MOBIILISOVELLSTEKNOLOGIAN VALINTA

Case Rossum Oy

LAB-AMMATTIKORKEAKOULU
Tieto- ja viestintätekniikan insinööri (AMK)
Mediatekniikka
Syksy 2020
Nico Peltola

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Peltola Nico	Opinnäytetyö, AMK	Syksy 2020
	29	

Työn nimi

Mobiilisovellusteknologian valinta
Case Rossum Oy

Tutkinto

Tieto- ja viestintätekniikka (AMK)

Tiivistelmä

Opinnäytetyössä tavoitteena oli tutkia, kuinka Rossum Oy:n asiakkaiden pyytämiä ominaisuuksia, kuten toimiminen verkottomassa tilassa sekä sovelluksen asennettavuutta kotivalikkoon kannattaa toteuttaa. Opinnäytetyössä käsitellään natiivi-, hybridi-, verkko- sekä PWA-sovelluksien teoriaa, joiden pohjalta vertailu Rossum Oy käyttöön soveltuvasta teknologiasta toteutettiin.

Vertailussa huomattiin PWA-sovelluksen olevan paras alusta tämän projektin kannalta, joten opinnäytetyössä käsitellään tarkemmin PWA-sovelluskehitystä. Opinnäytetyössä käydään läpi mitä ominaisuuksia PWA-sovellukselta vaaditaan sekä miten niitä toteutetaan. PWA-sovelluksen teknologioita valittaessa otettiin käyttöön suosittu React JavaScript kirjasto. Tämä valinta pohjautui pieneen tutkimukseen suosituista PWA-teknologioista, joissa React näytti olevan usein listojen kärjessä. Reactilla rakennettujen PWA-sovelluksien tietokantana käytettiin useissa esimerkeissä Googlen Firebasea, joka todettiin sopivaksi myös tähän projektiin.

Asetetut tavoitteet toteutettiin onnistuneesti ja jatkokehitystä voidaan tehdä Rossum Oy:n toimesta.

Asiasanat

Sovelluskehitysalusta, Natiivisovellus, Hybridisovellus, Verkkosovellus, Progressiivinen verkkosovellus, React, Firebase, Node

Abstract

Author(s)	Type of publication	Published
Peltola Nico	Bachelor's thesis	Autumn 2020
	Number of pages	
	29	

Title of publication

Title
Case Rossum Oy

Name of Degree

Information and Communications Technology

Abstract

The aim of the thesis was to study how the features requested by Rossum Oy's customers, such as operating in offline mode and the installability of the application in the home menu, should be implemented. The thesis deals with the theory of native, hybrid, network, and PWA applications, based on which comparison of the technology suitable for use by Rossum Oy was made.

In comparison, it was found that the PWA application is the best platform for this project, so the thesis deals with PWA application development in more detail. The thesis reviews what features are required of a PWA application and how they are implemented. The popular React JavaScript library was introduced when selecting PWA application technologies. This selection was based on a small study of popular PWA technologies where React often appeared to be at the top of the lists. The database of PWA applications built with React used Google Firebase in several examples, which was also found to be suitable for this project.

The set goals were successfully implemented and further development can be done by Rossum Oy.

Keywords

Application development platform, Native application, Hybrid application, Web application, Progressive web application, React, Firebase, Node

SISÄLLYS

1 JOHDANTO

Yrityksen, jonka tuotteissa mobiilisovellukset ovat keskeisessä osassa, on tärkeää kartoittaa, tarvitseeko kehitettävä sovellus mobiililaitteiden, puhelimien ja padien sisäänrakennettuja natiiveja ominaisuuksia kuten paikkatietoja ja kameraa. Jos näille ominaisuuksille on tarvetta, ei kehittäjillä ole ollut muuta vaihtoehtoa kuin kehittää sovellus natiivisovellukseksi.

Natiivisovellusten kehittämisessä on ollut suuria ongelmia niiden alustasidonnaisuuksien takia. Tämä johtuu suuresti Androidin ja iOS:n käyttöjärjestelmien ja niille kehitettävien sovellusten alustojen suurista eroavaisuuksista. Näitä sovelluksia voitiin kehittää vain alusta kerrallaan ja pahimmassa tapauksessa yhdelle sovellusidealle jouduttiin palkkaamaan useita eri kehitysryhmiä. Monien kehitysryhmien kanssa toiminen tuo lisää uusia ongelmia. Miten eri alustoilla toimivien sovellusten yleinen ulkoasu, sekä tuntuma saadaan samanlaiseksi? Miten sovellusten uudet ominaisuudet saadaan päivitettyä samaan aikaan? Näiden ongelmien takia yritykset ovat joutuneet tekemään tutkimusta siitä, onko heidän edes kannattavaa tehdä sovelluskehitystä kaikille alustoille. Näin ollen jättäen mahdollisia asiakkaita pois kehitysrajoitteiden takia.

Sovelluskehityksen edistyttyä ei ole enää välttämätöntä tehdä isoja panostuksia alustariippuvaan kehitykseen. Uusien teknologioiden avulla voidaan kehittää sovellusta yhdellä alustalla ja sen jälkeen julkaista tämä usealle. Yritys säästää tarvittavan kehittäjämäärän vähenemisellä, eikä sille ole enää välttämätöntä tutkia, miltä alustalta saadaan maksimaalinen yleisö. Koska sähköisellä alalla tuotteen hinta vaikuttaa suuresti, säästetty raha kehitystyössä voidaan siirtää esimerkiksi yhtiön kehittämiseen. Näin saadaan yritykselle lisää kilpailukykyä. Yritys voi myös löytää uusia tapoja vahvistaa omaa myyntiään. Ketterä kehitys ja sovelluksen yhtenäinen ilme nostavat yrityksen brändin arvoa. Näillä tekijöillä voidaan saada houkuteltua lisää kohdeasiakkailla ja vahvistaa myyntiä.

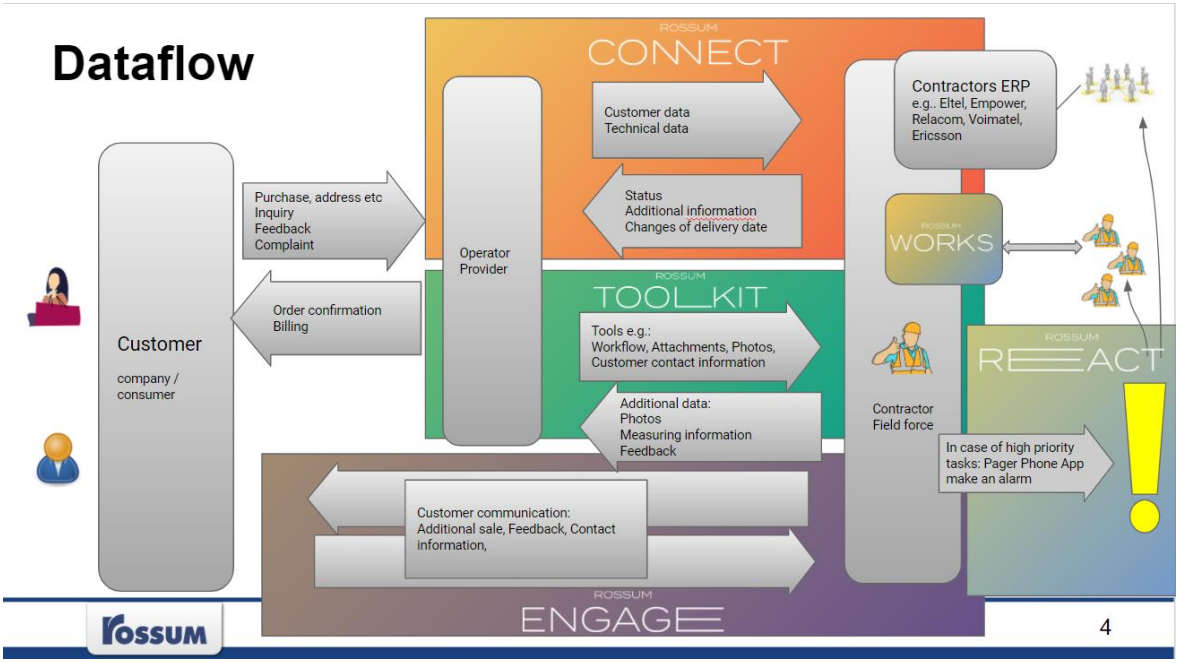
Insinöörityön tarkoituksena on tutkia sovelluskehitystä eri alustoille ja tehdä arvio teknologian soveltumisesta Rossum Oy:n käyttöön. Pyrkimyksenä on myös toteuttaa laaja katsaus eri teknologioihin, ja sen pohjalta valita yksi sovelluskehys pilottisovellukseen. Pilottisovelluksen tarkoitus on kattaa Rossum Oy:n yleisimmät käyttötarpeet ja toimia pohjana teknologian soveltuvuudesta lopulliselle arviolle.

2 TOIMEKSIANTAJA

Rossum Oy on vuonna 1994 perustettu suomalainen ohjelmistoyritys, jonka päätoimisto sijaitsee Lahdessa. Työtiimiin kuuluu kokonaisuudessaan 10 henkilöä ja yrityksen liikevaihto oli vuonna 2019 1,2 miljoonaa euroa. Rossum Oy toteuttaa työkaluja asennus- ja huoltotiimien, sekä toimitusketjun hallintaan. Yrityksellä on pitkä kokemus asennus- ja viankorjaustöiden käsittelystä erityisesti telecom-sektorilla.

Rossum tuoteperhe yhdistää tietoja ihmisten kanssa toimitusketjun eri vaiheissa. Rossumin palveluilla yhdistetään operaattorit ja urakoitsijat, työtilaukset, tilauksen tiedot, henkilöt ja järjestelmät. Seuraavassa kuvassa (Kuva 1) on esitetty Rossumin tuoteperhettä.

Tämä kaikki on tuotu yhteen järjestelmään. Eri osapuolet on integroitu toisiinsa ja näin jokainen saa työssään tarvitsemansa informaation.



Kuva 1. Rossum dataflow

- Connect

Rossum Connect käsittelee annetut tilaukset ja antaa ne tehtäväksi eteenpäin urakoitsijalle. Työtilaukset sisältävät kaiken tarpeellisen tiedon työn sisällöstä, suorituspaikasta, asiakkaasta, sekä lisäksi oikeat toimintatavat ja toimitusprosessit. Kun työ on kerran

lähetetty urakoitsijalle, niin Rossum Connect seuraa työn etenemistä ja tuottaa tarvittaessa ilmoituksen, kun työ tulee valmiiksi.

