Tampereen yliopisto

Jermu Toiviainen

ALUSTARIIPPUMATTOMAN MOBIILIARKKITEHTUURIN TOTEUTUS REACT NATIVELLA

Diplomityö
Informaatioteknologian ja viestinnän tiedekunta
David Hästbacka
Syyskuu 2022

TIIVISTELMÄ

Jermu Toiviainen: Alustariippumattoman mobiiliarkkitehtuurin toteutus React Nativella
Diplomityö
Tampereen yliopisto
Tietotekniikan maisterikoulutus
Syyskuu 2022

Alustariippumattomien toteutusten suosio on ollut kasvussa viime aikoina. Alustariippumattomalla arkkitehtuurilla pystytään vastaamaan usean eri alustan tarpeeseen ilman että sovelluskehitysresursseja joudutaan jakamaan eri alustojen kesken. Alustariippumattomuus tuo kuitenkin omia haasteita verrattuna natiivisovelluksiin. Suorituskyky usein kärsii alustariippumattomuuden seurauksena ja eri alustojen mukauttaminen vastaamaan haluttua lopputulosta tuo omia haasteita arkkitehtuurille. Käytännössä muutos on kuitenkin niin pieni, ettei sitä loppukäyttäjä huomaa.

Tämän tutkimuksen tavoitteena oli selvittää kuinka React Native soveltuu alustariippumattoman arkkitehtuurin toteutukseen. Tutkimuksessa toteutettiin React Nativella autopesulan prototyyppi maksusovellus Euro Car Wash nimiselle yritykselle, jonka avulla sovelluskehystä arvioitiin sille asetettuja tutkimuskysymyksiä vasten. Tutkimus toteutettiin konstruktiivisena tutkimuksena tarkoittaen ensin tutustumista aiheen teoriaan ja tämän tiedon pohjalta konkreettisen ratkaisuehdotuksen toteuttamista sovelluksen muodossa ja sen analysoimista seitsemää kriteeriä vasten. Tutkimuksessa todettiin React Nativen soveltuvuus alustariippumattomana sovelluskehitysalustana.

Avainsanat: React Native, React alustariippumattomuus, TypeScript, JavaScript, Android, iOS, arkkitehtuuri

ABSTRACT

Jermu Toiviainen: Cross-platform mobile architecture developed with React Native
Master of Science Thesis
Tampere University
Master's Degree Programme in Information Technology
September 2022

The popularity of cross-platform products has been increasing in recent times. Cross-platform architecture design enables the development of multiple platforms without the needing to divide the development resources between multiple platforms. Cross-platform development does however bring some challenges compared to native development. Performance and some customization features may often suffer because of the use of cross-platform framework. However, this isn't often noticed by the end-user.

The motivation for this study was to research how does React Native framework work with a cross-platform architecture design. In this study a carwash prototype payment app wash developed for a company named Euro Car Wash to research the framework via the given research questions. The study was done as constructive research, meaning that first the theory of the subject is research and then based on this knowledge a concrete solution is developed and analyzed via the seven analysis criterions. The study concluded the validity of the React Native as a capable cross-platform framework.

Keywords: React Native, React cross-platform, TypeScript, JavaScript, Android, iOS, architecture

ALKUSANAT

Tämä diplomityö on tehty yhteistyössä Euro Car Wash Oy:n kanssa. Työssä toteutettiin tutkimusta varten React Native frameworkilla mobiilisovellus itsepesulapalvelukonseptia varten. Tahtoisin kiittää työn mahdollistaneita osapuolia ohjaajaani David Hästbackaa Tampereen yliopistolta, Esa Mäkistä ja Pekka Kinnusta Euro Car Washilta tutkimuksen kanssa.

Tampereella, 28.9.2022

Päivittäjä

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

CLI	Command Line Client
DOM	Document object model
JDK	Java Development Kit
JSON	JavaScript Object Notation eli tiedostomuoto tiedonvälitykseen
JSX	JavaScript XML
MVP	Minimum viable product
npm	Node Package Manager
PoC	Proof of Concept
REST	Representational state transfer eli rajapinta datan siirtämistä varten

1. JOHDANTO

Tähän mennessä auton itsepesua varten on tarvittu usein henkilökuntaa sen hallinnoimiseksi. Tämän diplomityön tarkoituksena on tarkastella alustariippumattomia web-sovelluskehityksen malleja ja tämän pohjalta toteuttaa konseptitoteutus (PoC) auton itsepesupalvelulle, joka mahdollistaisi pesutapahtuman ilman tarvetta henkilökunnalle. Työn toimeksiantajana toimii Euro Car Wash. Palvelun tarkoituksena on mahdollistaa maksun suorittaminen halutulle pesuajalle ilman tarvetta erilliselle henkilökunnalle. Palvelussa käytännössä huoltoasemalla on pystyssä erillinen pesutoteemi, joka hallinnoi eri pesutoimien ajoa paikallisesti. Tästä rautapuolesta ollaan jo tekemässä erillinen opinnäytetyö toisen opiskelijan toimesta.

Työn haasteena on selvittää React Nativen soveltuvuus modernina alustariippumattomana mobiilikehitysalustana. Työssä on tarkoitus käytännössä toteuttaa React Nativella sovellukseen näkymä, jolla käyttäjä voi hallinnoida pesusuoritetta. React Nativella on mahdollista toteuttaa moderneja alustariippumattomia mobiilisovelluksia. Lopputuotteeksi on tarkoitus saada niin kutsuttu (MVP), eli pienin toimiva tuote jolla pesukonseptia on mahdollista lähteä testaamaan markkinoilla. Itsepesupalvelun tulee tukea alustariippumattomasti useaa eri laitetta. Tutkimuskysymyksenä lähden selvittämään kuinka React Native soveltuu mobiilikäytettävyysvaatimusten ja toiminnallisten vaatimusten täyttämiseen sekä Android että iOS alustalla.

Käytän tutkimuskysymysten selvitykseen konstruktiivista tutkimusmenetelmää. Konstruktiivisessa tutkimuksessa on tarkoituksena ensiksi perehtyä aiheen teoriapohjaan, jonka pohjalta lähdetään innovoimaan ratkaisua. Konstruktiivinen tutkimus soveltuu erityisen hyvin konkreettisten tosielämän ongelmien ratkaisuun. Empiiriset löydökset ovat tärkeä osa konstruktiivista tutkimusta. Lopuksi reflektoidaan lopputuotosta aikaisempaan teoriaan ja tehdään tämän perusteella johtopäätökset. [6]

2. ALUSTARIIPPUMATON ARKKITEHTUURI

Arkipäiväiset palvelut ovat siirtymässä yhä enenevissä määrin verkkoon. Tästä hyvänä esimerkkinä voidaan pitää ruuan kuljetuspalveluita, kuten Foodoraa ja Wolttia. Palveluiden suurimpana etuna on, että sillä voidaan vähentää tarvetta henkilökunnalle, jota tarvitaan paikan päällä. Itsepesupalvelulla halutaan vastata tähän kasvavaan tarpeeseen.

