (Touch screen, kuvassa 9) lisäparametrisointi. Ominaisuuden avulla käyttäjä voi valita, millaisen kuvion käyttäjän tulee näytölle piirtää, jotta nähdään, toimiiko kosketusnäyttö kaikkialla. Kuvioiden avulla näyttötestin tarkkuus ja nopeus vaihtelevat: mitä tarkempi, sitä hitaampi ja mitä epätarkempi, sitä nopeampi.

Näyttötestin kuvion valinta päätettiin tuoda myös työnkulkujen hallintajärjestelmään, jotta Volumea käytettäessä pystytään määrittelemään tarvittaessa puhelimien näyttötestien tarkkuus ja nopeus.

4 TEKNINEN TOTEUTUS

4.1 Ympäristö ja kirjastot

Product Management Console eli PMC on Express-kirjastoa käyttävänä Node.js-sovellus. Frontend on toteutettu hybrid-verkkosovelluksena, joka käyttää EJS-mallinnusjärjestelmää yksinkertaisempiin sivuihin, kuten asiakkaan tietojen näyttämiseen. Näissä sivuissa käytetään myös jQuery-sovelluskirjastoa sivujen toiminnallisuuden mahdollistamiseen. PMC:n vaativammat ominaisuudet on toteutettu React-alisovelluksina, kuten käsiteltävä Volumen työnkulun hallintajärjestelmä. React-sovellukset ovat upotettuja EJS-järjestelmällä tehtyihin sivuihin, jotka tarjoavat sovelluksille navigaatioon käytetyt ylä- ja sivupalkit.

React-sovelluksien kehittämisessä käytetään JSX-syntaksilaajennuskirjastoa, Babel-kääntäjää sekä Webpack-moduulien rakentajaa. React-kirjaston versio on 16.7, jolloin React-kirjaston Hooks-toiminnallisuutta ei ole mahdollista käyttää.

Työnkulun hallintajärjestelmän toteuttamiseen on käytetty lisäkirjastona react-dropdown-select:iä, jonka avulla monivalintaiset pudotusvalikot on toteutettu.

4.2 Yksisivuinen verkkosovelluskirjasto React

React on sovelluskirjasto, jota käytetään tässä projektissa yksisivuisen verkkosovelluksen toteutukseen.

React-kirjastolla verkkosovellukset jaetaan komponentteihin, jotka ovat joko tilallisia Javascript-luokkia tai tilattomia funktioita. Tilallisilla komponenteilla on nimensä mukaisesti tila, joiden avulla ne voivat itsenäisesti hallita omia tietojaan erillään muista ja muokata näitä tietoja. (React. n.d. Components and props.) Komponenteille pystytään antamaan parametrejä (React kutsuu näitä propertyiksi), joita komponentti käyttää itsensä piirtämiseen ja esimerkiksi tilan alustamiseen.

Sovellus koostuu React-komponenteista ja niiden alikomponenteista. Tärkeässä asemassa React-kehityksessä on Javascriptin avainsana ”this”. Javascript määrittää this-avainsanan suorituksen yhteydessä riippuen funktion kutsumisen ehdoista (Kyle Simpson this & object prototypes). Yleensä React-kehityksessä tilallisissa komponenteissa this-avainsanalla viitataan komponentin olion sisäisiin funktioihin ja muuttujiin, mutta opinnäytetyössä on käytetty this-avainsanaa epätyypillisellä tavalla React-sovelluksessa, mitä käsitellään luvussa 4.4.

4.3 Pääsovellus

Volumen työnkulun hallintajärjestelmää varten päätettiin tehdä uusi verkkosovellus, johon voitaisiin tulevaisuudessa tehdä kaikki työnkulun hallintajärjestelmän kaltaiset konfiguraatiot. Tätä varten luotiin uusi React-sovellus configurations_app.jsx, joka hallinnoi eri alikonfiguraatiosovelluksien piirtämistä sekä navigointilogiikkaa. Alisovelluksien piirtämistä hallinnoidaan selaimen URL-osoitteen sekä komponentin tilan (state) muuttujan avulla. Tilan muuttujassa säilötään piirrettävän komponentin nimeä, jonka avulla switch-lauseke päättää piirrettävän alisovelluksen. Tilan muuttujaa voidaan muuttaa configurations_app.jsx:n navigaatiofunktiolla, joka annetaan alisovelluksille parametreinä. Navigaatiofunktiota kutsuttaessa tilan muuttamisen lisäksi funktio käyttää selainympäristön history.pushState()-funktiota, jonka avulla selaimen nykyinen osoite muutetaan configurations_app.jsx:n alisovellusta vastaavaksi ja käyttäjän selaushistoriaan saadaan merkintä alisovelluksesta. Historiamerkintä on tärkeä sen vuoksi, että configurations_app.jsx tarkistaa ensimmäistä kertaa käynnistyessään URL-osoitteen. Kun alisovelluksen nimi havaitaan osoitteessa, asetetaan navigoimiseen käytetyn tilamuuttuja kyseiseksi, jolloin alisovellus piirretään. Tällöin mahdollistetaan, että configurations_app.jsx piirtää oikean alisovelluksen käyttäjän navigoidessa suoraan alisovelluksen osoitteeseen. Esimerkiksi:

pmc-piceasoft.com/customer/ASIAKKAAN-TUNNUS/configurations/volume

Pääsovellus osaa kaapata /volume-päätteen, jolloin Volumen työnkulun hallintajärjestelmän -alisovellus piirretään. Historiatietojen lisääminen mahdollistaa verkkoselainten taaksepäin-napin toiminnan sekä sen, että käyttäjä näkee historiassaan käyneensä eri alisovelluksissa.

configurations_app.jsx piirtää oletusarvoisesti alisovelluslistan, jonka avulla käyttäjä voi navigoida haluamaansa alisovellukseen, mutta tämä ei ole ainoa tapa, jonka avulla käyttäjä pääsee alisovelluksiin. PMC:n sivupalkista käyttäjä pystyy myös navigoimaan suoraan alisovelluksiin tai alisovelluslistaan.

Alikomponenttien piirtämisen ja navigoinnin logiikan lisäksi configurations_app.jsx hakee useille alikomponenteille yhteisiä tietoja backendiltä, jonne tehtiin oma rajapinta sovellusta varten. Yhteisiä tietoja ovat esimerkiksi Verifyn ja Diagnosticsin testien tiedot (testin nimi, testin id, testin ikonin nimi, jne), Erasuren pyyhintämetodien tiedot, Piceasoft Oy:n sisäisesti käyttämien puhelinalustojen identifikaattorit ja Piceasoftin laitetietokannasta haetut puhelinalustat, valmistajat ja mallit. Näiden lisäksi configurations_app.jsx antaa alikomponenteilleen sovelluksen käyttämät lokalisaatiotiedot.