- Engage

Rossum Engage on joukko yrityskuvaa parantavia työkaluja. Niiden avulla voit varmistaa, että urakoitsijan kenttähenkilöt tukevat ja edustavat oikein yrityksen brändiä. Mobiilisti toimivalla käyttöliittymällä Rossum Engage auttaa arvioimaan asiakastyytyväisyyttä, tarjoamaan lisäpalveluita ja tunnistamaan yrityksen uusia myyntimahdollisuuksia. Käyttäjä voi viedä asennustapahtumaan toiminnot palaute- ja tyytyväisyyskyselyistä yksityiskohtaisiin myyntitarjouksiin ja lisäpalvelujen tarjoamiseen. Oikeilla tiedoilla ja oikeilla työkaluilla tulokset nähdään reaaliaikaisesti ja asiakaspalvelutyökin onnistuu tehokkaasti.

- React

Rossum Reactin avulla asiakkaiden tiketit jaetaan kenttähenkilöille automaattisesti. Tiketin saapuessa Rossum React laskee automaattisesti kyseisen asiakkaan kanssa sovitun ratkaisuajan. Rossum React hälyttää vapaana olevan kenttähenkilön ja lähettää työmääräimen. Tämän jälkeen palvelu seuraa työn edistymistä ja tarvittaessa lähettää muistutuksia ja varmistaa, että tiketti ratkaistaan ajoissa.

- Toolkit

Rossum Toolkit tarjoaa urakoitsijoiden ja alihankkijoiden asentajille asennuksessa tarvittavat työkalut, kuten aikataulut, tarkistusluettelot, todentamiset ja valtuutukset, kaikki yhdessä paikassa. Tämä helpottaa tekemään laadukasta työtä. Missä he ovatkin – toimistossa, matkalla tai paikan päällä Rossum Toolkit varmistaa, että käytössä on oikeat tiedot ja oikeat työkalut, jotta työ voidaan tehdä tehokkaasti. Rossum Toolkit toimii nykyisten kenttätöiden hallintajärjestelmien rinnalla ja täydentää niitä.

- Works

Rossum Works on liikkuvien työntekijöiden kenttätyön ohjaukseen suunniteltu toiminnanohjauspalvelu. Palvelun avulla voi jakaa työt tekijöille ja seurata reaaliaikaisesti töiden etenemistä ja tilatietoja. Rossum Works kerää automaattisesti aineistot laskutusta ja palkanmaksua varten. Ohjelmisto on helppo integroida jo olemassa oleviin järjestelmiin ja myös asiakkaiden tilausrajapintoihin osaksi toimitusketjua.

Kuten edellä olevasta tuoteperheen esittelystä voi päätellä, Rossum’in tuotteiden keskiössä ovat sovellukset, joita käytetään mobiili- ja älylaitteilla. Asiakkaiden toiminta on usein kiireistä, jolloin toimintavarmuus ja helppokäyttöisyys ovat tärkeitä tekijöitä hyvän

käyttäjäkokemuksen aikaansaamiseksi. Seuraavassa on esitetty Rossum'in sovellusten keskeisiä ominaisuuksia.

- Offline cache

Asiakkaana olevat työntekijät ottavat kuvia puhelimellaan maan alla, jossa ei internet-yhteyttä välttämättä ole tai se on heikko. Sovelluksen pitää siis toimia tilanteissa, joissa yhteys internettiin katkeaa kesken käytön.

- Koti-iconi laiteen työpöydällä tai kotivalikossa

Osan asiakkaista toiveena on ollut sovelluksen saatavuus puhelimen tai tietokoneen työpöytäsovellukseksi. Toiveiden perusteella joillekin asiakkaille on luonnollisempaa ja helpompaa käyttää yrityksen tarjoamaa sovellusta käyttäen työpöydältä löytyvää ikonia web-URL sijaan.

3 SOVELLUSTYYPIT

3.1 Natiivisovellukset

Natiivisovellus on sovellus, joka on rakennettu käyttämällä tietyn laitteen tai käyttöjärjestelmän kehittämiseen tarkoitettuja kieliä ja rajapintoja. Sovelluksen on oltava rakennettu vastaamaan kunkin käyttöjärjestelmän vaatimuksia, eivätkä ne toimi muissa käyttöjärjestelmissä. Tällaisia sovelluksia voivat olla esimerkiksi mobiilisovellukset, älykellot, sekä älytelevisiot. Perinteisesti juuri mobiilisovelluksien kehityksessä on käytetty natiiveja sovelluksia.

Natiivisovellukset tarjoavat käyttäjälle parhaan mahdollisen käyttökokemuksen, koska ne noudattavat alustan käyttöliittymän asettamia käytäntöjä. Ne ovat lisäksi nopeita sekä luotettavia. Natiivisovelluksilla on helpoin pääsy käyttöjärjestelmän kaikkiin ominaisuuksiin, kuten kameraan, biometriseen tunnistautumiseen, kalenteriin jne. Yleisesti natiiveille alustoille kehittäminen on kalliimpaa, mutta ne tarjoavat myös eniten ominaisuuksia. Natiivisovellusten kehittämisen kalleus johtuu siitä, että eri alustoille joudutaan kehittämään omat sovellukset. Toiselle alustalle jo tehdystä kehitystyöstä ei ole varsinaista hyötyä, koska alustat ovat niin erilaiset. Koska natiivisovellukset ovat aina alustakohtaisia, suuremman asiakaskannan saamiseksi joudutaan kehittämään sama sovellus moneen kertaan eri alustoille. Tämä tarkoittaa sitä, että sovelluksia kehitettäessä yrityksen täytyy palkata kehittäjätiimii, joka osaa useita eri natiivien alustojen ohjelmointikieliä, palkata eri alustoille eri ohjelmoijat, tai ulkoistaa sovelluskehitys kokonaan muualle. (Kambda 2020.)

Toisin kuin verkkosovellukset, natiivisovellukset eivät käytä yleensä alustanaan selainta. Ne on ladattava alustakohtaisilta sovelluskaupoilta, kuten Applen “App Store” ja Googlen “Google Play”. Sovellukset tallentuvat laitteen omaan muistiin, josta niitä voidaan käyttää painamalla vastaavaa kuvaketta laitteen näytöllä. Alustakohtaisten sovelluskauppojen huonoja puolia on niille julkaistujen sovellusten tarkastusajat. Apple sekä Google omistavat oikeudet pitää sovelluksen julkistamista tai päivityksen julkaisua jäässä jopa kaksi viikkoa. (Google 2020)

3.1.1 iOS

iOS on Applen kehittämä käyttöjärjestelmä, joka löytyy Applen useista tuotteista, kuten iPhonesta ja iPodista. iPad-mallisto kuului aiemmin myös laitteisiin, joista tämä kyseinen käyttöjärjestelmä löytyi, mutta Apple vaihtoi syyskuussa 2019 omaan iPadOS-käyttöjärjestelmään. (Makeuseof 2020)

Laitteille, joiden alustana toimii iOS, kehitetään käyttämällä integroitua kehitysympäristöä nimeltä Xcode. Xcodella kehittämiseen käyttäjä tarvitsee itselleen Mac tuotteen, oli se sitten MacBook, iMac tai vastaava, sillä Xcode:n on tarkoitettu käytettävän ainoastaan Mac OS alustalla. On mahdollista saada tämä työympäristö toimimaan PC:llä käyttäen epävirallisia alustoja, mutta tämä ei ole suositeltavaa. (Apple Developer Documentation.)

Xcode:n pystyy kuka tahansa Mac OS - laitteen omistava lataamaan Applen App storesta ilmaiseksi. Xcode tarjoaa kehittäjälle erilaisia valmiita sapluunoita ja projekteja, minkä avulla voi kehittäjä testata helposti haluamiaan ominaisuuksia ja suunnitella sovelluksen ulkoasua sekä käytettävyyttä. Sapluunat sisältävät valmiin konfiguraation ja tarvittavat tiedostot sovelluksen rakentamisen aloittamiseksi ja kehityksen helpottamiseksi. (Apple Developer Documentation.)

IOS kehityksen heikkouksia on sen käyttötestaaminen. Hiiren simuloiminen kosketusnäytöllä ja puhelimen ominaisuuksien (kamera jne.) testaaminen on hankalaa. Tarkkaan iPhone ja iPad testaamiseen vaaditaan aito laite, mutta valitettavasti niiden hinnat ovat korkeita. (Digitaltrends 2020)

3.1.2 Android

Android on Linux kernel-pohjainen, avoimen lähdekoodin käyttöjärjestelmä, jota kehittää useat kehittäjät (pääasiassa Google). Android on suurimmaksi osaksi suunniteltu kosketusnäyttöisten puhelimien ja tablettien käyttöjärjestelmäksi. Tosin tätä käyttöjärjestelmää on kehitetty myös useille muille alustoille kuten pelikonsoleille, autoille, sekä digitaalisille kameroille. Myös tästä käyttöjärjestelmästä on ”jalostettu” toisia käyttöjärjestelmiä. Amazon on tehnyt omalle Fire-brändille oman FireOS:n, jota myös puheohjauksella toimiva kaiutin Echo käyttää. (Android Developer Documentation 2020.)

Toisin kuin iOS kehityksessä, Androidia voidaan kehittää käyttäen Windows:ia, MacOS:ää tai Linux:ia. Tämä helpottaa Android kehitystä ja tekee siitä joustavampaa. Kehittäjät voivat luoda Android Virtual Device (AVD)-laitteita, jotka toimivat Android emulaattorilla. Nämä AVD-laitteet mahdollistavat testaamisen, vaikka kehittäjä ei fyysistä Android-laitetta omistaisikaan. (Android Developer Documentation 2020.)

Android Studio kääntää kirjoitetun koodin yhteen APK (Android Package) tiedostoon, joka sisältää kaiken tarvittavan sisällön sovelluksen suorittamiseen. Android-pohjaiset alustat käyttävät näitä APK-paketteja sovellusten asentamiseen ja ne tunnistetaan .apk jälkiliitteestä. (Android Developer Documentation 2020.)

3.2 Verkkosovellukset

Verkkosovellukset ovat sovelluksia, joiden suorittamiseen käyttäjä tarvitsee vain laitteen selaimen. Sovellus ei vaadi erillistä lataamista ja ne ovat yleensä kehitetty responsiivisuus edellä. Tämä mahdollistaa käyttäjälle helpon ja esteettömän sovelluksen käyttöönoton. (Careerfoundry 2018.)