Tässä työssä toteutetaan itsepesupalvelun mobiilimaksamisen käyttöliittymä alustariippumattomasti React Nativen avulla. Alustariippumaton arkkitehtuuriratkaisu tarjoaa monipuolisemman tuen verrattuna perinteiseen natiivi mobiilikehitykseen. Kun samalla koodipohjalla pystytään vastaamaan sekä iOS että Android tarpeisiin organisaatio säästää huomattavasti tarvittavissa kehitysresursseissa. React Native onkin yksi suosituimmista alustariippumattomista mobiilikehitysalustoista ja sille löytyy paljon kehittäjätukea [13].

Alustariippumaton arkkitehtuuri tarjoaa samalla ohjelmointikielellä tuen usealle eri alustalle. Alustalla voidaan tarkoittaa prosessoria tai käyttöjärjestelmää, jolla ohjelmaa ajetaan. Se ei kuitenkaan tarkoita, että ohjelma toimisi millä tahansa alustalla, vaan alustat ovat ennalta määritelty. [25]

Alustariippumattomaton arkkitehtuuri on ollut yhtenä mobiilikehityksen viime vuosikymmenen nousevana trendinä [1]. Mahdollisuus keskittää organisaation resurssit yhteen alustaan ja samalla saaden sekä Android että iOS sovelluksen kuulostaa paperilla hyvin houkuttelevalta vaihtoehdolta. Samalla on kuitenkin oltava valmis tekemään kompromisseja usein sovelluksen suorituskyvyn kanssa. Alustariippumattomuus tuo mukanaan myös haasteita ja organisaation on analysoitava tarkkaan teknologian hyödyt ja haitat.

Suurimmat hyödyt alustariippumattomasta arkkitehtuurista organisaatio saavuttaa usein alentuneilla sovelluskehitys ja ylläpito kustannuksilla [17]. Kun pystytään käyttämään samaa koodipohjaa usealla eri alustalla, organisaatio selviää pienemmillä kehitysinvestoinneilla. Yleensä selvitään yhdellä kehittäjätiimillä kahden sijasta ja yhden sovelluksen kehittäminen on myös tavallisesti nopeampaa kuin kahden.

Isoin ongelma alustariippumattomuudessa tulee usein vastaan suorituskyvyn kanssa verrattuna aidosti natiiveihin ratkaisuihin [17]. Alustariippumaton sovelluskehys joutuu usein tekemään jonkin tason kompromisseja, kun ohjelmaa käännetään

ja suoritetaan usealla eri alustalla. Niin on myös React Nativen kanssa [7]. Alustariippumatomassa arkkitehtuurissa ei välttämättä ole saatavilla kaikkia natiiville sovellukselle tarjottuja ominaisuuksia, mikä voi heikentää sovelluksen käyttäjäkokemusta [17]. Erityisesti käyttöliittymän saaminen vastaamaan natiivia sovellusta sekä Android että iOS alustalla on vaikeaa, johtuen molempien alustojen uniikista tuntumasta.

2.1 React Nativen kilpailijat

React Native ei ole suinkaan ainoa toimija alustariippumattomassa mobiilikehitys kentässä vaan sillä on useita kilpailijoita. React Native on kuitenkin yksi tunnetuimmista alustariippumattomista mobiilikehityksen alustoista. Sen kilpailijat yleensä erottuvat joukosta joko ei natiiveina hybridi web-sovelluskehyksinä tai Googlen Flutter tapauksessa omalla erillisellä Dart ohjelmointikielellä.

Googlen kehittämä ja vuonna 2018 julkaisema Flutter on alustariippumaton mobiilikehitysalusta, jolla pystyy luomaan natiivisti kääntyviä applikaatioita Android ja iOS alustoille samalla koodikannalla. Flutterin ohjelmointikielenä toimii JavaScriptin sijasta Googlen kehittämä Dart ohjelmointikieli. Se on saanut vaikutteita C-kielen syntaksista ja on oliopohjainen ja vahvasti tyypitetty kieli. Flutterin heikkouksiin lukeutuu sen pienempi markkina-asema verrattuna React Nativeen, mistä johtuen sen kehittäjäyhteisö on pienempi ja tuki tulevaisuudelle on kyseenalaisempaa kuin vahvemman markkina-aseman omaava React Native. JavaScriptillä on myös paljon vahvempi asema kuin Dart kielellä, ja sitä osaavien kehittäjien löytäminen on haastavampaa. [2]

Muita React Nativen kilpailijoita ovat Cordova, Xamarin ja Ionic Framework. Cordovalla voidaan toteuttaa alustariippumattomia hybridisovelluksia JavaScriptillä ja HTML 5:llä. Sovellukset toimivat sekä Android että iOS laitteilla, mutta ne eivät ole natiivisovelluksia. Ionic on samaan tapaan sovelluskehys hybridisovelluksien tekoa varten. Xamarin on avoimen lähdekoodin sovelluskehys, joka käyttää .NET ympäristöä hyväkseen. Xamarin frameworkilla voi kehittää alustariippumattomasti natiivisovelluksia ja se on suosittu C# kehittäjien keskuudessa. [3]

React Nativen suosio verrattuna muihin alustariippumattomiin sovelluskehyksiin pohjaa sen käyttämään laajasti kehittäjien keskuudessa tunnettuun JavaScript-kieleen. React Nativen toisena etuna verrattuna useaan muihin alustariippumattomiin

sovelluskehyksiin on sen aito natiivisovellustuki verrattuna hybridisovelluksiin. Natiivilla sovelluksella pystytään paremmin vastaamaan käyttäjien vaatimaan tasoon sovelluksen käyttäjäkokemuksesta.

3. ALUSTARIIPPUMATTOMAT MOBIILISOVELLUKSET

Tässä luvussa käydään läpi diplomityöhön olennaisesti liittyviä teknologioita ja käsitteitä, jotka liittyvät React Nativeen ja alustariippumattomaan mobiilikehitykseen. Työn kannalta tärkeimmät teknologiat ovat React ja TypeScript ja niiden alaisuuteen vahvasti liittyvät teknologiat, kuten Node.js ja npm paketinhallintajärjestelmä.