4.4 Volumen työnkulun hallintajärjestelmän arkkitehtuuri

Työnkulun hallintajärjestelmän pääkomponenttina toimii volume_main.jsx, joka hallinnoi verkkosovelluksen logiikkaa, piirtämistä sekä komponentin tila toimii tietovarastona.

Työnkulun hallintajärjestelmän tilan tiedot voidaan jakaa kolmeen eri aihealueeseen.

- Käyttöliittymän tiedot
- Tuote-, työnkulku- ja konfiguraatiodata, jotka on haettu backendiltä
- Ponnahdusikkunoissa käsiteltävien konfiguraatioiden, sääntöjen, laitesuodattimen ja työnkulun data

Käyttöliittymän tilatiedoissa säilytetään tietoa siitä, mitä käyttöliittymäkomponentteja tulee piirtää. Tilatiedoista löytyy tieto myös kumpi taustanäkymistä (työnkulku- tai konfiguraatio-listausnäkymä) piirretään sekä tieto, mitkä ponnahdusikkunoista (konfiguraation luonti-, sääntö-, laitesuodatinikkunat, jne.) piirretään. Tietoja muokataan aina käyttäjän painaessa käyttöliittymässä olevaa nappia välilehden vaihtamiseen tai ponnahdusikkunan avaamiseen ja sulkemiseen.

Tuote-, työnkulku- ja konfiguraatiodata haetaan volume_main.jsx:n tilaan heti komponentin rakennettua itsensä valmiiksi (React-komponenttien funktio compenentDidMount). Työnkulku- ja konfiguraatiodata haetaan työnkulun hallintajärjestelmää varten tehdyltä rajapinnalta, joka kommunikoi tietokannan kanssa. Tuotetiedot haetaan PMC:n yleiseltä tuoterajapinnalta. Työnkulku- ja konfiguraatiodata annetaan välilehtikomponenteille, jotka listaavat työnkulut ja konfiguraatiot. Tuotedataa käytetään uuden työnkulun luonnissa, kun käyttäjän tulee valita pudotusvalikosta käytettävät tuotteet. Työnkulku- ja konfiguraatiodata haetaan tarvittaessa rajapinnalta uudelleen, kun verkkosovellus saa vastauksen työnkulun tai konfiguraation luonti-, muokkaus- tai poistopyyntöön.

Työnkulun hallintajärjestelmän rajapintaa (rajapinta ei ole osa opinnäytetyötä) voitaisiin parantaa nykyisestä muokkaamalla rajapinnan luonti-, muokkaus- tai poistopyynnön vastetta. Jos pyyntöihin vastattaisiin palauttamalla päivitetty työnkulku- tai konfiguraatiodata, verkkosovelluksen ei tarvitsisi odottaa vastausta pyyntöön. Nykyinen rajapinta palauttaa vain tiedon operaation onnistumisesta. Tämän takia nykyinen työnkulun hallintajärjestelmä joutuu lähettämään uuden pyynnön työnkulku- tai konfiguraatiodatasta heti luonti-, muokkaus- ja poistopyyntöjen vastauksen jälkeen.

Ponnahdusikkunoissa käsiteltävien konfiguraatioiden, sääntöjen, laitesuodattimen ja työnkulun dataa säilytetään volume_main.jsx:n tilassa. Tilan tietoja käyttävät ja muokkaavat ponnahdusikkunoissa toimivat tilattomat komponentit, jotka ovat sidottuja volume_main.jsx:n tilaan JavaScriptin bind-funktiolla. Tällöin komponentit pystyvät suoraan lukemaan ja muokkaamaan volume_main.jsx:n tilassa olevia tietoja. Tilattomien komponenttien this-avainsana viittaa volume_main.jsx:n olioon, jolloin sidotut komponentit näkevät volume_main.jsx:n tilan ja pystyvät manipuloimaan sitä.

Emokomponenttinsa tilaan sidotut alikomponentit vaativat itselleen omistetun objektin emokomponentin tilaan, jotta alikomponentti osaa käsitellä tilaa oikein. Emokomponenttiin saadaan helposti luotua alikomponentin tarvitsema tilaobjekti tiedostosta, jossa on kaikkien eri alikomponettien vaatimat objektit. Tämä tapah-

tuu emokomponentin luonnin yhteydessä, jolloin emokomponentin tilaa alustetaan. Kunkin alikomponentin vaatimasta tilaobjektista tehdään kopio hyödyntäen JavaScriptin JSON.stringify()- ja JSON.parse()-funktioita, jotta kopioitu objekti on kokonaan uusi. Tämän jälkeen objekti voidaan ottaa käyttöön emokomponentin tilassa käyttämällä JavaScriptin Spread-syntaksia (…), joka yhdistää tilaobjektin emokomponentin tilaan. Tämän jälkeen alikomponentit osaavat käyttää emokomponentin tilaa ja ne voidaan ottaa käyttöön emokomponentissaan bind-funktioita käyttäen.

Komponenttien tilan sitominen bind-funktiolla ei ole tyypillisin tapa toteuttaa React-sovellus. React-kirjaston verkkosivujen esimerkeissä alikomponenteille annetaan tarvittava data komponenteille, mukaan lukien funktiot, joita alikomponentit käyttävät mahdollisesti käsitellessään emokomponentin tilaa (React n.d Handling Events).

Alikomponenttien tila kuitenkin päädyttiin toteuttamaan sitomalla alikomponenttien tila emokomponenttiin, koska volume_main.jsx tarvitsi keskeisen datavaraston, jotta se pystyi käyttämään tietoja ristiin. Sidottavilla alikomponenteilla oli tarve olla helposti tuotavissa muihin komponentteihin, koska suunnitteilla oli muita konfiguraatiosovelluksia, jotka tarvitsivat samoja alikomponentteja. Esimerkiksi Verifyn ja Diagnosticsin sääntökomponentit otettiin käyttöön toisessa alisovelluksessa opinnäytetyön aikana.