Koska web-sovellukset suoritetaan selaimesta verkon yli, niiden toimivuus lakkaa tai heikkenee suuresti internet-yhteyden lakattua tai sen puuttuessa kokonaan. Niillä on myös rajoitettu pääsy laitteen natiiveihin ominaisuuksiin. Joten on tärkeää ennen kehitysvaihetta tutkia, mitä ominaisuuksia sovellus vaatii. Web-sovellusten luominen on halpaa verrattuna muihin sovellusratkaisuihin. Mutta koska niiden tulee toimia useiden eri selainten, sekä laitteiden kanssa, niiden suunnitteluun ja ylläpitoon menee paljon resursseja.

Web-sovelluksia kehitettäessä törmätäänkin usein samoihin ongelmiin. Miten ne toimivat eri selaimissa. Vaikkakin verkkoselaimet ovat hyvin samanlaisia, niissä myös on pieniä eroja, joihin täytyy kiinnittää huomiota. Näiden pienien eroavaisuuksien, sekä mobiiliselainten vaihtelevan HTML- ja CSS-tuen takia on kehitetty sovelluskehyksiä ratkaisemaan näitä yhteensopivuusongelmia. Niiden tarkoituksena on toimia mahdollisimman hyvin yhteen useiden eri mobiilialustojen kanssa. Yleisesti nämä sovelluskehykset ovat tehokkaita, mikä mahdollistaa sulavan käyttökokemuksen asiakkaalle.

3.3 Hybridisovellukset

Hybridisovellukset ovat ohjelmistosovelluksia, jotka toimivat useiden eri alustojen kanssa. Hybridisovellus on yhdistelmä natiivisovellusten ja web-sovelluksen väliltä. Niiden vahvuus tulee yhdelle alustalle kirjoitetusta koodipohjasta, josta sovellus käännetään sitten eri alustoille. Yleisesti tämä koodipohja tehdään käyttäen JavaScript-, HTML- ja CSS-teknologioita. Koska yhtä koodipohjaa käytetään sovelluksen kehittämiseen, voidaan tehdä nopeampaa sekä kustannustehokkaampaa sovelluskehitystä. (Ionicframework 2020.)

Hybridisovellukset ovat web-sovelluksia pohjimmiltaan, jotka ovat upotettu natiivisovelluksen sisään. Tämä upotettu web-sovellus pystyy saamaan sovelluslaajennuksien avulla ominaisuudet, jotka mobiilialusta tarjoaa sovellukseen upotetun selaimen kautta. Tallaisia ominaisuuksia ovat esimerkiksi biometrinen tunnistautuminen, kamera, sekä yhteystiedot. (Ionicframework 2020.)

Hybridisovelluskehyksiä on useita erilaisia ja niiden tavat toteuttaa kehitystä voivat erota hyvinkin paljon. Esimerkiksi sovelluskehykset kuten, Ionic, sekä Flutter tarjoavat käyttäjälle valmiit UI-kirjastot komponentteihin. Nämä sovelluskehyksien valmiit komponentit

antavat käyttäjälle aidonoloisen käyttäjäkokemuksen. On myös olemassa hybridisovelluskehyksiä, jotka käyttävät laitteen omia komponentteja hyväkseen. Näillä resursseilla saadaan aikaan sovelluksia, jotka vastaavat natiivisovellusten käyttöliittymää kaikissa käyttöjärjestelmissä. (Ionicframework 2020.)

3.4 Progressiiviset web sovellukset

Progressiiviset verkkosovellukset ovat yllä mainituista teknologioista tuorein ja useat yritykset kuten Google, Apple, Microsoft, sekä Mozilla ovat tekemässä siitä uutta verkkosovellus standardia (Howtogeek 2020). Progressiiviset verkkosovellukset kehitetään verkkosovelluksiksi, mutta ne käyttäytyvät kuin natiivit sovellukset. Näiden sovellusten “progressiivisuus” tulee käyttäjän tarpeen tyydyttämisestä. Jos käyttäjä tarvitsee tai haluaa käyttää vain verkkosovelluksen tarjoamia ominaisuuksia, hänellä ei ole tarvetta ladata laitteelleen ylimääräistä sovellusta. Jos taas käyttäjä tuntee tarvitsevansa lisää ominaisuuksia, joita sovellus tarjoaa tai pitää sovelluksen käynnistämistä kotivalikosta miellyttävämpänä kokemuksena, on hänellä mahdollisuus “edetä” sovelluksen lataamiseen. (Web dev 2020.)

Progressiivisten sovelluksien etuna on myös niiden riippumattomuus mobiililaitteiden julkaisualustoista. Kehittäjien ei tarvitse hyväksyttää omia sovelluksiaan Googlen tai Applen sovelluskaupoissa aina kun he haluavat päivittää tai julkaista oman sovelluksensa. Sovellukset päivittyvät silloin kun kehittäjät tahtovat ja turhia välikäsiä jää pois. Lisäksi ladattaessa progressiivisen verkkosovelluksen käyttäjälle aukeaa uusia ominaisuuksia, mitä ei välttämättä voida verkkosovelluksessa käyttää. Sovellus latautuu nopeammin ja tarjoaa käyttömahdollisuudet silloin kun internet yhteyttä ei ole saatavilla. Lisäksi sovellus voi tarjota push-ilmoituksia käyttäjälleen oleellisten tai akuuttien tapahtumien sattuessa. (Simicart 2020.)

Kuten nykyiset verkkosovellukset, niiden tarvittavat tiedot tulevat suoraan palvelimelta, mutta käyttäjä saa natiivisovelluksille tutun pikakuvakkeen.

3.5 Alustan valinta

Näiden haluttujen ominaisuuksien pohjalta toteutetaan vertailu, jonka perusteella valitaan pilottisovellukselle sopivin alusta. Alustoja edustavat natiivi-, hybridi-, verkko-, sekä progressiiviset verkko sovellukset. Taulukossa painotetaan Rossumin mielenkiinnon kohteina olevia ominaisuuksia, mutta taulukossa otetaan myös huomioon sovellusalustalle yleisesti oleellisia ominaisuuksia.

Vertailu sovellusalustan teknologioista riippuu monista tekijöistä: sovelluskehityksen kustannus, olennaisten ominaisuuksien toteuttamisen mahdollisuus, sekä käytetyt laitteet. Tässä taulukossa on otettu huomioon seuraavat asiat.

- Offline caching

Offline caching ei varsinaisesti ole alustakohtainen ongelma. Offline caching tarjoaa sovelluksen käyttäjälle paljon mukavamman käyttäjäkokemuksen. Internet yhteyden ollessa heikko tai sen puuttuessa sovelluksen toimivuus voi hidastua tai lakata kokonaan. Offline caching:in avulla voidaan ladata sovelluksen muistiin sovelluksen osia, jotka mahdollistavat sovelluksen toimivuuden verkottomassa tilassa. (Winningwp 2020.)

Offline caching voidaan toteuttaa kaikilla verratuilla alustateknologiolla. Natiivissa, sekä hybridissä ympäristössä voidaan käyttää laitteen omaa välimuistia datan caching. Erilaiset verkkosovellus-alustat tarjoavat valmiita komponentteja myös tämän toteuttamiseen. Esimerkiksi Ionic sovelluskehys toteuttaa ionic/storagen, sekä ionic-cache-observable pakettien avulla offline toiminnot. PWA-sovellukset taas käyttävät service workeriä Offline ominaisuuksien toteuttamiseen.

- Koti-ikoni

Koti-ikoni on käyttäjäystävällinen tapa käynnistää sovellus. Käyttäjä löytää haluamansa sovelluksen aina samasta paikasta, eikä linkkejä tarvitse muistaa ulkoa. Jos sovellus on esimerkiksi jokapäiväisessä käytössä, säästää käyttäjä aikaa huomattavasti painamalla yhtä nappia, selaimen avaamisen ja URL:in kirjoittamisen sijaan.

Koti-ikoni voidaan toteuttaa kaikilla muilla alustoilla paitsi puhtaalla verkkosovellus-alustalla. Sovelluskaupat vaativat koti-ikonin löytymistä natiivi-, sekä hybridisovelluksista, jotta he suostuvat sovelluksen listaamisen heidän alustoilleen.

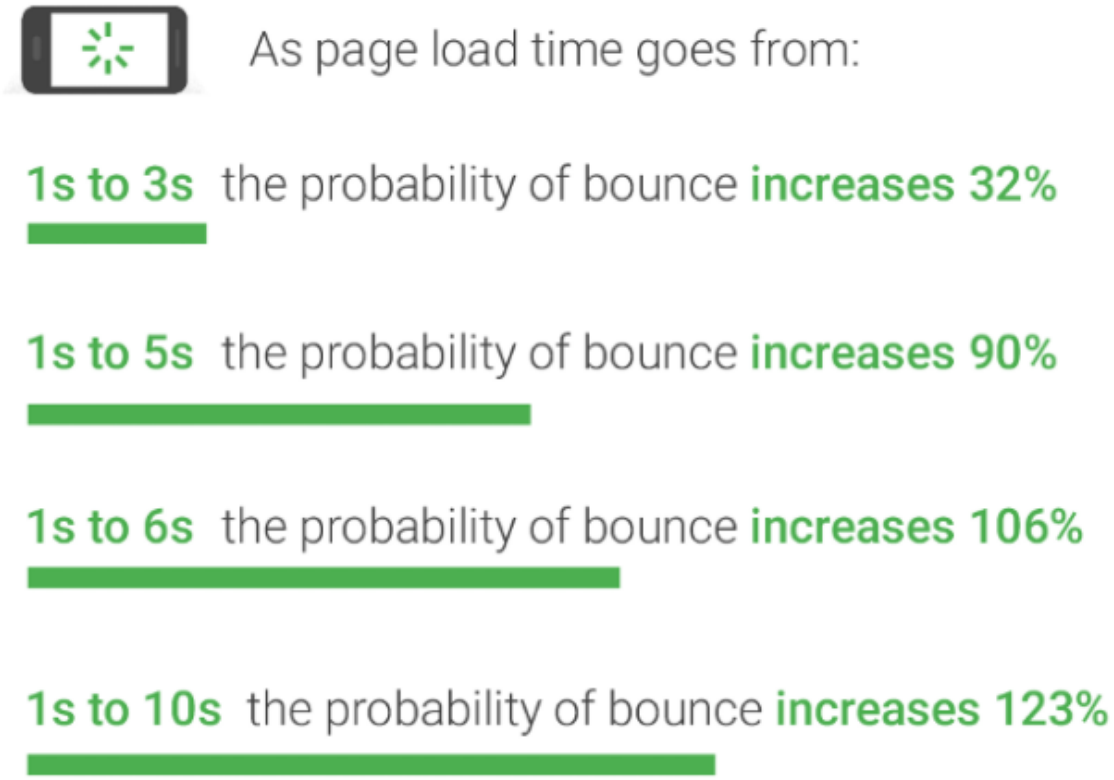
- Näkyvyys

Näkyvyydellä tarkoitetaan sovelluksen löytymistä eri alustoilta. Verkko-, sekä PWA-sovelluksilla on selvä etu muihin alustoihin siinä, että nämä löytyvät hakukoneilla etsiessä. Tämä antaa yrittäjälle useita eri vaihtoehtoja oman sovelluksen mainostamiseen, kuten maksamalla Googlelle oman sovelluksen näkyvyydestä tai optimoimalla omia hakutermejä.