3.1 React

React on ilmainen avoimen lähdekoodin JavaScript sovelluskehys. React julkaistiin ensimmäisen kerran kahdeksan vuotta sitten vuonna 2013 Facebookin eli nykyisen Metan toimesta. [29]

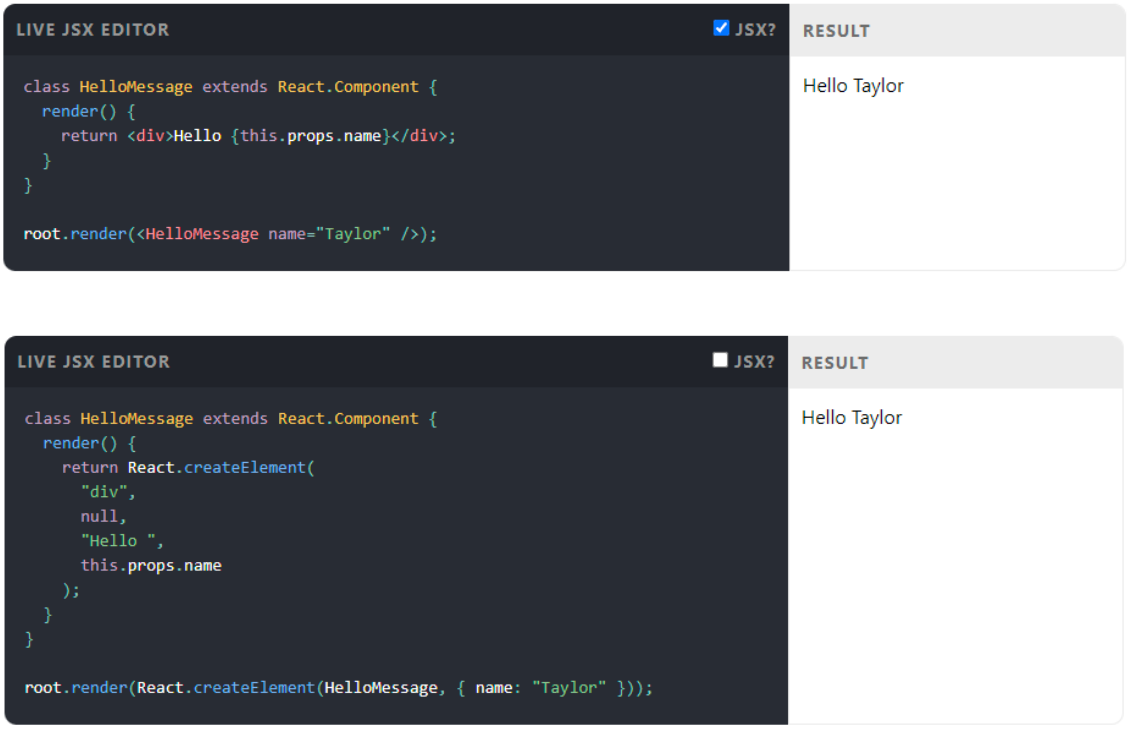
Avoin lähdekoodi tarkoittaa vapaasti käytössä olevaa ohjelmakoodia. Sitä on mahdollista muokata ja käyttää myös kaupallisiin tarkoituksiin [19]. React Native on myös avoimen lähdekoodin projekti. Muita suosittuja avoimen lähdekoodin projekteja on Linux, Android-käyttöjärjestelmä ja Firefox selain. Avoimen ja suljetun lähdekoodin erona on, että kuka tahansa pääsee tarkastelemaan ja tekemään muutoksia koodiin toisin kuin suljetussa projektissa koodiin pääsee käsiksi vain rajattu joukko [19].

Avoimen lähdekoodin projektit ovat kasvattaneet suosiotaan viime vuosikymmeninä. Avoimen lähdekoodin ympärille onkin usein kasvanut myös erilaisia kehittäjäyhteisöjä. Kun koodin tuottamiseen ja tarkasteluun osallistuu suurempi yleisö koodin laatu ja tietoturvallisuus kasvaa verrattuna suljettuun ohjelmistoprojektiin. [28]

React käyttää niin sanottua deklaratiivista ohjelmointimallia, joka on imperatiivisen ohjelmoinnin vastakohta [29]. Deklaratiivisessa ohjelmoinnissa näytetään haluttu tila itse varsinaisen algoritmin sijasta ja sitä käytetään muun muassa SQL kyselykielessä relaatiotietokantojen kanssa [26]. React siis vastaa näkymän päivittämisestä aina kun komponentin data muuttuu [16]. Kehittäjän vastuulle jää pelkästään näkymien toteuttaminen, kun React hoitaa näkymien ylläpidon. Deklaratiivinen ohjelmointi malli tekee koodista suorituksesta ennalta arvattavampaa ja helpottaa koodin debuggausta [16].

React arkkitehtuuri on suunniteltu komponenttipohjaiseksi. Käyttöliittymäkomponentit ovat uudelleenkäytettäviä ja ne vastaavat omasta tilanhallinnasta deklaratiivisen ohjelmointimallin mukaan [16]. Komponenttien tulee sijaita SRC nimisessä kansiossa ja seurata Pascal Case nimeämiskonventiota [29].

JSX on JavaScript kielen laajennos. Se muistuttaa HTML:ää syntaksiltaan ja se on luotu helpottaakseen käyttöliittymä komponenttien tekoa, vaikka myös pelkkä JavaScriptin käyttö on sallittua. [29]



Kuva 1. *Sama komponentti toteutettu JSX:llä ja ilman [16].*

Reactiin lisättiin versiossa 16.8 Hook-niminen toiminnallisuus [15]. Hook mahdollistaa eri React-toiminnallisuuksien käytön ilman tarvetta toteuttaa niille omaa kokonaista luokkaa [15]. Hook mahdollistaa muun muassa tilan hallinnan ja elinkaarimetodien käytön [23].

Hook-toiminnallisuuden käytössä on kolme tärkeää sääntöä, jotka tulee ottaa huomioon. Niitä voidaan kutsua vain komponentin ylimmältä tasolta, joka on myös React funktio komponentti. Hook ei voi olla ehdollinen eikä sitä voi käyttää React luokka komponentissa. [23]

```
import React, { useState } from "react";
import ReactDOM from "react-dom";

function FavoriteColor() {
  const [color, setColor] = useState("red");

  return (
    <>
      <h1>My favorite color is {color}!</h1>
      <button
        type="button"
        onClick={() => setColor("blue")}
      >Blue</button>
      <button
        type="button"
        onClick={() => setColor("red")}
      >Red</button>
      <button
        type="button"
        onClick={() => setColor("pink")}
      >Pink</button>
      <button
        type="button"
        onClick={() => setColor("green")}
      >Green</button>
    </>
  );
}

ReactDOM.render(<FavoriteColor />, document.getElementById('root'));
```

Kuva 2. *Esimerkki React Hook toiminnallisuudesta [23].*

Esimerkissä on havainnollistettu React Hook toiminnallisuutta lempivärin avulla. Ensiksi useState niminen Hook on tuotu React kirjastosta projektiin. UseState Hook mahdollistaa funktio komponentin tilan seurannan. Tässä esimerkissä tila on alustettu ”red” parametrilla. Funktio komponentti sisältää kaksi parametria eli tässä tapauksessa ensimmäinen parametri color sisältää nykyisen tilan ja toinen parametri setColor funktio päivittää tilan. Kun tilaa halutaan päivittää, kutsutaan setColor

funktiota halutulla parametrilla. Tilan suora päivittäminen color parametria muuttamalla ei ole sallittua vaan se on tehtävä aina funktion kautta. Parametrin näyttäminen tapahtuu lukemalla {color} muuttuja.