Toteuttamalla alikomponentit epätyypillisellä tavalla saavutettiin seuraavia hyötyjä:

- Alikomponentit ovat tuotavissa helposti muihin komponentteihin, jotka tarvitsevat alikomponentteja tilansa muokkaamiseen.
- Alikomponenttien logiikka pystytään rakentamaan helposti yhteen tiedostoon.
- Emokomponentti pystyy käsittelemään alikomponenttien tilaa helposti.

Alikomponenttien tilan toteuttaminen olisi ollut mahdollista myös eri keinoin. Toinen tapa olisi ollut Redux-tilanhallintakirjasto, jolla olisi voitu myös luoda koko

työnkulun hallintajärjestelmän laajuinen tietovarasto. Opinnäytetyöntekijä ei halunnut tuoda verkkosovelluksen toteuttamiseen kuitenkaan ylimääräisiä kirjastoja, joita ei käytetty muualla PMC:ssä. Toinen tapa olisi ollut luoda perinteinen React-komponentti, jolle olisi annettu käytetty data ja sen muokkaamiseen käytetyt funktiot sen parametreissä. Tällöin kuitenkin jokaisen komponentin, joka käyttää alikomponenttia, täytyisi toteuttaa ja antaa datan muokkaamiseen käytetyt funktiot itse. Mahdollista olisi ollut myös luoda kaikille alikomponentin tilanmuokkaamiseen käytetyille funktioille pysyvät tiedostot, jotka tuodaan emokomponentin tilaan sidottaviksi ja annettaviksi alikomponentille. Yksikertaisempaa kuitenkin oli sitoa koko alikomponentti omine datan muokkaamiseen käytettyine funktioineen emokomponenttiin useiden funktioiden sijaan.

Kolmas mahdollinen vaihtoehto sovelluksen toteuttamiseen sitomalla olisi ollut perinteinen tilallinen React-komponentti, joka olisi ottanut parametreissä vastaan vaaditun datan ja pystynyt kopioimaan sen omaan tilaansa, jota se olisi itse muokannut. Tietojen talletus emokomponenttiin olisi ollut haastavaa, koska emokomponentin olisi pitänyt käskeä alikomponenttiaan antamaan tietoja emokomponentille sekä emokomponentin olisi pitänyt pystyä lisäämään lennossa dataan lisää dataa muualta, kun muut alikomponentit saivat suoritettua tehtävänsä (Esimerkiksi laitesuodattimien lisäys Verify- ja Diagnostics-sääntökomponentteihin).

Helpoin tapa oli toteuttaa yhteinen tietovarasto volume_main.jsx-tiedostoon, jonka tilaa alikomponentit pystyivät muokkaamaan, mutta myös volume_main.jsx pystyi helposti käsittelemään ja siirtämään dataa tilaobjekteista toiseen.

Alikomponenttien tilan sitominen emokomponentteihin ei kuitenkaan ole ongelmatonta. Alikomponenttien tilan sitominen ei ole yleinen tapa toteuttaa React-sovelluksia, mikä saattaa aiheuttaa hämmennystä muiden lukiessa ohjelmaa. Sitominen myös aiheuttaa sen, että yhtä emokomponentin tilaa muokkaa monta eri funktioita monista eri tiedostoista, mikä aiheuttaa sovelluksen logiikkaan sekavuutta ja hajaantuneisuutta.

4.5 Verify-konfiguraatio ja -sääntö

4.5.1 Verify-konfiguraatio

Verify-konfiguraation ja sääntöjen luontiprosessi on kuvattu kappaleessa 3.4.2, josta selviää käyttöliittymän rakenne. Verify-konfiguraatio tehdään omassa ponnahdusikkunassaan, kuten kaikki muutkin konfiguraatiot ja säännöt. Verify-konfiguraatiolle on tehty oma tiedosto modal_verify_configuration.jsx. Konfiguraatio piirretään ponnahdusikkunakomponentin sisällä. Ponnahdusikkunakomponentti on tehty työnkulun hallintajärjestelmää varten.

Komponentin piirtämistä säätelee työnkulun hallintajärjestelmän pääkomponentti volume_main.jsx. modal_verify_configuration.jsx on yksinkertainen komponentti, jolla on vähän toiminnallisuutta. Konfiguraatio käsittelee volume_main.jsx:ssä olevaa verifyConfiguration-nimistä tilaobjektia, joka pitää sisällään konfiguraation nimen, taulukon kyseisen konfiguraation sääntöobjekteista sekä tiedon siitä, onko kyse konfiguraation editoimisesta vai uuden luomisesta.

Käyttäjän luodessa uutta konfiguraatiota, volume_main.jsx ensin alustaa verifyConfiguration-objektin, minkä jälkeen se piirtää komponentin. Konfiguraation nimeä säädellään tekstikentällä. Konfiguraation sääntöjen luonti ja muokkaaminen tapahtuu erillisen komponentin avulla.

Säännön luonti alkaa painamalla ”Create rule” -painiketta, joka pyytää volume_main.jsx:ää alustamaan sääntöä varten verifyRule-objektin tilaansa sekä piirtämään Verifyn sääntökomponentin. Sääntökomponentin tallennuksen jälkeen volume_main.jsx lisää uuden sääntöobjektin konfiguraation sääntötaulukkoon.

Käyttäjän halutessa muokata jo olemassa olevaa sääntöä Verifyn konfiguraatiokomponentti pyytää volume_main.jsx:ää kopioimaan omasta tilaobjektistaan säännön ja siirtämään sen verifyRule-tilaobjektiin volume_main.jsx:ssä käsiteltäväksi. Tällöin Verifyn sääntökomponentissa on muokattavan säännön tiedot. Käyttäjän tallennettua tiedot volume_main.jsx korvaa sääntötaulukossa olevan sääntöobjektin, jolla on sama identifikaattori kuin muokattavalla säännöllä. Käyt-

täjä voi halutessaan myös poistaa säännön konfiguraatiosta, jolloin sääntö poistetaan verifyConfiguration-tilaobjektin sääntötaulukosta. Kun käyttäjä tallettaa konfiguraation, volume_main.jsx lähettää Volume Workflown backendin rajapinnalle pyynnön talletuksesta. Muokatessa volume_main.jsx lähettää rajapinnalle muokkauspyynnön käyttäen verifyConfiguration-tilaobjektissa olevaa ID:tä. volume_main.jsx:n saadessa rajapinnalta vastauksen luonnin tai muokkaamisen onnistumisesta se sulkee Verifyn konfiguraatiokomponentin.