- Suorituskyky

Suorituskyvyllä tarkoitetaan sovelluksen yleistä nopeutta, sekä ”järeyttä”. Käyttäjälle tulee negatiivinen käyttökokemus, jos napin painamisen jälkeen on suuri viive käyttäjän

olettaman tapahtuman välillä. Jo muutaman sekunnin viive nostaa käyttäjän halua lopettaa sovelluksen käyttö saman tien. Ja jos käyttäjä joutuu käyttämään sovellusta esim. työn puolesta, huono käyttäjäkokemus voi vaikuttaa hänen työpanokseensa. (Uxplanet 2019.)



Kuva 1. Rossum dataflowtodennäköisyys lähteä verrattuna viiveeseen

Sovelluksen järeyttä taas voidaan tarvita pelien tai lisätyn todellisuuden toteuttamiseen laitteella. Tämän kaltaisia sovelluksia tehdäänkin melkein puhtaasti natiiveina sovelluksina. Todellisuudessa tavallisten sovellusten suorituskyvyssä on vain marginaalisia eroja, eikä tämän pilottisovelluksen kannalta ole suorituskyvyn kannalta suuria vaatimuksia.

- Tekemisen tehokkuus

Tiedon uudelleenkäytöllä tarkoitetaan jo ennalta tunnetun tiedon, sekä alustalla opitun tiedon uudelleenkäytettävyyttä muussa sovelluskehityksessä. Tämä ominaisuus vaikuttaa erityisesti tekemisen tehokkuuteen. Natiivien sovelluksien konseptit ja teknologiat eroavat suuresti muiden alustojen sovelluskehityksestä. Verkkosovelluksien kehityksessä opittujen konseptien, sekä kielien avulla voidaan tehdä kehitystä useille hybridialustoille, sekä PWA-sovelluksille.

- Ominaisuuksien laajuus

Laitteen ominaisuuksien käyttäminen on myös olennainen osa sovelluskokonaisuutta. Ominaisuuksien kuten kameran käyttö on haluttu ominaisuus suuressa osaa nykyaikaisissa sovelluksissa. Kaikilla vertailtavista alustoista on mahdollisuudet valjastaa kamera käyttöönsä, mutta biometrisen tunnistautumisen ja liikesensoreiden käyttöön kaikista alustoista ei löydy tukea. Sovelluksilla, kuten natiivi-, sekä hybridisovellukset, jotka on kirjoitettu Javalla, Kotlinilla, sekä Swiftillä on laajin pääsy laitteiden ominaisuuksiin. PWA- ja websovellukset taas käyttävät HTML5 mahdollistamia ominaisuuksia, joten niillä käytettävien ominaisuuksien määrä on huomattavasti vähemmän.

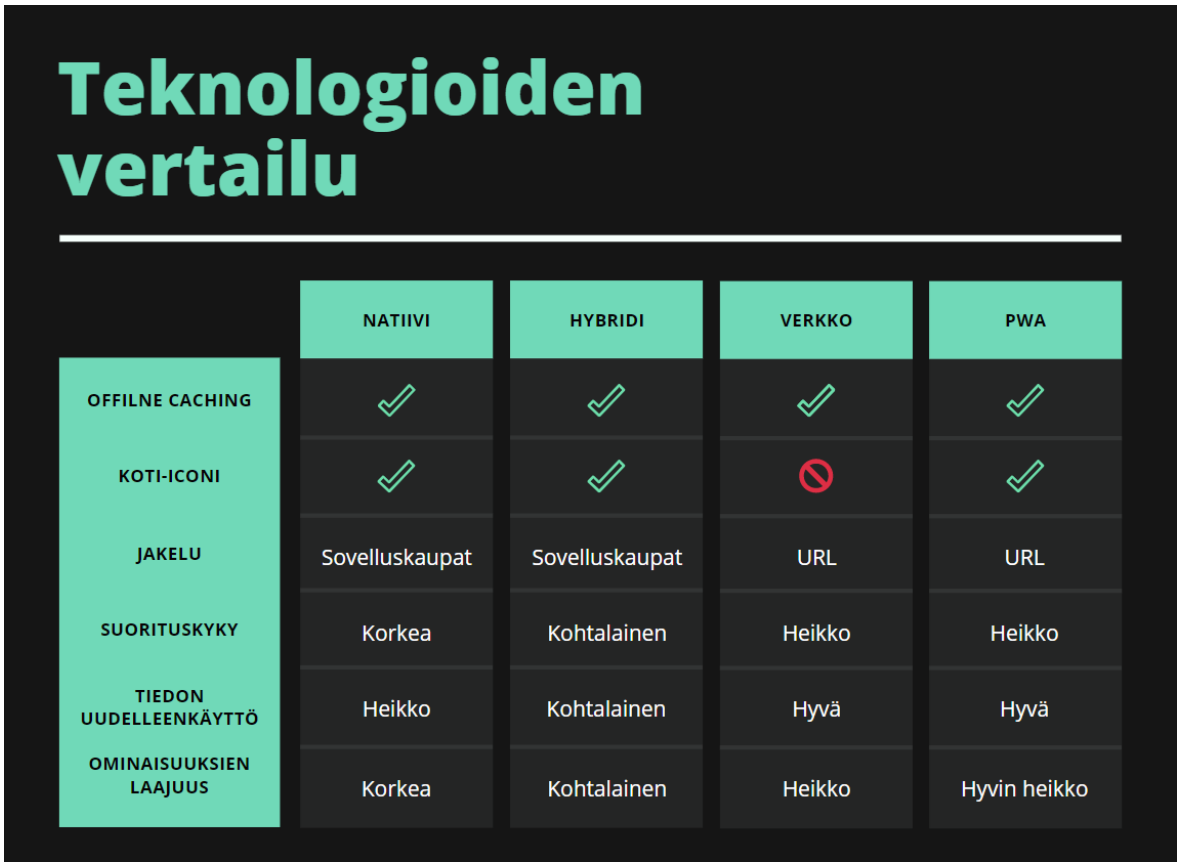
3.5.1 Tulokset

Teknologioita vertaillessa huomattiin Rossumin kiinnostuksen kohteina olevien ominaisuuksien: Offline caching:in sekä koti-ikonin toteutuksen onnistuvan kaikilla muilla alustoilla paitsi puhtailla verkkosovelluksilla. Näin ollen puhdas verkkosovellus ei ollut soveltuva Rossum Oy:n käyttöön.

Natiivisovelluksilla on laajin pääsy laitteiden ominaisuuksiin sekä paras suorituskyky muista alustoista. Tässä projektissa näillä ominaisuuksilla ei ole niin suurta merkitystä, koska heikommalla suorituskyvyllä ja rajoitetulla ominaisuuksilla voidaan tämä pilotti silti toteuttaa. Natiivisovelluksien suuri eroavaisuus Rossum Oy:n aikaisempiin sovelluksiin tarkoittaa sitä, että aiemmin opituista teknologioista saatua tietoa ei voida hyödyntää niin laajassa määrin kuin muilla alustoilla. Tämä tarkoittaisi suuria investointeja kehittäjien koulutukseen eikä näiden seikkojen, johdosta ole optimaalinen valinta Rossum Oy:n käyttöön.

Hybridisovellukset ovat yleisesti hyviä kaikessa, mutta sidonnaisuus sovelluskauppojen julkaisukäytäntöihin ja tekemisen tehokkuuden ollessa heikompi kuin valitulla teknologialla ei tähän ratkaisuun päätytty.

Tulosten pohjalta valittiin sovellusalustaksi PWA. Pyydetyt ominaisuudet kyetään toteuttamaan tällä teknologialla, mikä oli vaadittu vähimmäismäärä. Koska pyydetyt ominaisuudet voitiin toteuttaa kaikilla alustoilla, tähän valittuun teknologiaan päädyttiin vanhan tiedon uudelleenkäytön sekä laajimman jakelun perusteella.



Kuva 1. Rossum dataflow

4 TEKNOLOGIAT

PWA-sovelluksen teknologioita valittaessa otettiin käyttöön suosittu React JavaScript kirjasto. Tämä valinta pohjautui pieneen tutkimukseen suosituista PWA-teknologioista, joissa React näytti olevan usein listojen kärjessä. Ajan säästämiseksi tietokannaksi valittiin Googlen Firebase, joka mahdollistaa myös helpon käyttäjän tunnistautumisen.

4.1 React

React on Facebookin, sekä yksityisten kehittäjien ja yritysten ylläpitämä avoimen lähdekoodin front-end JavaScript kirjasto. Reactia voidaan myös kutsua nimillä ReactJS tai React.js, mutta näillä kaikilla nimillä tarkoitetaan loppupeleissä samaa asiaa. Toisin kuin esimerkiksi .NET, Angular ja Ruby on Rails, React ei ole sovelluskehys vaan JavaScript kirjasto. Muiden kirjastojen tavoin React voidaan tuoda projektiin ja hyödyntää sen menetelmiä. (c-sharpcorner 2020.)