3.2 JavaScript

JavaScript on vuonna 1995 julkaistu verkkosivuille suunnattu dynaaminen ohjelmointikieli. JavaScript on nykypäivänä käytännössä lähes ainoa ohjelmointikieli, jolla dynaamista sisältöä voidaan verkkosivulle lisätä Flashin ja Java Applettien poistuttua laajasti käytöstä niiden tietoturva ja suorituskyky ongelmista johtuen. JavaScript on tulkattava, dynaamisesti tyypitetty oliopohjainen ohjelmointikieli ja sen syntaksi on saanut vaikutteita C-ohjelmointikielestä. Kaikilla merkittävillä selaimilla on käytössä jokin JavaScript-moottori, jolla suoritetaan JavaScript koodia. [27]

JavaScript-ohjelmointikielen päälle on rakennettu lukuisia erilaisia kirjastoja kuten tässä työssä käytetty React Native ja sen käyttö on kasvanut valtavasti viime vuosikymmeninä. Node.js palvelinajoympäristö, TypeScript vahva tyypitys ja npm pakетinhallintajärjestelmä ovat modernisoineet JavaScriptin käytön osana nykypäivän sovelluskehitys työkalupakkia. Ilman muun muassa edellä mainittuja työkaluja JavaScritpin käyttö yksinään on haastavaa, jos tavoitteena on skaalautuva, moderni ja vikasietoinen järjestelmä.

3.3 TypeScript

TypeScript on Microsoftin kehittämä ohjelmointikieli, joka julkaistiin ensimmäisen kerran vuonna 2012 [30]. TypeScript lisää vahvan tyypityksen tuen muuten heikosti tyypitettyyn JavaScript kieleen [20]. Vahvasti tyypitetty ohjelmointikieli eroaa heikosti tyypitetystä kielestä siten, että sen muuttujien tyypit tulee määritellä eksplisiittisesti jo luotaessa toisin kuin heikosti tyypitetyssä kielessä tätä määritystä ei tarvitse tehdä. Heikosti tyypitetyt kielet ajautuvatkin helposti ajon aikaisiin ongelmiin, kun muuttujan tyyppi on jotain muuta, kun mitä alun perin oletettiin [20]. Tämä johtaa helposti ohjelman ajon aikaisiin kaatumisiin.

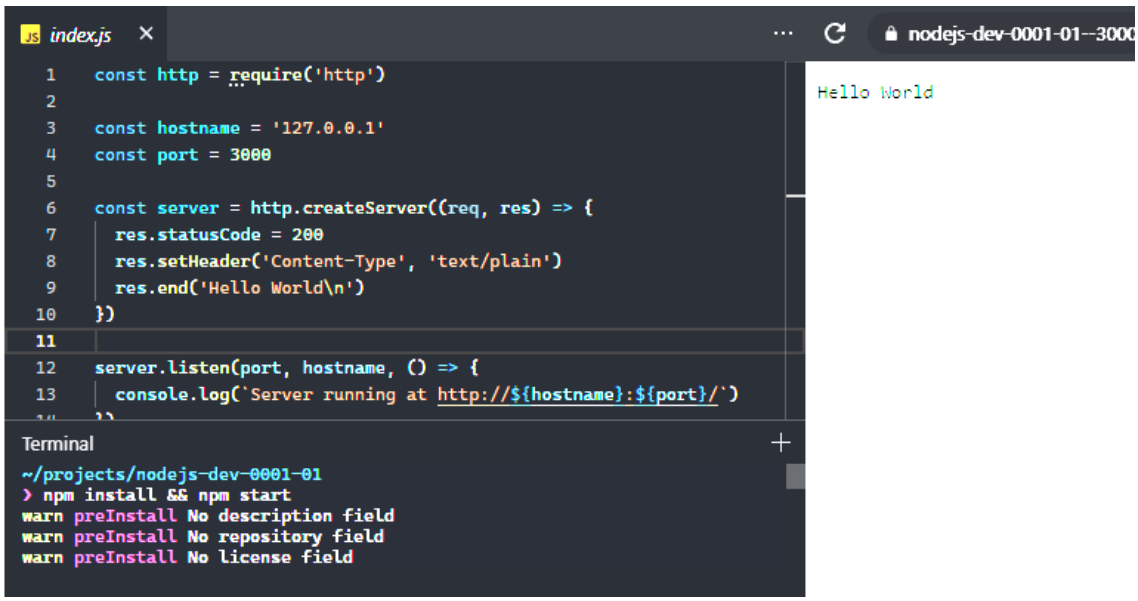
TypeScript on niin sanottu ”superset” JavaScript kielestä eli jo olemassa olevat JavaScript ohjelmat ovat myös valideja TypeScript ohjelmia. Tämä helpottaa huomattavasti sen käyttöönottoa, kun se voidaan ottaa osittain osaksi vanhaa JavaScript projektia rikkomatta aikaisempaa JavaScript toteutusta. [20]

Kuva 3. *Havainnollistava esimerkki TypeScriptin ja JavaScriptin eroista [21].*

3.4 Node.js

Node.js on ilmainen avoimen lähdekoodin alustariippumaton JavaScript koodin palvelinajoympäristö [22]. Node.js on rakennettu Chromen V8 JavaScript moottorin päälle [9].

Node.js sovellusta ajetaan yksittäisessä prosessissa ilman että uutta säiettä luodaan erikseen jokaiselle kutsulle. Node.js arkkitehtuuri on suunniteltu tapahtuma pohjaiseksi ja se toimii asynkronisesti käsitellessä I/O kutsuja. Kun Node.js suorittaa I/O operaation kuten lukutapahtuman tietokannasta sen sijaan että Node.js lukitsisi säikeen ja tuhlaisi suorittimen syklejä Node.js palaa operaatioon, kun se saa vastauksen takaisin. Tämä mahdollistaa tuhansien samanaikaisten yhteyksien käsittelyn yhdellä palvelimella ilman usean säikeen hallinnan tuomaa painolastia. [10]



Kuva 4. *Node.js Hello World esimerkki [10].*

3.5 Npm paketinhallintajärjestelmä

Npm on maailman suurin paketinhallintajärjestelmä ja se on tässäkin työssä käytössä [11]. Npm:llä on yli 11 miljoonaa käyttäjää ja sen käyttö on ilmaista [11]. Npm rekisteri sisältää yli 800 000 koodipakettia ja se on laajalti käytössä avoimen lähdekoodin projekteissa, mutta sitä käytetään myös yksityisissä projekteissa [24].

```
C:\>npm install <package>
```

Kuva 5. *Npm paketin asentaminen CLI:n kautta [24].*

Npm:n käyttö vaatii Node.js:n asentamisen, jonka mukana se tulee. Npm on alun perin luotu Node.js:n paketinhallintajärjestelmäksi. Npm paketit on määritelty package.json tiedostoon, joka on JSON formaattia. [11]