4.5.2 Verify-sääntö

modal_verify_rule.jsx vastaa Verifyn sääntöjen luonnista ja muokkauksesta. volume_main.jsx:n tilassa se käsittelee verifyRule-nimistä objektia, joka pitää sisällään seuraavat asiat:

- säännön nimen
- taulukon säännössä päällä olevista testitapauksien ID:ista
- taulukon testitapauksien ID:ista, jotka pysäyttävät Volumen suorituksen puhelimen osalta, jos testi antaa negatiivisen tuloksen
- säännön laitesuodattimet
- säännön id:n, jos sääntöä muokataan.

Verifyn säännöt muodostetaan Verifyn testitapausdatasta, jonka configurations_app.jsx hakee rajapinnalta. Testitapausdatan avulla luodaan käyttöliittymään lista Verifyn testitapauksista, joita modal_verify_rule.jsx käsittelee.

Käyttäjän painaessa testitapauksen päälle tai pois komponentti poistaa tai lisää verifyRule-tilaobjektissa olevaan testitapaustaulukkoon testin ID:n. Samanlainen prosessi tapahtuu käyttäjän painaessa Volumen suorituksen pysäyttävää negatiivisen testituloksen valintaruutua. modal_verify_rule.jsx käyttöliittymässä näkyvät painikkeet ”Select all” ja ”Stop all on failure” lisäävät kaikki testitapauksien ID:t tai poistavat ne verifyRulen taulukoista, riippuen siitä, ovatko kaikki testien valintaruudut ennestään valittuja.

Verify-säännön laitesuodattimien toiminnallisuus käsitellään kappaleessa 3.4.2.

Käyttäjän tallentaessa säännön volume_main.jsx muuntaa verifyRule-tilaobjektin sopivaan muotoon tietokannalle. verifyRule-tilaobjektin testitapausten ID:t yhdistetään yhdeksi JSON-objektiksi, jossa on tieto testin ID:stä ja totuusarvo pysäytetäänkö Volumen suorittaminen negatiivisesta testituloksen sattuessa. Tämän jälkeen objekti siirretään Verify-konfiguraation sääntöjen alle, kuten aikaisemmassa kappaleessa mainittiin. Käyttäjän halutessa muokata Verify-konfiguraation sääntöä sen data muunnetaan jälleen sopivaksi modal_verify_rule.jsx:lle, jossa testitapauksien ID:t ja pysäyttävien testien ID:t ovat erillisissä taulukoissa.

4.6 Diagnostics-konfiguraatio ja -säännöt

4.6.1 Diagnostics-konfiguraatio

Diagnostics-konfiguraatioista vastaa modal_diagnostics_configuration.jsx. Se toimii samalla tavalla kuin Verify-konfiguraatioon tarkoitettu komponentti. Sillä on oma diagnosticsConfiguration-tilaobjekti volume_main.jsx:ssä, joka pitää sisällään Diagnostics-konfiguraation nimen, säännöt sekä konfiguraation muokkaamispyynnössä käytetyn ID:n.

4.6.2 Diagnostics-sääntö

Diagnostics-sääntöjä hallinnoiva komponentti on modal_diagnostics_rule.jsx, jonka volume_main.jsx:n tilaobjekti on diagnosticsRule. Tilaobjekti pitää sisällään seuraavia tietoja:

- säännön nimen
- taulukon säännössä päällä olevista testitapauksien ID:ista
- taulukon testitapauksien ID:ista, jotka pysäyttävät Volumen suorituksen puhelimen osalta, jos testi antaa negatiivisen tuloksen
- taulukko testisettien ID:ista ja niiden järjestyksestä
- erikoistestitapauksien parametrit
- säännön laitesuodattimet.

Diagnostics-sääntökomponentti toimii samantapaisesti kuin Verifyn sääntökomponentti. Se pitää sisällään testitapauksien määrittämiseen käytetyt taulukot, joiden avulla Diagnostics-säännön testitapaukset toimivat. Diagnostics-kuitenkin eroaa siinä, että testitapaukset ovat nimellisesti testisettien alla käyttöliittymässä, kuten kuvassa 9 nähdään. Testitapaukset ovat eroteltuja testisettien alle, jotta käyttäjän on helpompi käsittää testien tarkoitus ja jaotella ne mielessään. Testitapauksien valinta tapahtuu samoin kuin Verify sääntökomponentin testitapauksien valinta.

Diagnostics-säännössä on mahdollista muuttaa testitapauksien suoritusjärjestystä, kuten kuvassa 10 näkyy. Tämä tapahtuu testisettien ID-taulukon avulla. Taulukossa pidetään kirjaa settien ID:sta, joiden alla olevia testitapauksia on valittuna. Käyttäjä voi muokata testisettien järjestystä raahaamalla kuvassa 10 näkyvien testisettien nimiä uuteen järjestykseen. Tämä toiminnallisuus on toteutettu HTML Drag and Drop API:n avulla. Käyttäjän aloittaessa raahauksen raahattu elementti otetaan muuttujaan ja html-sivuun lisätään selaimen document-muuttujan avulla näkymätön elementti. Tämä näkymätön elementti asetetaan raahauskuvaksi, joka näytetään käyttäjän kursorin vieressä. Tällöin saadaan paranneltua vaikutelmaa, että käyttäjä rahaa elementtiä listassa. Kun raahattu elementti viedään muun testisettilistassa olevan elementin päälle, se siirtää elementin HTML-puussa olevan elementin eteen tai jälkeen. Raahaustapahtuman loputtua lisätty näkymätön elementti poistetaan, testisettien järjestys tarkistetaan HTML-puussa isäntäelementissään ja testisettien järjestys muutetaan sen mukaisesti diagnosticsRule-tilaobjektissa. Safari-selaimen takia näkymätön elementti lisätään HTML-sivuun, koska Safari-selain vaatii, että raahauskuva on HTML-puussa, eikä raahauskuvaa voida väliaikaisesti luoda selaimen document-muuttujan avulla.

Käyttäjän tallentaessa sääntöä konfiguraatioon testitapaukset muunnetaan testisettien järjestyksen mukaisesti JSON-objekteiksi, joissa on tieto testin ID:sta ja siitä, että pysäytetäänkö Volumen suoritus testituloksen ollessa negatiivinen.

Diagnostics testisettien ja testien data haetaan configurations_app.jsx:n toimesta rajapinnalta ja annetaan volume_main.jsx:lle, joka edelleen välittää ne modal_diagnostics_rule.jsx:lle.