React pohjaiset sovellukset käyttävät itseasiassa hyvin pienen osan Reactia. React sovellus on todellisuudessa verkosto työkaluja, jotka saavat tämän sovelluskokonaisuuden toimimaan. React kääntää sovelluksessa käytetyn JSX-syntaksilla kirjoitetun koodin Reactille luettavaksi koodiksi. (Codeacademy 2020) JSX-syntaksia käsitellään hetken päästä tarkemmin. React myös vahtii sovelluksessa tapahtuvien muutoksien, kuten nappuloiden painalluksen, syötetyn tekstin tai HTTP-pyynnön palaamista. Pelkistettynä React luo virtuaalisen version selaimessa ovelasta Document Object Model :sta eli DOM:sta. DOM esittää verkkosivulla olevan datan ja kun tämä data muuttuu, React luo nykyisestä DOM:sta virtuaalisen version, jota se vertaa alkuperäiseen DOM:iin. Jos muutoksia löytyy React päivittää alkuperäisen DOM:in uudella datalla. (Adhititiravi 2019.)

Seuraavassa on esitetty lista yleisistä työkaluista, joilla React sovelluksia tehdään.

- Node – Node kääntää React kehityksessä sovelluksen sekä luo tästä paketin. Node mahollistaa myös JavaScriptin suorittamisen pelkän selaimen ulkopuolella, kuten esimerkiksi käyttöjärjestelmässä. (Nodejs 2020.)
- Npm – node package manager. Maailman suurin ohjelmistorekisteri. Mahdollistaa node pakettien asentamisen, sekä komentojen ajamisen komentoriviltä. Node suorittaa alla olevia komentoja tarkastamalla projektista löytyvän package.json-tiedoston. Näitä komentoja voidaan määritellä erilaisilla valmiilla sovellus sapluunoilla tai kirjoittamalla niitä suoraan käsin. (Nodejs 2020.)

    - Npm start - Jos pakcage/script start ei ole määrittänyt muuta node käynnistää kehityspalvelimen, mikä mahdollistaa sovelluksen reaaliaikaisen testaamisen ja tarkastelun samalla kun sovellusta kehitetään. Komento luo laitteen välimuistiin tarvittavat kansiot sovelluksen käynnistämiseen, jotka hävitetään palvelimen sulkeutuessa.
    - Npm build – Rakentaa sovelluksen tuotantoon, jolloin sovellusprojekti käännetään muotoon, jossa se voidaan siirtää palvelimelle.

- Npx – Npx on komento, joka mahdollistaa node-pakettien käytön ilman asennusta. (Nodejs 2020.)

Alla olevassa kuvassa on vasemmalla lyhennetty komento ja oikealla noden ajama oikea scripti, johon lyhyempi viittaa. Noden tehtävänä on ajaa näitä scriptejä pyydettässä. Näiden scriptien lähdekoodit löytyvät Facebookin githubista ja yllä mainituista löytyviä tärkeitä riippuvuuksia käsitellään seuraavaksi. (Nodejs 2020.)

```
"scripts": {
  "start": "react-scripts start",
  "build": "react-scripts build",
  "test": "react-scripts test",
  "eject": "react-scripts eject"
},
```

Kuva 1. Rossum dataflow

- Babel – Transpiler työkalu kääntää kirjoitettua koodia vanhemmille selaimille, sekä kääntää JSX-syntaxia JavaScript koodiksi. Se on automaattisesti käytössä, jos React projekti luodaan “create-react-app" komentorivityökalulla.
- Webpack – ajettaessa “npm start" tai “npm build”-komentoja, webpack-työkalu käy läpi koko projektihakemiston ja luo siitä riippuvuuskaavion. Tästä riippuvuuskaaviosta löytyvät kaikki oleelliset osat sovelluksen suorittamiseen. Käyttäen luotua riippuvuuskaaviota hyväkseen luodaan uusi sovelluspaketti, joka koostuu vähimmäisestä mahdollisesta tiedostomäärästä. Tämä tiedosto voidaan sitten lisätä html-tiedostoon ja käyttää sovelluksen ajossa. (Freecodecamp 2019)

4.1.1 JSX

React elementtejä, sekä komponentteja toteutetaan yleisesti kirjoittamalla JavaScript- ja XML kieltä eli JSX syntaksia. JSX on JavaScript kielen laajennos, joka mahdollistaa HTML-koodin upottamisen JavaScriptiin. JSX formaatin käyttäminen ei ole välttämätöntä

React kehityksessä, mutta se tekee koodista helppolukuisempaa ja paremmin ymmärrettävää. (React Documentation 2020.)

```
return (
    <div>
        <h1>Hello Nico</h1>
    </div>
)
```

Kuva 1. Rossum dataflow

Kuvasta (Kuva 5) nähdään, että HTML:ssä käytettyjä div- sekä h1-elementtejä voidaan käyttää suoraan JavaScriptin sisällä.

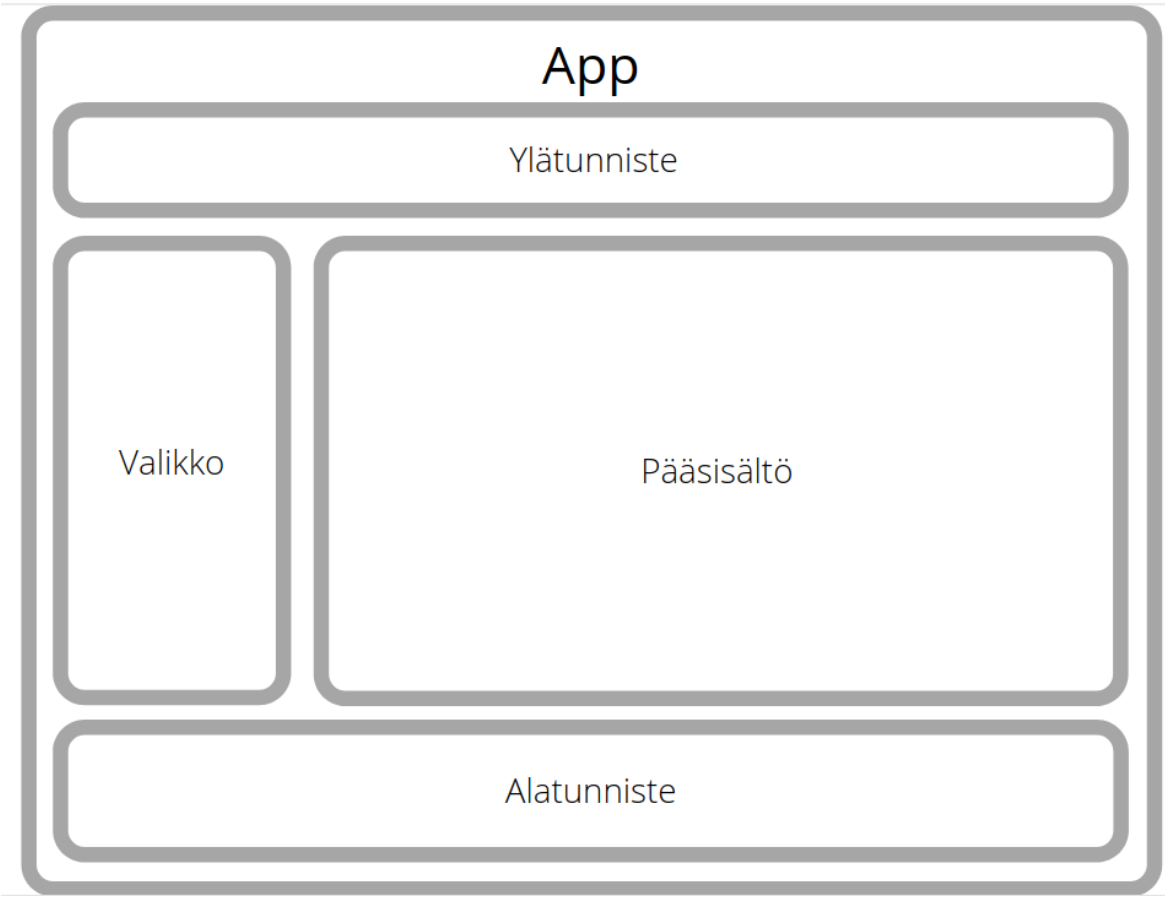
```
return (
    React.createElement("div", null,React.createElement("h1",null, "Hello Nico"))
    )
```

Kuva 1. Rossum dataflowKuva samasta komponentista, joka on kirjoitettu Reactilla.

Sama tulos saadaan kirjoittamalla Reactia ja se on esitetty yllä olevassa kuvassa (Kuva 6) , mutta komponentista on paljon vaikeampi saada selvää.

4.1.2 Komponentti

Reactissa komponentti edustaa osaa käyttöliittymästä. Komponentteja voidaan käyttää useamman kerran sovelluksen sisällä. Reactissa kaikkien komponenttien tulee olla käärittynä yhden pääkomponentin sisään. Yleensä tätä pääkomponenttia kutstutaan App-komponentiksi. Alla olevassa kuvassa on esitetty sovelluksen käyttöliittymä, joka koostuu viidestä komponentista. Ylätunnisteesta, pääsisällöstä, valikosta, alatunnisteesta, sekä App:sta. Muut neljä komponenttia ovat kääritty App:n sisään. (React Documentation 2020.)



Kuva 1. Rossum dataflowEsimerkki komponentti hierarkiasta

4.1.3 Komponenttityypit

React komponentteja voidaan tehdä kahdella eri tavalla: funktionaalisina komponentteina tai luokka komponentteina. Kummatkin komponenttityypit tarvitsevat renderöintifunktion näyttääkseen sen sisällä kirjoittetun JSX-sisällön. Funktionaalinen komponentti on itsessään renderöintifunktio, jossa ei erikseen tarvitse kutsua render()-funktiota. Luokkakomponenteissa taas kaikki JSX-sisältö luodaan render()-funktion sisään. Kuvassa alhaalla on esitetty funktionaalinen komponentti ja alhaalla luokkakomponentti. Kummatkin näyttävät Reactille samalta ja saavat aikaan saman lopputuloksen. Funktionaalisten komponenttien siisteyden takia tämä on suositeltu tapa toteuttaa komponentit, joten insinöörityössä komponentit tullaan tekemään niitä käyttäen. (React Documentation 2020.)

```
const Tervehdys = () => {
    return (
    <h1>Tervehdys Nico</h1>
        )
 }

class Tervehdys extends React.Component {
  render() {
  return <h1>Tervehdys Nico</h1>
  }
}
```

Kuva 1. Rossum dataflowFunktionaalinen komponentti sekä luokkakomponentti

4.1.4 Funktiot

Funktiot mahdollistavat sovelluksien osien uudelleenkäytön. Niillä pyritään saamaan koodista helppolukuisempaa, sekä poistamaan koodin turhaa toistoa. Jos koodia kirjoitettaessa huomataan, että samaa koodin osaa toistetaan useampaan kertaan, kannattaa siitä pyrkiä tekemään funktio. (React Documentation 2020.)