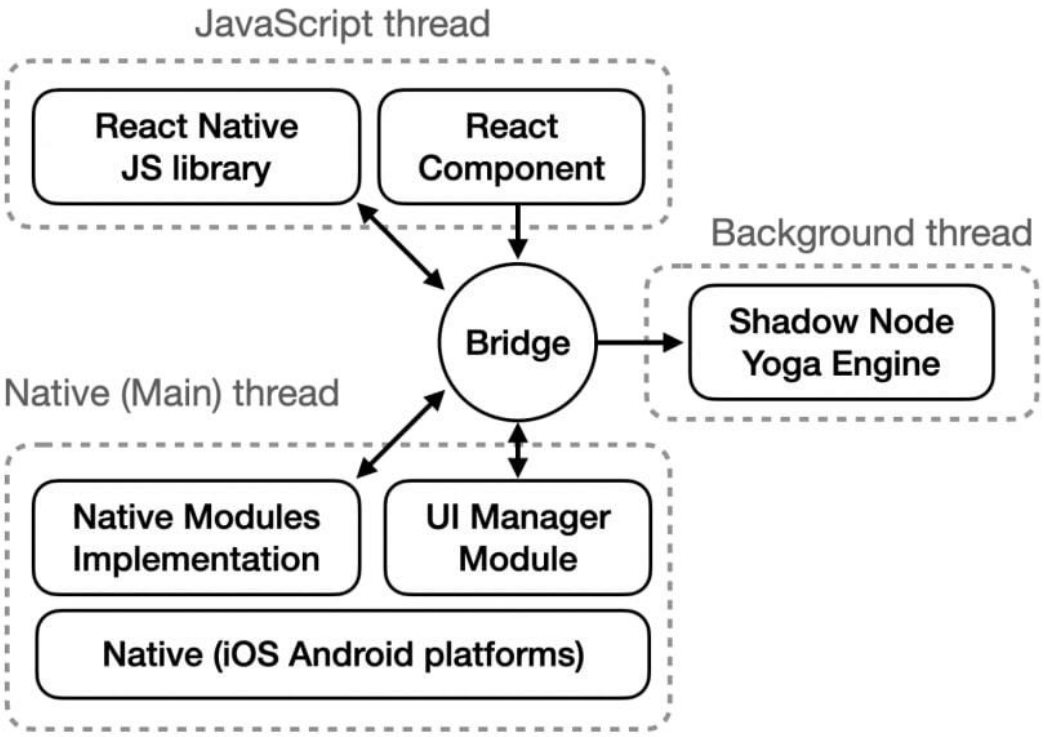
```
{
"name" : "foo",
"version" : "1.2.3",
"description" : "A package for fooing things",
"main" : "foo.js",
"keywords" : ["foo", "fool", "foolish"],
"author" : "John Doe",
"licence" : "ISC"
}
```

Kuva 6. *Esimerkki npm paketinhallintajärjestelmän versiohallinnasta [24].*

3.6 React Native

React Native on Metan eli entisen Facebookin kehittämä avoimen lähdekoodin käyttöliittymäkirjasto. Sillä pystyy kehittämään alustariippumattomasti sovelluksia muun muassa Android, iOS ja Web alustoille [13]. Sovelluskehyskielenä toimii JavaScript niin kuin Reactissa. Näkymän eri komponentit kuitenkin renderöidään kyseisen alustan natiivikomponentteina. Näin sama sovellus saadaan räätälöityä eri alustoille ikään kuin se olisi alun perin kehitetty kyseisen käyttöjärjestelmän natiiviohjelmointikielellä [13]. Verrattuna erilaisiin hybridiratkaisuihin, joissa alustariippumattomuus on toteutettu HTML5 pohjaisena toteutuksena, kuten PhoneGap tai Io-

nic on React Nativen kaltaisen ratkaisun etuna sen parempi käyttäjäkokemus johtuen sen paremmasta suorituskyvystä ja käyttäjälle jo entuudestaan tutuista natiivikomponenteista [18].



Kuva 7. *React Nativen arkkitehtuurikuvaus. [5]*

React Nativen kehitysperiaatteet ovat hyvin lähellä perus React versiota. Isoimpana erona on, että React Native ei manipuloi (DOM) kerrosta virtuaali DOMin kautta kuten React tekee. React Nativen toiminta on jaettu kolmeen suoritussäikeeseen. JavaScript säikeessä sijaitsee React komponentit ja React Nativen JavaScript kirjasto. Päähaarassa sijaitsee natiivimoduuleiden implementaatio ja niiden toiminnanohjaus. Taustaprosessina toimii Shadow Node Yoga Engine, mikä tulkitsee kehittäjien kirjoittamaa JavaScript koodia. Nämä kolme säiettä kommunikoivat keskenään niin kutsutun sillan välityksellä. Silta lähettää serialisoituja viestejä eri säikeille asynkronisesti, jotta säikeet eivät jäisi odottamaan toisiaan täten hidastaen ohjelman suoritusta. [5]

3.7 React Native kirjallisuudessa

React Nativesta on toteutettu lukuisia eri tutkielmia. Monika Boharan vuonna 2020 tekemässä kirjallisuustutkielmassa [7] verrattiin React Nativea natiiveihin Android ja

iOS sovellus alustoihin. Tutkielmassa todettiin, että React Nativen raakasuorituskyky häviää hieman natiivi sovelluksiin verrattuna, mikä on ymmärrettävää ylimääräisestä sovellustasosta johtuen. React Nativella toteutettu sovellus vei hieman enemmän RAM muistia, käytti enemmän prosessorin tehoja ja kulutti enemmän tallennustilaa kovalevyltä verrattuna aitoon natiivisovellukseen.

Tutkielmassa todettiin kuitenkin, että käyttökokemus vastasi hyvin natiivia sovellusta. Käyttäjille tehtiin kysely käyttökokemukseen liittyen natiivi sovelluksen ja React Nativella tehdyn sovelluksen välillä. Siinä mitattiin sovelluksen houkuttelevuutta, ymmärrettävyyttä, tehokkuutta, luotettavuutta, stimulaatiota ja uutuudenviehätystä. Kyselyssä huomattiin, että käyttökokemus oli hyvin lähellä natiivia sovellusta, ilman suurempi poikkeavuuksia Tutkielmassa todettiinkin, että React Nativen vahvuudet ovatkin enemmän kehityspuolella. React Nativen käyttäminen mahdollistaa alustariippumattoman kehityksen, yhdellä samalla ohjelmointikielellä. Tämä säästää organisaation resursseja ohjelmistokehityksessä ja helpottaa usean eri alustan haltuunottoa saman aikaisesti.

4. TOIMINTAYMPÄRISTÖ JA VAATIMUKSET

Tässä kappaleessa tarkastellaan toimintaympäristöä ja mitä vaatimuksia se asettaa sovellukselle ja React Native-sovelluskehykselle. Sovelluksen on tarkoitus toimia auton itsepesuympäristössä. Ympäristönä pesupiste on kiireinen, ruuhkainen ja sotkuinen. Pesupaikka voi olla joko ulkona tai sisällä eli varautuminen eri sää tiloihin on tärkeää. Tämän takia maksupäätelaite on sään kestävä. Sovellukselle tärkeitä ominaisuuksia on sen helppokäyttöisyys, vikasietoisuus ja toimintavarmuus. Hyvän käyttökokemuksen takaamiseksi on tärkeää, että sovelluksessa on iso ja selkeä käyttöliittymä.