4.7 Laitesuodattimet

4.7.1 Laitesuodattimien lisäys

Verify- ja Diagnostics-säännöillä on taulukko laitesuodattimista, joiden avulla PiceaServices päättelee, suoritetaanko Verify- tai Diagnostics-operaatioita laitteelle. Laitesuodattimia ja niiden luontia kuvataan kappaleessa 3.4.2.

Ponnahdusikkunassa on alustoille, laitevalmistajille ja puhelinmalleille monivalintaiset pudotusvalikot. Pudotusvalikoiden valintojen muodostamiseen käytetty data saadaan configurations_app.jsx:ltä, joka hakee tiedon Piceasoft Oy:n laitetietokannasta. Laitedata on JSON-muotoista ja laitedata muodostaa puumallisen kuvauksen laitteista.

Laitesuodattimien ponnahdusikkunasta löytyy myös mahdollisuus lisätä maksimi- ja minimikäyttöjärjestelmäversio laitteille, joiden avulla pystytään rajaamaan laitteita pois.

Käyttäjän tallentaessa laitesuodattimia valitut kohteet muunnetaan eri muotoon. Pudotusvalikkojen valintojen ja minimi- ja maksikäyttöjärjestelmäversiosyötteiden avulla tallennettava data muunnetaan JSON-taulukoksi, joka sisältää JSON-objekteja. JSON-objektit ovat seuraavassa muodossa:

```
{
   fid: "", // Piceasoft Oy:n käyttämä alustan uniikki ID-arvo
   manufacturer: "", // Valmistajan nimi
   model: "", // Puhelimen malli
   os_version_min: 1.2, // Minimi käyttöjärjestelmäversio
   os_version_max: 2.4 // Maksimi käyttöjärjestelmäversio
}
```

Laitesuodattimia voidaan tehdä alusta-, valmistaja- ja puhelinmallitasolla. Jos käyttäjä tekee esimerkiksi Android-tason laitesuodattimen, kaikki Android-puhelimet voidaan ajaa säännön testeillä. Tällöin suodatinobjektin ainoa pakollinen

avain on alusta ja muita avaimia voidaan lisätä ja jättää pois riippuen siitä, kuinka tarkka yhdestä suodatinvalinnasta halutaan.

4.7.2 Laitesuodattimien tuominen CSV-tiedostosta

Lisäominaisuutena työnkulun hallintajärjestelmään lisättiin mahdollisuus tuoda laitesuodattimia CSV-tiedostoista sääntöön. Tuonti toteutettiin käyttäen selaimen File Reader API:a, joka mahdollistaa tiedostojen lukemisen selaimessa. Tiedoston lukeminen toteutettiin selaimessa, koska palvelimilta haluttiin säästää työkuormaa, joka voidaan toteuttaa käyttäjän selaimessa.

CSV-tiedostojen riviformaatissa päädyttiin seuraaviin muotoihin:
alusta, valmistaja, malli, minimi-, maksimikäyttöjärjestelmäversio
ja
alusta, valmistaja, malli

Arvot voidaan myös korvata tähdillä, jolloin arvo pätee kaikkiin mahdollisiin sen alla oleviin laitteisiin. Esimerkiksi

*Android, Samsung, **

suodatin pätee kaikkiin Samsung-puhelimiin.

Laitesuodattimien tuonnissa täytyi kiinnittää huomiota suorituskykyyn. Laitesuodattimien tuonnissa haluttiin varmistaa, ettei laitesuodatintaulukossa ole samoja laitteita kahta kertaa. Kaksoiskappaleiden poistoa varten tuli verrata laitesuodatinobjekteja keskenään, jotta mahdolliset kaksoiskappaleet pystyttiin poistamaan. Tätä varten etsittiin kolme eri tapaa, joilla pystyttiin poistamaan taulukosta kaksoiskappaleet ja vertailtiin niiden suoritusaikoja keskenään 10 000 laitesuodattimella, josta 489 oli uniikkeja.

Kaksoiskappaleiden poistamiseen käytetyt funktiot:

1. Merkkijonoksi muuttaminen: taulukon objektit muutetaan merkkijonoiksi, jonka jälkeen kutsutaan Set-funktiota käyttäen new-avainsanaa, jonka jälkeen merkkijonot taulukossa muutetaan takaisin objekteiksi. Set-funktio poistaa automaattisesti kaksoiskappaleet ja muuntaa tiedot Set-objektiksi. Tämän jälkeen tiedot voidaan iteroida ja muuntaa takaisin merkkijonoista objekteiksi.
2. Objektien avaimien vertailu: taulukon kaksoiskappaleet suodatetaan Array.prototype.filter-funktion ja Array.prototype.findIndex-funktion avulla käyttäen kahden objektin omistamien avaimien vertailuun käytettyä funktiota.
3. Objektien avaimien vertailu käyttäen Lodash-kirjastoa: taulukon objekteista poistetaan kaksoiskappaleet käyttäen Lodash-kirjaston uniqWith-funktiota ja isEqual-funktiota.

Vertailuun ja poistoon käytetyt funktiot ovat liitteessä 1. Vertailu tehtiin kutsumalla performance.now-funktiota ennen ja jälkeen käytettyä funktiota. Jokaista funktiota varten selain käynnistettiin uudelleen. Tämän jälkeen testi ajettiin viidesti, josta laskettiin keskiarvo.

Funktioiden suoritusajat:

1. Merkkijonoiksi muuttaminen: 12 ms
2. Objektien avaimien vertailu: 111 ms
3. Lodash-kirjaston objektien avaimien vertailu: 2670 ms

Merkkijonojen vertailu oli suoritusajaltaan nopein. Hypoteesina oli, että objektien avaimien vertailu olisi nopein. Merkkijonoiksi muuttamista epäiltiin ennen testejä hitaaksi, koska objektien merkkijonoksi ja takaisin muuttamista epäiltiin raskaaksi operaatioksi. Tämä osoittautui epätodeksi. Objektien avaimia vertailevien funktioiden aikaerot olivat suuria, vaikka niiden periaatteessa pitäisi olla samanlaiset.