4.1.5 Prop

“Prop”-arvoa voidaan ajatella funktion argumenttina. Tämä on siis jokin arvo, joka tulee funktion ulkopuolelta. Sana “prop” on lyhenne propertysta ja niitä käytetään Reactissa tiedon välittämiseen pääkomponentilta lapsikomponentille. Yleisenä nyrkkisääntönä Reactissa on, että komponenttia luotaessa, sen “property”-arvoja ei saa muuttaa. Kuvassa alhaalla tehdään pääkomponentti App, joka määrittelee parametrin “nimi”, jonka arvona on merkkinonoksi tallennettu “Nico”. Lapsikomponentissa “Tervehdys” otetaan vastaan annettu parametri, joka liitetään otsikkoelementtiin, kutsumalla sitä aaltosulkeiden sisällä props.nimi arvolla. (React Documentation 2020.)

```
//Pääkomponentti
function App() {
  return (
    <Tervehdys nimi="Nico"></Tervehdys>
  );
}

//Lapsikomponentti
const Tervehdys = (props) => {
  return (
  <h1>Tervehdys {props.nimi}!</h1>
      )
}

export default App;
```

Kuva 1. Rossum dataflowPääkomponentti ja lapsikomponentti

4.1.6 useState

Jos propseja käytetään funktion ulkopuolisen datan näyttämiseen, niin useStatea käytetään datan muuttamiseen funktionaalisen komponentin sisällä. Tämä on suhteellisen uusi tapa Reactille uudelleenkäyttää tilallista logiikkaa muuttamatta funktionaalisten komponenttien hierarkiaa. Ongelmana on ollut komponenttien osien jakamisen vaikeus, sekä toisiinsa liittyvien komponenttien pirstoutuminen. Komponenttia ladattaessa se on saattanut suorittaa metodeita, jotka voivat aiheuttaa virheitä sovelluksessa. Tätä varten on kehitetty koukkuja (hooks), jotka ratkaisevat näitä ongelmia.

useStaten käyttämiseksi täytyy sovellukseen tuoda useState koukku, jolla saadaan tarvitut metodit käyttöön. UseState-koukku suoritetaan aina funktion alussa, jossa sille annetaan aloitusarvo. UseState-funktio palauttaa aina taulukon, jossa on kaksi arvoa. Ensimmäinen arvo on useStaten nykyinen arvo ja toinen arvo on funktio, joka mahdollistaa nykyisen arvon päivittämisen.

Seuraavassa kuvassa (Kuva 10) on esitetty yksinkertainen laskuri, jonka numeraalinen arvo on alkutilanteessa 0. Painettaessa nappia, nappi kutsuu vähennä- tai lisääNumero-funktiota. Tämä funktio kutsuu setNumero funktioita, jonka parametrina on funktio, jonka input parametrina on nykyinen arvo ja se palauttaa uuden arvon. (React Documentation 2020.)

```
function App() {
  return (
    <Counter></Counter>
  );
}

 const Counter = () => {
  const [numero, setNumero] = useState(0)

   function vähennäNumero(){
    setNumero(vanhaNumero => vanhaNumero- 1)
   }
   function lisääNumero() {
    setNumero(vanhaNumero => vanhaNumero + 1)
   }

   return(
  <div>
    <button onClick={vähennäNumero}>-</button>
   <span>{numero}</span>
    <button onClick={lisääNumero}>+</button>
  </div>
   )

 }
export default App;
```

Kuva 1. Rossum dataflowYksinkertainen laskuri

4.2 Progressiivisen web sovelluksen avaintekijöitä.

Seuraavassa on esitelty Progressiivisen web sovelluksen avaintekijöitä.

- Service Worker

Service worker on progressiivisen web sovelluksen selkäranka. Se toimii virtuaalisena välityspalvelimena (virtual proxy) selaimen ja verkon välillä. Service worker toimii omassa säikeessä sivussa sivun omasta pääkoodista, eikä sillä ole pääsyä document object modeliin eli DOM:iin. DOM:in sijaan service worker voi hyödyntää tallennettuja objekteja, tehdä verkkopyyntöjä käyttämällä Fetch rajapintaa**,** sekä tallentaa dataa. Hyödyntämällä esimerkiksi Cache Storage Apia voidaan tallentaa sovelluksen osia laitteen välimuistiin, mikä mahdollistaa kyseisten osien välittömän latautumisen seuraavalla käyttökerralla.

Näin mahdollistetaan natiivisovellusten kaltainen käytettävyys, sekä nopeus. Tällä samalla tekniikalla mahdollistetaan myös yhteydettömässä tilassa luodun datan tallennus palvelimelle yhteyden palattua. Service workerit kykenevät myös antamaan käyttäjälle push-ilmoituksia, sekä suorittamaan raskaita laskentaoperaatiota erillisessä säikeessä. Koska service workerillä on mahdollisuudet siepata, muokata, sekä tehdä räätälöityjä verkkopyyntöjä on niitä käytettäessä käytettävä suojattua HTTPS protokollaa. (Mozilla 2020)

- Manifest

Progressiivinen verkko sovellus vaatii toimiakseen Manifest-tiedoston, mikä tarjoaa tärkeää yleistä tietoa sovelluksesta selaimelle. Tämä JSON tiedosto kertoo myös käyttäjän työpöytä- tai mobiilisovellukselle, kuinka sen tulee tässä ympäristössä toimia. Manifestista yleisesti löytyviä tietoja on esitetty seuraavassa:

- Name – Sovelluksen nimi, joka tulee näkyviin useissa sovelluksen paikoissa, kuten esimerkiksi asennusruudun laidassa. Tämä kenttä on pakollinen. Kentän suurin merkkimäärä on 45 merkkiä. (Developer Chrome 2020)
- Short_name – Lyhennetty sovelluksen nimi. Vaihtoehtoinen tieto, jonka tilalla käytetään “name”:n arvoa, jos short_namea ei ole määritelty. Tämä arvo ottaa paikan työpöytäsovelluksen ikonin alta, sekä selaimen välilehden yläreunasta.
- Icons – Sovellukselle tarkoitettu logo. Logoja voidaan määritellä useille eri alustoilla ja niiltä täytyy löytyä src-, sizes-, sekä type-ominaisuus.
- Start_url – Sovelluksen ensimmäinen sivu ja se kertoo selaimelle mikä on ensimmäinen sivu, joka näytetään sovelluksen avanneelle käyttäjälle. Tällä tavoin estetään sovelluksen avautuminen siltä sivulta, mihin käyttäjä jäi ennen kuin lisäsi sovelluksen kotivalikkoon.
- Background_color – Sovelluksen taustaväri. Tätä ominaisuutta käytetään aloitusruudun taustavärinä ensimmäisen käynnistyksen yhteydessä.
- Display – Sovelluksen ulkoasu. Tällä arvolla määritellään sovelluksen ulkoasu käynnistettäessä. Sovellus voidaan käynnistää monissa eri tyyleissä, kuten fullscreen, standalone, minimal-ui, sekä browser.
- Scope – Sovelluksen sisäiset linkit. Tällä ominaisuudella määritellään sovelluksen sisäiset linkit. Jos käyttäjä painaa linkkiä, jota ei ole määritelty scope omainaisuuden sisälle aukeaa se PWA:n sisällä, eikä selaimen ikkunassa.

- Theme_color – Työkalurivin väri. Ominaisuus määrittää sovelluksen työkalurivin värin
- Shortcuts – Pikavalinnat. Lista app shortcut objekteista, joiden tavoitteena on tarjota nopea pääsy tärkeisiin sovelluksen tehtäviin. (Web dev 2020)

4.3 Firebase

Firebase on Googlen tarjoama alusta mobiilisovellusten tarvitsemien palveluiden toteuttamiseen. Se helpottaa sovelluksen kehittämistä tarjoamalla työkaluja yleisesti oleellisten sovellusten ominaisuuksien luomiseen, joita kehittäjä ei välttämättä itse halua käsin tehdä. Tallaisia ominaisuuksia ovat esimerkiksi käyttäjien todennus, analytiikka, tietokantojen luominen ja niin edelleen. Firebasen ominaisuudet on suunniteltu toimimaan yhdessä ja niitä kakkia hallitaan Firebasen omasta konsolista. Tällä konsolilla kehittäjä pääsee vuorovaikuttamaan suoraan näihin ominaisuuksiin, jolloin tarve väliohjelmistolle palvelimen ja sovelluksen väliin katoaa. Jos kehittäjä haluaa käsitellä jollain tavoin palvelimella olevaa dataa esimerkiksi Firebasen Firestroressa, hän tekee sen suoraa kutsumalla rajapintaa sovelluksessa. Alla olevassa kuvassa nähdään Firebasen luomat avaimet, joilla saadaan sovelluksesta yhteys Firebasen rajapintaan. Firebase on oivallinen alusta testisovellusten toteuttamiseen. Se on lähtökohtaisesti ilmainen ja veloittaa vasta sovelluksen käytön kasvaessa. (Firebase Google 2020) (Medium 2020.)

```
const firebaseConfig = {
  apiKey: "AIzaSyCU4Eh5QbmDx1Zpt1ws_vCTQUFoVS2gSZo",
  authDomain: "login-and-dataupload.firebaseapp.com",
  databaseURL: "https://login-and-dataupload.firebaseio.com",
  projectId: "login-and-dataupload",
  storageBucket: "login-and-dataupload.appspot.com",
  messagingSenderId: "313284002306",
  appId: "1:313284002306:web:c9999905cebb2594125d1b"
};
```

Kuva 1. Rossum dataflowFirebasen configuraatioavaimet.

4.4 Material UI

Material UI on komponenttikirjasto, joka mahdollistaa Material Design suunnitteluohjeen mukaisten käyttöliittymien toteuttamisen resurssitehokkaasti. Komponentit on kehitetty Material Design pohjalta, joka on eräänlainen käyttöliittymäkomponettien parhaiden käytäntöjen ohjeistus. Material UI tarjoaa kattavan määrän erilaisia valmiiksi toteutettuja syöttökenttiä, navigointielementtejä, sekä datan visualisointiin oleellisia komponentteja kuten taulukoita. (Material UI 2020.)

Material UI:n käyttämät komponentit ovat itsenäisiä ja ne lataavat vain käytetyt elementit, joten turhien tyylien lataaminen jää pois. Lisäksi käytetyt komponentit ovat helposti muunneltavia ja Material UI tarjoaa selkeät ja kattavat ohjeet komponenttien käyttöön. (Material UI 2020.)