4.1 Arkkitehtuurin tavoitteet ja vaatimukset

Tutkimuksen kohteena olevalle arkkitehtuurille asetetaan tietyt reunaehdot, jotta sen onnistumista pystyttäisiin objektiivisesti tarkastelemaan. Vaatimuksilla ja tavoitteilla halutaan varmistaa, että lopputuote olisi mahdollisimman lähellä suunniteltua lopputulosta ja siitä olisi aitoa hyötyä myös yritykselle. Arkkitehtuurin päätavoite on tukea alustariippumattomasti Android ja iOS alustaa ja olla tarpeeksi monipuolinen, että sitä on mahdollista jatkokehittää ketterien ohjelmistoperiaatteiden mukaisesti. Alle on listattu arkkitehtuurin eri tavoitteet ja vaatimukset.

Taulukko 1. *Arkkitehtuurin vaatimukset listattuna*

Arkkitehtuurille asetetut vaatimukset
1. **Arkkitehtuurin tulee tukea alustariippumatotonta kehitystä sekä Android että iOS alustoilla.**
2. **Arkkitehtuuria pitää olla mahdollista laajentaa tulevaisuudessa esimerkiksi lisäämällä se osaksi toista sovellusta verrattain pienellä työmäärällä.**
3. **Käyttöliittymän elementtien tulee huomioida sovelluksen laaja käyttäjäkunta mahdollistaakseen sujuvan käyttökokemuksen.**
4. **Arkkitehtuurin tulee olla moderni, vikasietoinen ja tietoturvallinen.**

Alkuvaiheessa tuki Android laitteelle on oleellisin ja työ keskittyy siihen, koska sovellus asennetaan Android-pohjaiselle maksupäätelaitteelle. Tulevaisuudessa kuitenkin sovelluksen käyttöä halutaan laajentaa myös iOS alustalle. Tästä syystä

alustariippumaton toteutus heti alusta asti on paras lähestymistapa projektille. Näin taataan, että samaa koodia pystytään uudelleen käyttää myös myöhemmin. Alustariippumattomuudesta on lukuisia hyötyjä organisaatiolle. Sillä pienennetään kuluja ja selvitään pienemmällä määrällä kehitysresursseja, kun sovellusta ei tarvitse kehittää kahdelle eri ympäristölle eri ohjelmointikielellä. Hyödyt jatkuvat myös tulevaisuuteen, kun sovellusta tullaan päivittämään ja jatkokehittämään.

Arkkitehtuuri on tarkoitus pitää tarpeeksi joustavana, jotta sen käyttöä on mahdollista laajentaa tulevaisuudessa ilman kovin suurta työmäärää. Tämä halutaan varmistaa käyttämällä ohjelmistoalan hyväksi havaittuja standardeja ja teknologioita kuten esimerkiksi TypeScriptiä.

Sovelluksella tulee oletettavasti olemaan laaja käyttäjäkunta ja tämä tulee huomioida käyttöliittymän suunnitteluvaiheessa, jotta käyttökokemus pysyisi mahdollisimman hyvänä kaikille käyttäjille. Tätä varten käyttöliittymän tulee olla minimalistinen ja käytettyjen komponenttien selkeitä ja tarpeeksi suurikokoisia. Arkkitehtuurin tietoturva ja vikasietoisuus tullaan pääsääntöisesti takaamaan käyttämällä ajan tasalla olevia teknisiä ratkaisuja, ohjelmistoviitekehyksiä ja alalla aiemmin todettuja hyviä käytäntöjä. Arkkitehtuurin modernisuus halutaan varmistaa käyttämällä uusia teknologioita kuten React Hookia, Flexboxia ja TypeScriptiä.

4.2 Toiminnalliset vaatimukset

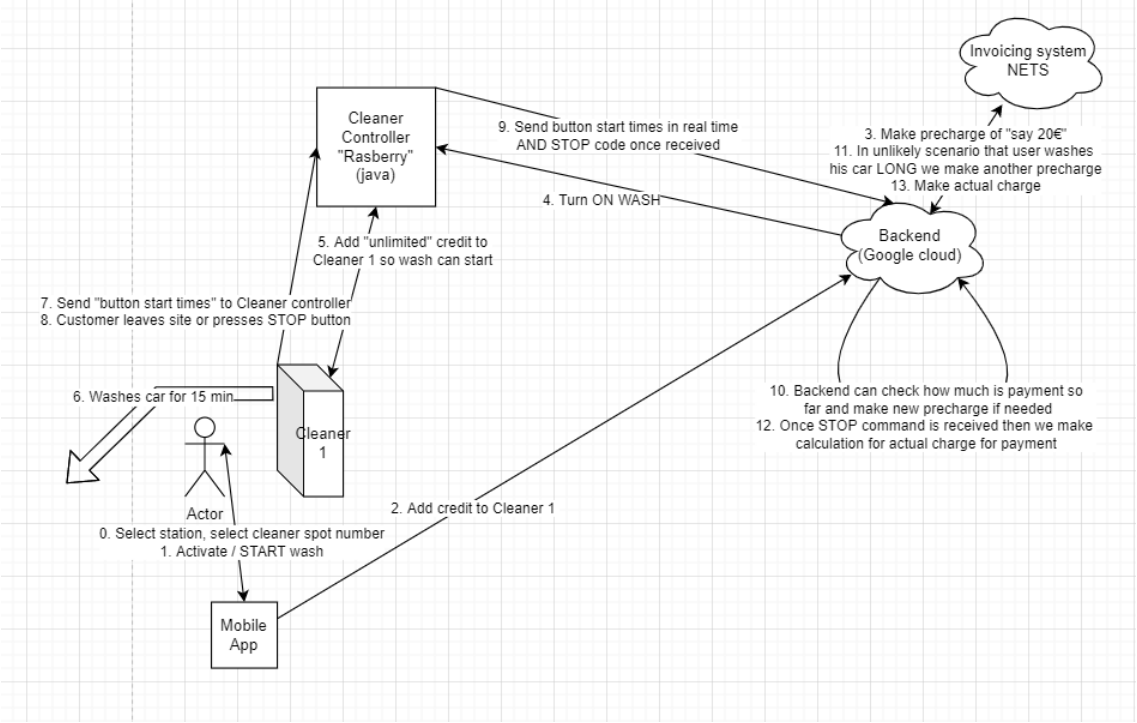
Sovellukselle asetetaan myös sen vaatimat toiminnalliset vaatimukset. Esitetyt vaatimukset perustuvat sovelluksen käytännön vaatimiin tarpeisiin, eli maksutapahtuman suorittamiseen. Kommunikointi taustajärjestelmän kanssa tapahtuu REST-rajapinnan välityksellä ja sovelluksen tulee käsitellä siltä saatuja JSON-objekteja.