Syy tähän on mahdollisesti se, että Lodash-kirjaston isEqual-funktio tarkistaa objektien avaimien arvot myös siltä varalta, että ne ovat objekteja tai taulukkoja. Vastaavasti objektien avaimien vertailuun käytetty funktio vertaa avaimien arvoja toisiinsa kiinnittämättä huomiota ovatko arvot mahdollisesti objekteja tai taulu-

koita, mikä saattaisi aiheuttaa vertailussa ongelmia. Tätä ongelmaa ei ole laitesuodattimien objektien vertailussa, koska vertailtavien objektien arvojen tiedetään olevan merkkijonoja. Merkkijonovertailussa on mahdollisena ongelmana se, että objektin avaimien ollessa väärässä järjestyksessä vertailu ei onnistu. Silloin merkkijonot eivät täsmää toisiinsa ja vertailu epäonnistuu. Avaimien eri järjestyksestä aiheutuvaa ongelmaa ei kuitenkaan ole, koska laitesuodatinobjektien muodostamisessa avaimien järjestys on taattu.

4.8 Erasure-konfiguraatio

Erasure-konfiguraatio on yksinkertainen verrattuna aikaisempiin konfiguraatioihin, koska kaikki Erasure-konfiguraatioon liittyvä tehdään yhdessä ponnahdusikkunassa. Konfiguraation luonti kuvataan kappaleessa 3.4.4.

volume_main.jsx:n tilassa olevaa tilaobjekti erasureConfiguration:ia hallitsee modal_erasure_configuration.jsx, joka on myös sidottu volume_main.jsx:n tilaan. Tilaobjekti poikkeaa muista, koska se muodostetaan dynaamisesti Erasuren pyyhintävaihtoehdoista, jotka se saa configuration_app.jsx:n avulla haetulta rajapinnalta. Tilaobjekteissa säilytetään seuraavia tietoja:

- Konfiguraation nimi
- Konfiguraation muokkaamisessa käytetty ID
- Konfiguraation säännöt eli dynaamisesti pyyhintävaihtoehdoista muodostettu objekteja sisältävä taulukko. Objekteihin on määritetty Piceasoft Oy:n alustoille määrittämä ID-arvo ja alustalle määritetty pyyhintämetodin ID-arvo.

modal_erasure_configuration.jsx muodostaa tilaobjektissa olevista säännöistä käyttöliittymän configuration_app.jsx:stä saatujen pyyhintävaihtoehtojen kanssa. iOS-erasoinnin lisäkysymykseen on luotu erityiskäsittely. iOS- ja Android-erasointisäännöt jaetaan käyttöliittymässä omiin paikkoihinsa erityiskäsittelyllä. Mahdolliset tulevaisuudessa lisättävät erasointisäännöt piirretään automaattisesti iOS- ja Android-erasointisääntöjen alle, jotta pyyhintävaihtoehtoja voidaan lisätä tulevaisuudessa ilman käyttöliittymään koskemista.

Ponnahdusikkunassa olevat pudotusvalikkojen vaihtoehdot haetaan configuration_app.jsx:ltä saaduista pyyhintävaihtoehtotiedoista. Käyttäjän valitessa pudotusvalikoista pyyhintämetodin ID asetetaan tilaobjektissa alustan pyyhintämetodiksi.

Erasure-konfiguraation tallentaminen ja muokkaaminen tapahtuu samalla tavalla kuin aikaisemmissa konfiguraatioissa.

4.9 Volume työnkulku

Työkulun luonti tapahtuu samantapaisesti kuin konfiguraatioiden (kuvattu kappaleessa 3.4.7). Ponnahdusikkunassa olevasta näkymästä vastaa modal_volume_workflow.jsx, joka on sidottu muiden tapaan volume_main.jsx:n tilaan ja muokkaa siellä tilaobjektiaan. Tilaobjekti on yksinkertainen verrattuna konfiguraatioiden ja sääntöjen tilaobjekteihin. volumeWorkflow-tilaobjektissa pitää sisällään seuraavat tiedot:

- Työnkulun nimi
- Taulukko tuotteiden ID:ista, joissa kyseistä työnkulkua käytetään
- Työnkulussa käytetyn Verify-konfiguraation ID
- Työnkulussa käytetyn Diagnostics-konfiguraation ID
- Työnkulussa käytetyn Erasure-konfiguraation ID
- Työnkulussa muokkaukseen käytetty ID

Käyttöliittymästä löytyvät konfiguraatioiden valintaan käytettyjen pudotusvalikkojen vaihtoehdot muodostetaan volume_main.jsx:n tilassa pidetyistä konfiguraatiotaulukosta. Taulukosta erotellaan konfiguraatiot perustuen niiden tyyppiin (Verify, Diagnostics, Erasure), minkä jälkeen niistä muodostetaan ID:iden ja nimien avulla pudotusvalikon vaihtoehdot. Pudotusvalikoissa on myös vaihtoehtona, ettei tiettyä operaatiotyyppiä ajeta, jolloin kyseisen operaation ID:ksi asetetaan null-arvo tilaobjektissa.

Tuotteiden monivalintapudotusvalikon vaihtoehdot muodostetaan PMC:n rajapinnalta saatujen asiakkaan tuotetietojen avulla käyttäen react-dropdown-select -kirjastoa.

Käyttäjän tallentaessa työnkulun se lähetetään työnkulun hallintajärjestelmän backendille, jonne se talletetaan. Tämän jälkeen PiceaServicesissä voidaan käyttää uutta työnkulkua.

5 POHDINTA

Volumen työnkulun hallintajärjestelmää voidaan pitää onnistuneena yksisivuisena verkkosovelluksena. Hallintajärjestelmästä ei ole löydetty kriittisiä vikoja ensimmäisen virallisen julkaisun jälkeen. Verkkosovelluksen kehitysvaiheessa vikoja löytyi, mutta se on osa ohjelmistotuotannon arkipäivää. Verkkosovelluksen toteutus pysyi aikataulussa muun projektin kanssa.

Asiakas, jonka vaatimusten pohjalta uutta työnkulun hallintajärjestelmää aloitettiin tekemään, on ottanut sen käyttöönsä onnistuneesti ja järjestelmä on saanut jopa hyvää palautetta helppokäyttöisyydestä. Myös toiselle asiakkaalle esiteltiin uutta järjestelmää sen kehitysvaiheessa, jolloin saatiin myös hyvää palautetta järjestelmästä. Tästä voidaan päätellä, että projekti täytti tavoitteensa luoda uusi verkkosovellus työnkulun hallintaan yksisivuisena sovelluksena, jonka pohjaan on lisätty myöhemmin muita vastaavia yksisivuisia verkkosovelluksia.