5 SOVELLUSPROJEKTI

5.1 Suunnittelu

Projektin ensimmäinen osa oli sovelluksen ominaisuuksien kartoitus. Muutamaa ominaisuutta kuten verkottomassa tilassa toimimista, sekä sovelluksen käynnistämistä kotivalikosta oli pyydetty aikaisemmin. Yleistä toimivuutta varten valittiin sovellukselle vaatimuksiksi käyttäjän autentikointi, sekä tiedoston tallentaminen pilveen. Sovellus toteutettiin käyttämällä Reactia. Käyttäjien tunnistautuminen, sekä tietokanta tehtiin käyttäen Firebasen eri ominaisuuksia.

Sovellukselle ei ollut valmiiksi suunniteltua ulkoasua. Ajan säästämiseksi valittiin jo tuttu Material design kirjasto ja pyrittiin noudattamaan tämän asettamia parhaita käytäntöjä.

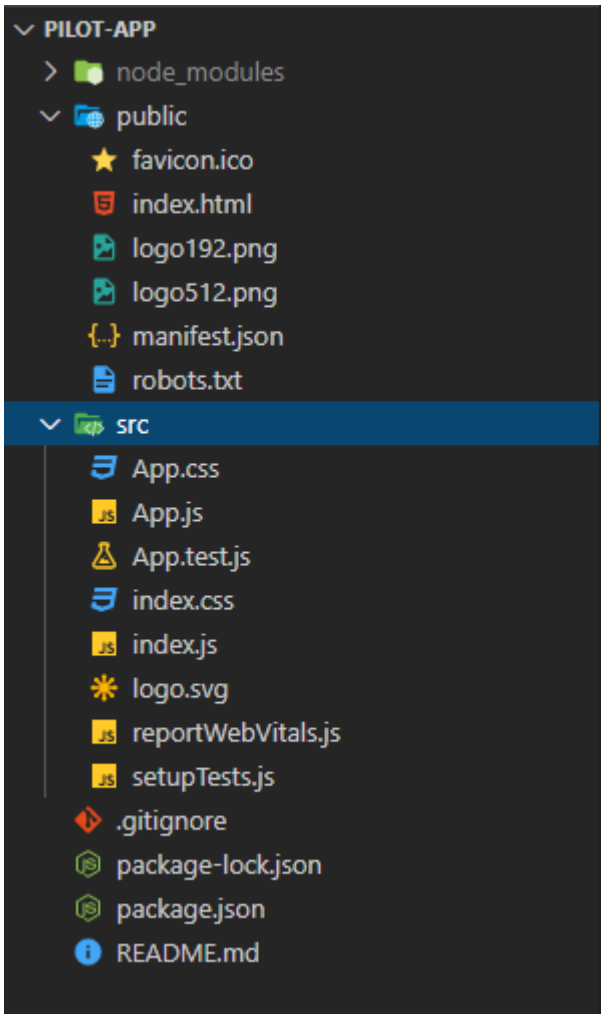
5.1.1 Projektin luonti

Projekti luotiin cli työkalun avulla ajamalla seuraavassa kuvassa (Kuva 12) esitetyllä tavakka. Tämä yleinen ja suosittu komento on hyvä pohja testiprojektille, ja se määrittää tarvittavat työkalut sovelluksen kehitykseen. React projekteja luotaessa täytyy ottaa huomioon heidän uusi nimeämiskäytäntönsä, jossa kaikkien uusien projektien nimet pitää kirjoittaa pienillä kirjaimilla.

```
PS D:\Coding\Oppari> npx create-react-app pilot-app
```

Kuva 1. Rossum dataflowKomento React projektin luomiseen

Komento luo kansiorakenteen, joka sisältää React sovelluksen rungon. Valmiin sovelluksen tarkoituksena on antaa olla toimiva pohja sovellukselle, josta jatkokehittäminen onnistuu helposti.



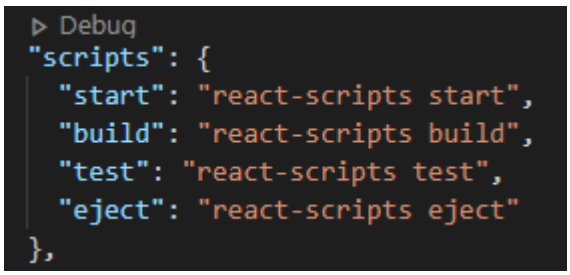
Kuva 1. Rossum dataflowReact kansiorakenne

Kansiorakenne sisältää kolme kansiota: node_modules, public ja src. Node_modules-kansio sisältää kaikki tarvittavat moduulit mitä sovellus tarvitsee. Oletuksena tämä kansio sisältää React sekä React-Dom moduulit. Public-kansio on kansio johon sovelluspakettia kääntävä Webpack ei koske. Kansio sisältää valmiin favicon-ikonin, joka lisää selaimen yläkulmaan React-logon. Index.html-tiedosto sisältää sapluunan kehitettävälle sovellukselle. Tässä projektissa ei tähän tiedostoon tehty muutoksia, mutta on oleellista ymmärtää, että React tarvitsee tätä tiedostoa oman sisältönsä näyttämiseen. React pääsee näyttämään sisältöä index.html-tiedostossa olevan div-tagin, jonka id on ”root” avulla. Public kansiosta löytyy myös aikaisemmin mainittu manifest.json-tiedost. (c-sharpcorner 2020.).

Seuraava kansio on src-kansio. Tämän kansion sisään tehdään suurin osa React kehityksessä tehdystä työstä. Index.js-tiedoston tehtävä on määritellä App-komponentti, joka näyttää sisältöä index.html-tiedoston root nimisen divin sisällä. Kansio sisältää myös

App.js-tiedoston, johon renderöitävä sisältö laitetaan. Se sisältää sekä tyylittelyjä että testaamista varten tehtyjä tiedostoja. (Medium 2018.)

Pilottisovelluksen juuritasolta löytyy package.json-tiedosto. Tämä tiedosto sisältää projektiin liittyviä “asetuksia”, kuten riippuvuuksia, sekä scriptejä. Projekti käynnistetään ajamalla npm start komento. Projektin luonnin yhteydessä luodaan myös automaattisesti manifest.json-tiedosto.



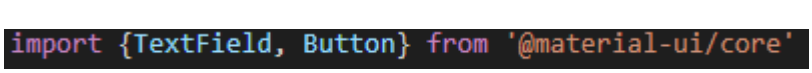
Kuva 1. Rossum dataflowManifest.json-tiedoston scriptejä

5.1.2 PWA ominaisuudet

Konfiguroimalla manifest.json-tiedosto sekä serviceWorker oikein saadaan sovellus lautautumaan verkottomassa tilassa, sekä selaimet tunnistamaan sovellus PWA-sovellukseksi. Laitteiden selaimet ehdottavat automaattisesti työpöytäsovelluksen asennuksen mahdollisuudesta. Käyttäjän kotinäytölle tulee ikoni, josta sovellus voidaan käynnistää vaivattomasti.

5.1.3 Material Ui

Pilotissa on käytetty Material Ui-kirjaston komponentteja. Saadakseen nämä komponentit toimimaan tarvitsee asentaa package.json-tiedostoon riippuvuudet tästä kirjastosta. Riippuvuudet asennetaan ajamalla kansion juuritasolla ”npm install @material-ui/core”. Asentamisen jälkeen komponentteja voidaan käyttää seuraavassa kuvassa (Kuva 15) esitellyllä tavalla. Sovelluksessa on käytössä tekstikenttä sekä paino-nappi. (Material Ui 2020.)



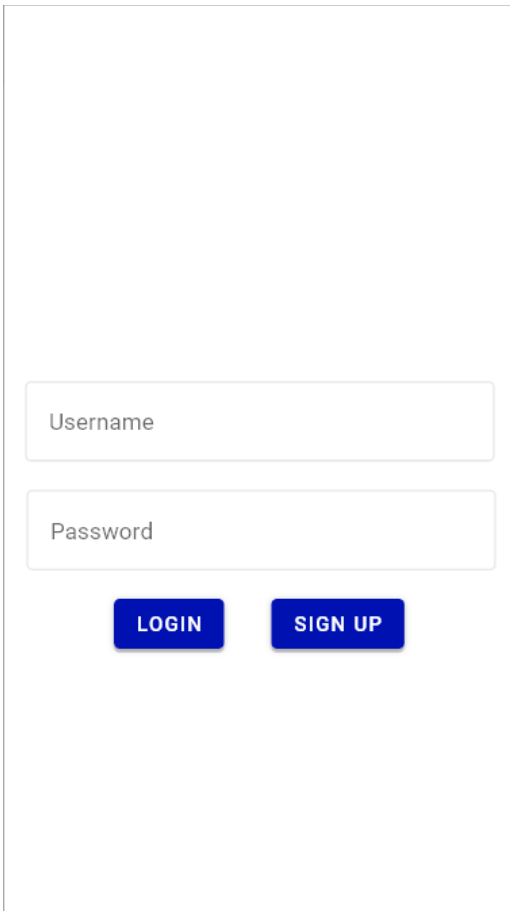
Kuva 15. Material Ui komponenttien tuonti projektiin.

5.1.4 Käyttäjän varmennus

Sovelluksen tarkoituksena on luoda käyttäjälle mahdollisuus tehdä uusi käyttäjä, sekä kirjautua tehdyllä käyttäjällä sisään. Käyttäjän autentikointiin käytettiin Googlen Firebase palvelua, joka tekee suurimman osan työstä valmiiksi. Toimiakseen tarvitaan Firebaseen

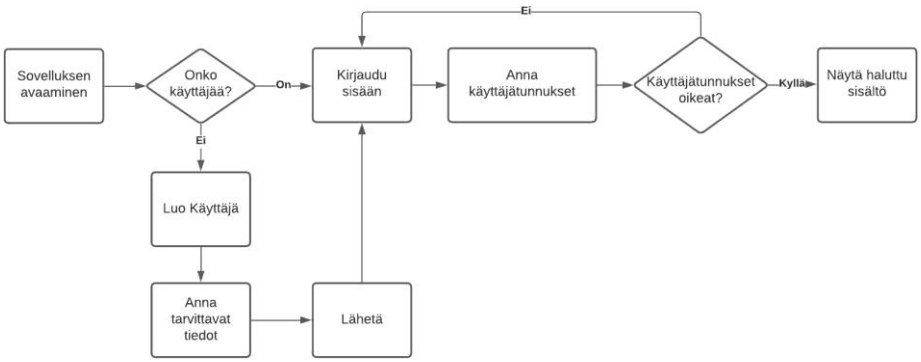
käyttäjä, sekä sinne luotu projekti, josta saadaan tarvittavat rajapinta-avaimet. Yhdessä Firebase-projektissa voi olla useampia sovelluksia, joten sovelluksien selkeä nimeäminen on tärkeää.