Taulukko 2. *Toiminnalliset vaatimukset listattuna*

Toiminnalliset vaatimukset
1. **Sovelluksen tulee pystyä muodostamaan yhteys taustajärjestelmään REST-rajapinnan välityksellä.**
2. **Sovelluksen tulee pystyä käsittelemään taustajärjestelmältä saatuja JSON-dataobjekteja.**
3. **Sovelluksen tulee pystyä lähettämään katevaraus taustajärjestelmään.**

Sovelluksen on tarkoitus tulevaisuudessa saada taustajärjestelmältä muun muassa asemakohtaisia hintatietoja REST-rajapinnan kautta. Maksutapahtuman on myös tarkoitus tapahtua REST-rajapinnan yli hyödyntäen NETS katevaraus järjestelmää.

Taustajärjestelmän toiminnallisuuden toteutus on kuitenkin rajattu tämän diplomityön ulkopuolelle ja sen toteutus tapahtuu työn toimeksiantajan puolelta.



Kuva 8. *Alustava yleiskuva pesutapahtuman kokonaisarkkitehtuurista.*

4.3 Maksupäätelaite

Yksittäiselle pesupisteelle on tarkoitus asentaa Android pohjainen Saturn 1000-E UPT niminen maksupäätelaite, johon React Native sovellus voidaan asentaa alkuvaiheessa. Maksutapahtumasovellus on myöhemmin tarkoitus lisätä erilliseen sovellukseen, jonka käyttäjä voi myös ladata omaan puhelimeen.

Kuva 2. *Saturn 1000-E UPT. [4]*

Maksupäätelaite on sään kestävä, jotta se voidaan asettaa ulos pesupisteen läheisyyteen. Laitteessa on 4G LTE, Ethernet ja Wi-Fi valmius maksutapahtuman suorittamista varten. Maksamisen on mahdollista lähimaksun avulla. Laite tulee viiden tuuman värinäytöllä ja 1280 x 720 pikselin tarkkuudella. Laitteen tarkemmat spesifikaatiot löytyvät alla olevasta taulukosta.

Taulukko 3. *Saturn 1000-E UPT spesifikaatio*

Prosessori	High Speed Quad-Core Processor plus Secure MCU
Muisti	8GB/1GB (Up to 16GB/2GB)
Käyttöjärjestelmä	Android v7.x
Näyttö	5” Color TFT LCD with Capacitive Touch Panel | 1280 x 720 Pixels
Salaus	TRNG | DES | T’DES | AES | RSA | MK/SK | DUKPT
Paino	600 g
Mitat	191.8 x 110 x 65.6 mm
IP-luokitus	IP 54
Yhteydet	4G LTE | Wi-Fi | Bluetooth | Ethernet
Kortinlukija	ISO 14443 Type A, B | ISO 18092 NFC | Mifare and Sony FeliCa Working Frequency 13.56MHz

5. RATKAISUEHDOTUS KONSEPTITOTEUTUKSELLE

Tässä luvussa käydään läpi tutkimusta varten toteutettava prototyyppisovellus. Prototyyppisovelluksen avulla on tarkoitus löytää vastaukset aikaisemmin esitetyille tutkimuskysymyksille. Sovelluksella voidaan suorittaa maksu itsepesupalvelussa, mutta tämä lopputyö keskittyy itse React Nativen tarkasteluun prototyyppisovelluksen avulla.

Sovelluksen on tarkoitus toimia alkuvaiheessa aiemmin esitetyssä Android pohjaisessa maksupäätelaitteessa, mutta sen käyttöä on mahdollista laajentaa tulevaisuudessa asiakkaiden omiin Android ja iOS pohjaisiin mobiililaitteisiin. Sovellus toimii yritykselle auton itsepesupalvelun konseptitoteutuksena, jonka arkkitehtuuri mahdollistaa on hyvät laajentamis-mahdollisuudet tulevaisuudessa.

5.1 Toteutus ja tiedostorakenne

Projektin alustamista varten tuli ensiksi asentaa React Nativen tarvitsemat riippuvuudet. React Native kehitysympäristö vaatii toimiakseen Node.js web-palvelimen ajoympäristökseen. Android tukea varten React Native vaatii myös JDK:n eli Java Development Kit ja Android Studio riippuvuudet. Lopuksi tarvitaan React Native komentokehote tulkki ja Python 2 riippuvuudet. Chocolatey nimisellä paketinhallinta työkalulla voidaan kätevästi asentaa Node.js Python 2 ja JDK. Node.js:n mukana tulee npm paketinhallinta työkalu, minkä avulla voidaan asentaa React Native komentokehote tulkki.

```
choco install -y nodejs.install python2 jdk8
```

```
npm install -g react-native-cli
```

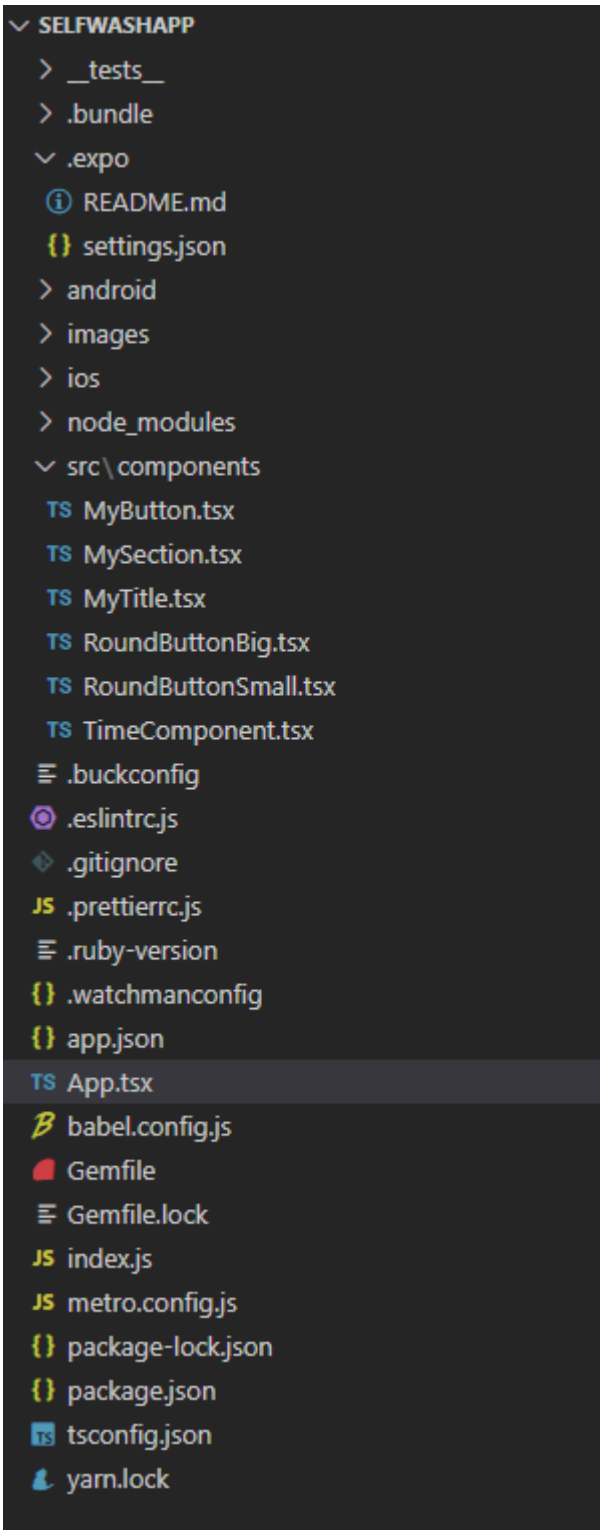
Kuva 9. *Käytetyt komennot kehitysympäristön pystyttämistä varten.*