Yksisivuisena React-verkkosovelluksena hallintajärjestelmän toteuttaminen onnistui hyvin. Asiakkuudenhallintajärjestelmässä oli käytetty React-sovelluksia jo aikaisemmin, joten React-sovelluksen kehittämiselle asiakkuudenhallintajärjestelmään oli jo täysi tuki ennen projektin alkua. Muilla yksisivuisilla verkkosovelluskirjastoilla sovelluksen toteuttaminen ei olisi ollut viisasta. Asiakkuudenhallintajärjestelmään oli toteutettu jo aikaisemmin React-sovelluksia, joten asiakkuudenhallintajärjestelmän alisovellusten toteuttaminen usealla yksisivuisten verkkosovellusten kirjastolla olisi aiheuttanut sekavuutta asiakkuudenhallintajärjestelmän tulevaan hallinnointiin. Työnkulun hallintajärjestelmä olisi kuitenkin pystytty yhtä hyvin toteuttamaan millä muulla tahansa suosituista yksisivuisista kirjastoista. React oli kuitenkin paras vaihtoehto niin Piceasoft Oy:n kuin opinnäytetyöntekijän kannalta.

Tulevaisuudessa toteutettua työnkulun hallintajärjestelmää voitaisiin parannella vielä nykyisestään. Luvussa 4.4 kuvattu arkkitehtuuriratkaisu alikomponenttien tilan sitomisesta pääkomponentin tilaan voitaisiin vaihtaa. Ponnahdusikkunoiden komponentit voitaisiin vaihtaa tilallisiin ja komponenteille voitaisiin antaa pääkomponenttiin sidottu funktio, joka osaa asettaa annetut tiedot ponnahdusik-

kunoiden komponenteilta pääkomponentin tilaan. Ongelmaksi jää silti, että pääkomponentin tulee hallita tilatietoja jossain määrin, koska ponnahdusikkunakomponentit tarvitsevat tietoja toisiltaan. Tämä parantaisi työnkulun hallintajärjestelmän luettavuutta ja hallittavuutta, koska sovellus olisi toteutettu tyypillisesti React-kirjastolle.

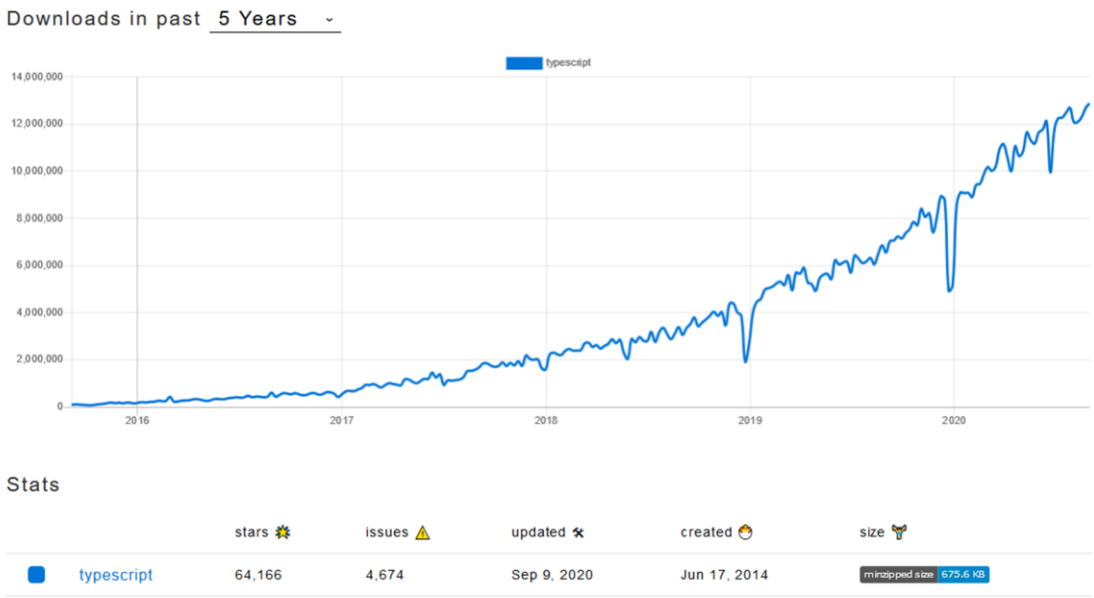
Myös React-kirjaston uudehko ominaisuus Hooks voisi olla mahdollinen toteutustapa alikomponenttien tilan sitomiselle. Tämä olisi vielä parempi vaihtoehto edellä komponenttien luokiksi vaihtamiseen, koska alikomponenttien tyyppiä ei tarvitsisi muuttaa tilallisiksi luokiksi (class), vaan tila pystyttäisiin luomaan Hooksien avulla olemassa olevaan komponentin funktioon. Tämä vähentäisi muutokseen kuluvaa työmäärää. Hooks-ominaisuutta ei ole kuitenkaan mahdollista käyttää tällä hetkellä, koska asiakkuudenhallintajärjestelmän React-versio on 16.7 ja Hooks-ominaisuus julkaistiin vasta versiossa 16.8.

Edellisen monisivuisena toteutetun sovelluksen laajentaminen sopimaan uusiin asiakasvaatimuksiin ei olisi ollut myöskään paras ratkaisu. Yksisivuisten verkkosovellusten tarjoamat edut monisivuisiin ovat huomattavat. Yksisivuisten verkkosovellusten myötä verkkosovelluksista on saatu yhä enemmän työpöytäsovellusten kaltaisia, mikä parantaa niiden käytettävyyttä. Tulevaisuudessa yksisivuisia verkkosovelluksia tullaan todennäköisesti suosimaan vielä enemmän niiden tarjoamien ominaisuuksien takia monimutkaisissa verkkosovelluksissa.

Verkkosovellusten kehittäminen siirtyy samaan aikaan myös pois perinteisistä verkkosovelluskehityksen menetelmistä, kun WWW-sisällönhallintaohjelmat, kuten WordPress, keräävät suosiota helppokäyttöisyydellään. WordPress:ille on kehitetty lukuisia kirjastoja laajentamaan sen toimintaa, mikä vähentää tarvetta perinteiselle verkkosovelluksen kehitykselle.

Perinteinen verkkosovelluskehitys saattaa myös muuttua tulevaisuudessa. Verkkosovelluksissa toiminnallisuus saadaan aikaan JavaScript-ohjelmointikielellä, mutta Microsoft on kehittänyt uuden TypeScript-kielen, joka voidaan koota JavaScriptiksi ja suorittaa selaimessa. TypeScript tarjoaa ominaisuuksia, joita JavaScriptissä ei ole mahdollista käyttää. Kielen suosio onkin kasvanut viimei-

sen viiden vuoden aikana (Kuvio 5). TypeScript tulee todennäköisesti nousemaan suosiossa yhä enemmän tulevaisuudessa. Etenkin, koska suositut yksisivuiset verkkosovelluskirjastot kuten React, Vue ja Angular tukevat TypeScriptin käyttöä.