Käyttäjä voi alla olevasta pilotista(Kuva 16) luoda käyttäjän syöttämällä sähköpostiosoitteen, sekä salasanan tai kirjautua jo tehdyillä tunnuksilla sovellukseen. Jos käyttäjän tunnukset vastaavat Firebaseen luotuja tunnuksia päästetään käyttäjä näkemään seuraava sivun.



Kuva 1. Rossum dataflowNäkymä käyttäjän sisäänkirjautumisesta

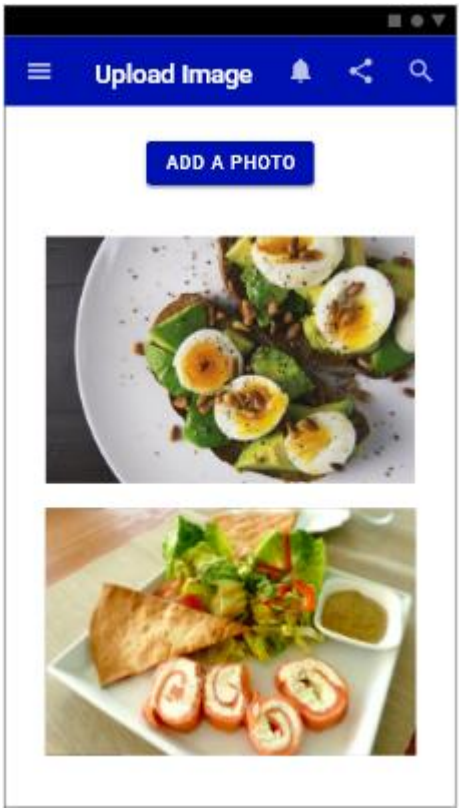
Käyttäjän varmennuksessa käytetään Firebasen tarjoamaa auth nimiavaruutta. Auth tarjoaa tarvittavat metodit käyttäjän luomiseen, kirjautumiseen, sekä seuraamisen. Firebase olettaa, että syötetty sähköposti on formatoitu oikein, joten kehittäjän tulee tarkastaa, onko sähköposti oikeassa muodossa. Alla olevassa kuvassa on kuvattu sisäänkirjautumisen toimintaperiaate.



Kuva 1. Rossum dataflowFirebasin sisäänkirjautumisen dataflow

5.1.5 Kuvan lataaminen

Käyttäjä lataa kuvan painamalla nappia. Nappia painaessa käyttäjälle aukeaa ikkuna, josta hän voi valita haluamansa kuvat tietokantaan lähetettäväksi. Kuvat lähetetään Firebasen Firestore tietokantaan. Firestore tallentaa dataa kansioittain ja tässä tapauksessa tiedot tallennetaan ”images”-kansioon. Käyttäjän lisäämät kuvat haetaan tietokannasta ja listataan sovellukseen allekkain.



Kuva 1. Rossum dataflowKuvan lataamisen ulkoasu

6 YHTEENVETO

Tämän opinnäytetyön tavoitteena oli perehtyä, miten eri tavoin voidaan sovelluskehitystä eri alustoille tehdä, sekä millaisia ongelmia nämä alustat ratkaisevat. Tavoitteena oli valita Rossum Oy:n käyttöön sopiva sovellusalusta ja toteuttaa pilottisovellus, jolla teknologian toimivuutta testataan. Opinnäytetyössä käsitellään natiivi-, hybridi-, verkko- sekä PWA-sovelluksien teoriaa, joiden pohjalta vertailu Rossum Oy käyttöön soveltuvasta teknologiasta toteutettiin. Vertailussa huomattiin asiakkaiden pyytämien ominaisuuksien olevan mahdollista toteuttaa kaikilla muilla paitsi verkkosovellusalustalla. Näin ollen valinta ottaa PWA-sovellus muiden alustojen yli tuli tekemisen tehokkuuden sekä jaettavuuden pohjalta.

Asetetut tavoitteet toteutettiin onnistuneesti ja jatkokehitystä voidaan tehdä Rossum Oy:n toimesta. React ja Firebase tuntuivat olevan hyvä ratkaisu yksinkertaisen pilottisovelluksen tekoon. Reactin sekä Firebasen suuren käyttäjämäärän takia informaatiota sekä ohjeita oli helppo löytää. Näistä opittua voidaan myös soveltaa jatkokehityksessä.

Sovelluskehitys on viimevuosien aikana kehittynyt huimasti. Markkinoilla olevien mobiililaitteiden määrä on kasvanut, niiden monipuolisuus on kehittynyt sekä niiden suorituskyky on noussut. On tullut tarve tehokkaimmille, sekä käyttäjäystävällisemmille mobiilisovelluksille todennäköisesti kasvamaan huomattavasti. Joten kyky kehittää sovelluksia näille alustoille tehokkaasti, sekä älykkäästi tulee olemaan tärkeää.

Sovelluskehityksessä tulee aina olemaan monia erilaisia vaihtoehtoja. Näistä kaikista vaihtoehdoista tulee löytymään huonoja, sekä hyviä ominaisuuksia. Tärkeää on käyttää aikaa ennen sovelluskehitystä kartoittamaan, mitä vaatimuksia sovelluksella on. Käy halvemmaksi käyttää aikaa alussa ja tehdä sovellus oikein, kuin aloittaa sovelluskehitys ja huomata, että valittu sovelluskehys ei tarjoa tarvittuja ominaisuuksia.

Joten miten sinun kannattaisi valita sovelluksesi? Ellei tarkoituksena ole rakentaa paljon suorituskykyä vievää sovellusta kuten esimerkiksi peliä, saattaa olla oikea valinta tutkia progressiivisten web sovellusten tuomaa, helpotettua kehitystapaa, kustannussäätöjä, sekä yhteensopivuutta muiden alustojen kanssa.

LÄHTEET

Adhithiravi 2019. React Virtual DOM Explained in Simple English [viitattu 10.11.2020]. Saatavissa: https://adhithiravi.medium.com/react-virtual-dom-explained-in-simple-english-fc2d0b277bc5

Android Studio Documentation 2020 [viitattu 1.03.2020]. Saatavissa: https://developer.android.com/guide/components/fundamentals?hl=en

Apple 2020 [viitattu 01.03.2020]. Saatavissa: https://developer.apple.com/documentation

Careerfoundry 2018. What Is The Difference Between A Mobile App And A Web App? [viitattu 02.02.2020]. Saatavissa: https://careerfoundry.com/en/blog/web-development/what-is-the-difference-between-a-mobile-app-and-a-web-app/

Chrome 2020. Manifest - Name and Short Name [viitattu 27.11.2020]. Saatavissa: https://developer.chrome.com/apps/manifest/name

Codeacademy 2020. [viitattu 28.11.2020]. Saatavissa: https://www.codecademy.com/learn/bwa-intro-to-react/modules/react-101-jsx-u/cheatsheet

C-sharpcorner 2019. Folder Structure In React [viitattu 8.11.2020]. Saatavissa: https://www.c-sharpcorner.com/article/folder-structure-in-react/

Digitaltrends 2020. Android vs. iOS: Which smartphone platform is the best? [viitattu 1.03.2020]. Saatavissa: https://www.digitaltrends.com/mobile/android-vs-ios/

Firebase 2020. [viitattu 1.11.2020]. Saatavissa: https://firebase.google.com

Freecodecamp 2019. An intro to Webpack: what it is and how to use it [viitattu 4.11.2020]. Saatavissa: https://www.freecodecamp.org/news/an-intro-to-webpack-what-it-is-and-how-to-use-it-8304ecdc3c60/

Google support 2020 [viitattu 5.03.2020]. Saatavissa: https://support.google.com/googleplay/android-developer/answer/6334282?hl=en

Google developers 2020 Introduction to Progressive Web App Architectures [viitattu 1.11.2020]. Saatavissa: https://developers.google.com/web/ilt/pwa/introduction-to-progressive-web-app-architectures

Howtogeek 2020. What Are Progressive Web Apps? [viitattu 4.03.2020]. Saatavissa: https://www.howtogeek.com/342121/what-are-progressive-web-apps/

Kambda 2020. What Is Native Mobile App Development [viitattu 03.03.2020]. Saatavissa: https://www.kambda.com/what-native-mobile-app-development-definition-benefits/

Makeuseof 2020. What Is iOS? Apple's iPhone Software Explained [viitattu 01.03.2020]. Saatavissa: https://www.makeuseof.com/tag/what-is-ios/

Material Ui 2020. [viitattu 25.11.2020]. Saatavissa: https://material-ui.com/

Medium 2018. What is Firebase? The complete story, abridged [viitattu 28.11.2020]. Saatavissa: https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0

Mozilla 2020. Making PWAs work offline with Service workers [viitattu 26.11.2020]. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Offline_Service_workers

NodeJS 2020. [viitattu 9.11.2020]. Saatavissa: https://nodejs.org/en/docs/

ReactJS 2020. Components and Props [viitattu 9.11.2020]. Saatavissa: https://reactjs.org/docs/components-and-props.html

ReactJS 2020. State and Lifecycle [viitattu 9.11.2020]. Saatavissa: https://reactjs.org/docs/state-and-lifecycle.html

Uxplanet 2019. How page speed affects Web User Experience [viitattu 15.03.2020]. Saatavissa: https://uxplanet.org/how-page-speed-affects-web-user-experience-83b6d6b1d7d7

Web 2020. Add a web app manifest [viitattu 26.11.2020]. Saatavissa: https://web.dev/add-manifest/

Web 2020. What are Progressive Web Apps? [viitattu 6.03.2020]. Saatavissa: https://web.dev/what-are-pwas/

Winningwp 2020. What is Website Caching and Why is it so Important? [viitattu 25.11.2020]. Saatavissa: https://winningwp.com/what-is-website-caching-and-why-is-it-so-important/

Yugasa 2020. PWA VS NATIVE APP 2020: WHICH IS BETTER? [viitattu 7.03.2020]. Saatavissa: https://yugasa.com/mobile-apps/pwa-vs-native-app-which-is-better/