KUVIO 5. Typescript-kirjaston suosio paketinhallintajärjestelmä NPM:ssä viimeisen 5 vuoden ajalta (npm trends 2020).

Tulevaisuudessa myös WebAssembly todennäköisesti vaikuttaa verkkosovellusten kehitykseen. WebAssembly mahdollistaa C-, C++-, C#- ja Rust-kielien kääntämisen verkkoselaimella sopivaksi. WebAssembly on matalan tason ohjelmointikieli, jota voidaan suorittaa käyttäjän selaimessa melkein natiivisuorituskyvyllä (MDN web docs 2020 WebAssembly). Tämä mahdollistaa suorituskykyä vaativien sovellusten toteuttamisen verkkosovelluksina. Hyvä esimerkki tästä on käyttöliittymien suunnittelutyökalu Figma, joka käyttää WebAssemblya (Wallace 2017). Työkalulla voidaan tehdä käyttöliittymä-, rautalanka- tai prototyyppimalleja verkkosovelluksessa ja samanaikaisesti mallia voi olla työstämässä usea ihminen (Figma n.d). WebAssemblyn avulla Figma pystyi pienentämään latausaikojaan kolminkertaisesti JavaScriptiin verrattuna (Wallace 2017).

Verkkosovelluksien toteuttamismahdollisuudet lisääntyvät tulevaisuudessa. On vain ajan kysymys, milloin näemme yhä monimutkaisempia verkkosovelluksia toteutettavan verkkoselaimeen.

LÄHTEET

Angular. n.d. Sivuston alaviittessä oleva ”powered by Google” -merkintä. Luettu 10.9.2020.
https://angular.io/

Bengtson, J. 2019. Library Web Development: Beyond Tips and Tricks. Chicago: American Library Association.

Carvalho, R. 2017. Single Page Applications: When and Why You Should Use Them. Julkaistu 29.11.2017. Luettu 7.8.2020. ¨
https://www.scalablepath.com/blog/single-page-applications/

Compare package download counts over time. n.d. Kuvakaappaus TypeScript-kirjaston suosiosta NPM-paketinhallintajärjestelmässä. npm trends 2020. Kaapattu 10.9.2020.
https://www.npmtrends.com/

Compare package download counts over time. n.d. Kuvakaappaus yksisivuisten verkkosovellusten suosiosta NPM-paketinhallintajärjestelmässä. npm trends 2020. Kaapattu 7.8.2020.
https://www.npmtrends.com/

Emmit, S. 2015. SPA Design and Architecture: Understanding single-page web applications. New York: Manning Publications.

Figma. n.d. Figman toiminnallisuuksia esittelevä video. Katsottu 15.9.2020.
https://www.figma.com/

jQuery. n.d. What is jQuery? Luettu 7.8.2020.
https://jquery.com/

MacPherson. 2019. Website vs Web App: What’s the Difference? Julkaistu 27.7.2019. Luettu 23.6.2020.
https://medium.com/@essentialdesign/website-vs-web-app-whats-the-difference-e499b18b60b4

MDN web docs. 2020. CSS basics. Päivitetty 19.6.2020. Luettu 7.8.2020.
https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/CSS_basics

MDN web docs. 2020. Introduction to the DOM. Päivitetty 26.1.2020. Luettu 7.8.2020.
https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

MDN web docs. 2020. JavaScript Introduction. Päivitetty 16.7.2020. Luettu 7.8.2020.
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction

MDN web docs. 2020. WebAssembly. Päivitetty 21.8.2020. Luettu 28.8.2020. https://developer.mozilla.org/en-US/docs/WebAssembly

Mikowski, M., Powell, J. 2013. Single Page Applications. New York: Manning Publications.

React. 2020. Sivuston alaviittessä oleva tekijänoikeus. Luettu 7.8.2020. https://reactjs.org/

React. n.d. Components and props. Luettu 14.6.2020. https://reactjs.org/docs/components-and-props.html

React. n.d. Handling Events. Luettu 1.7.2020. https://reactjs.org/docs/handling-events.html

Staff. 2019. Static vs Dynamic Website: What Is the Difference? Julkaistu 25.4.2019. Luettu 24.6.2019. https://wpamelia.com/static-vs-dynamic-website/

TEPA-termipankki. n.d. verkkosovellus. Luettu 8.6.2020. https://termipankki.fi/tepa/fi/haku/verkkosovellus

Wallace. 2017. WebAssembly cut Figma's load time by 3x. Luettu 10.9.2020. https://www.figma.com/blog/webassembly-cut-figmas-load-time-by-3x/

You. 2020. Evan You:n työkokemusluettelo. Luettu 7.8.2020. https://www.linkedin.com/in/evanyou/

LIITTEET

Liite 1. Suodattimien kaksoiskappaleiden poistoon käytetyt funktiot

```
1.  // 1. Duplikaattien poisto muuttamalla objektit merkkijonoiksi
2.  removeDuplicates(devices) {
3.      let duplicatesRemoved = new Set(devices.map(device => JSON.stringify(de-
    vice)));
4.      let removed = [];
5.      for(const deviceString of duplicatesRemoved) {
6.          removed.push(JSON.parse(deviceString));
7.      }
8.      return removed;
9.  }
10.
11. // 2. Duplikaattien poisto vertailemalla objektien avaimia
12. removeDuplicates(devices) {
13.     function areEqualShallow(a, b) {
14.         for(let key in a) {
15.             if(!(key in b) || a[key] !== b[key]) {
16.                 return false;
17.             }
18.         }
19.         for(let key in b) {
20.             if(!(key in a) || a[key] !== b[key]) {
21.                 return false;
22.             }
23.         }
24.         return true;
25.     }
26.
27.    return devices.filter((device, index, self) => {
28.            return index === self.findIndex((d) => {
29.                return areEqualShallow(d, device);
30.            })
31.        }
32.     )
33. }
34.
35. // 3. Duplikaattien poisto Lodash-kirjastoa käyttäen
36. removeDuplicates(devices) {
37.     return lodash.uniqWith(devices, lodash.isEqual);
38. }
```