Tämän jälkeen tulee vielä asentaa Android Studio ja Android SDK Android laitteen emuloimista ja projektin kääntämistä varten. Lopuksi uusi projekti voidaan alustaa React Native init komennolla. Android Studion avulla voidaan emuloida fyysistä laitetta sovelluksen testaamista varten.

```
react-native init AwesomeProject
```

Kuva 10. *Esimerkki uuden projektin alustamisesta.*

Tutkimusta varten toteutettava konseptitoteutus toteutettiin käyttämällä TypeScript ohjelmointikieltä. TypeScript mahdollistaa modernimman ja vikasietoisemman arkkitehtuurin rakentamisen verrattuna pelkkään JavaScript toteutukseen. TypeScript mahdollistaa muuttujien tyyppitarkistuksen ja parantaa täten koodin ylläpidettävyyttä ja vikasietoisuutta. TypeScript on yksi osa, jolla vastataan vaatimusten 4. kohtaan.



Kuva 11. *Sovelluksen tiedostorakenne.*

Yllä olevassa kuvassa on avattu projektin tiedostorakenne sen havainnollistamista varten. Eri alustoille spesifit riippuvuudet on eritelty omiin kansioihin Androidille ja iOS:lle. Images kansioon tallennettu sovelluksessa käytetyt yhden ja kahden euron kuvakkeet. Components kansiossa on sovelluksen käyttöliittymäkomponentit. Package.json tiedostossa on listattu npm-paketinhallintajärjestelmän riippuvuudet.

Työssä on käytössä ESLint kirjasto, joka on staattinen koodin analysaattori. Se löytää ja korjaa automaattisesti virheitä koodin syntaksista ja täten osaltaan parantaa koodin vikasietoisuutta.

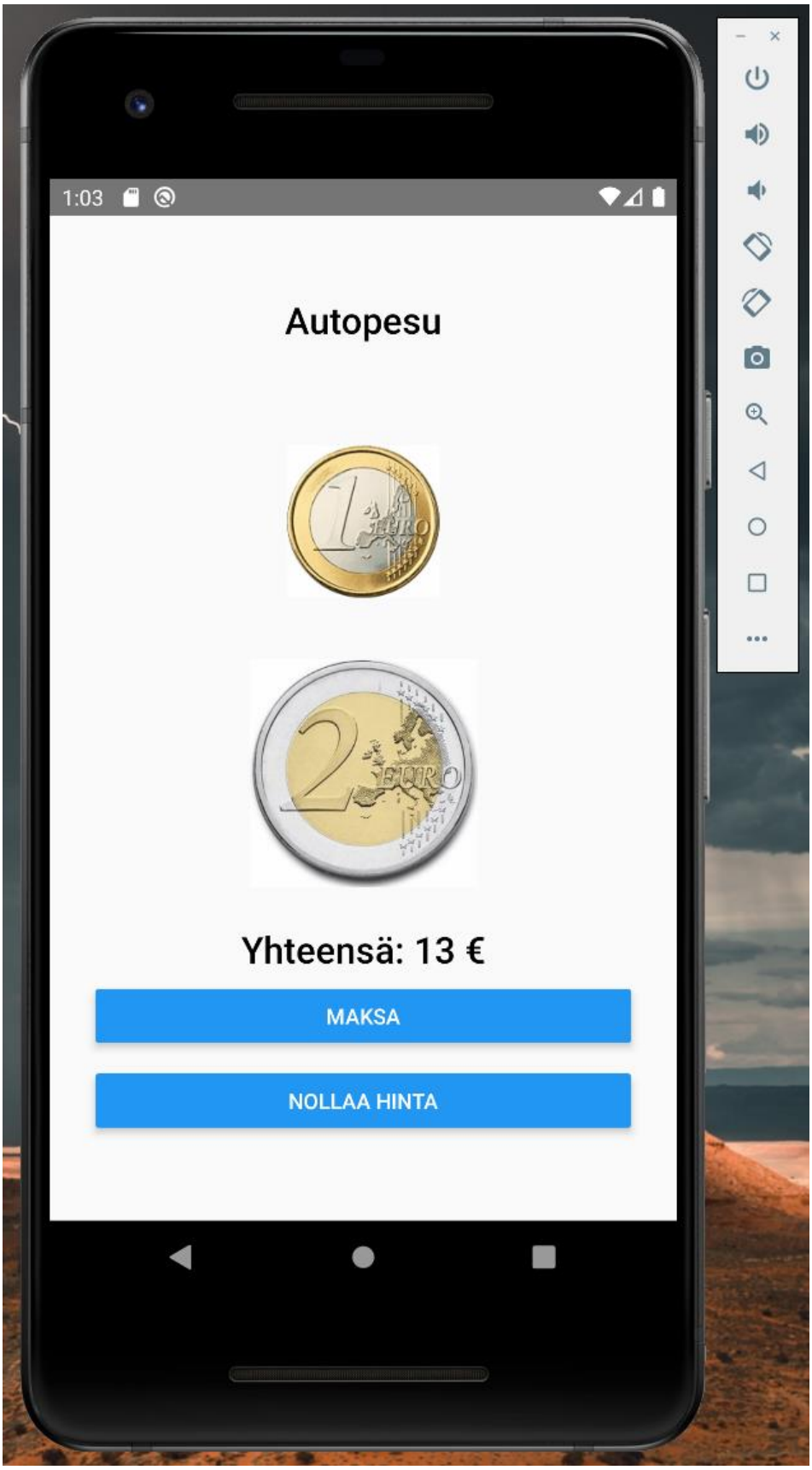
Components kansiossa on eriteltynä sovelluksen käyttöliittymäkomponentit. Sovelluksessa ei ole yhtä erillistä kansiota tai tiedostoa CSS tyylejä varten. Tavallisessa web-projektissa yleensä on yksi keskitetty tiedosto CSS tyylejä varten, mutta tässä työssä käytetyssä React toteutusmallissa CSS tyylit ovat eroteltu suoraan käyttöliittymäkomponentien alaisuuteen. Kun tyylit löytyvät suoraan käyttöliittymäkomponentti luokasta on kehittäjälle huomattavasti helpompaa palata niihin mahdollisia muutoksia varten.

5.2 Käyttöliittymä

Sovelluksen käyttöliittymä päädyttiin tekemään pajatsomallin mukaan. Asiakas pystyy lisäämään rahaa kokonaissummaan painamalla halutun kolikon kuvaa. Kokonaissumma näytetään näytön alareunassa. Pajatsomalliin oli alun perin tarkoitus lisätä niin sanottu minuuttihinta pesulle, josta voitaisiin laskea kuinka paljon pesuaikaa kyseisellä summalla saa. Pesuaika päivittyisi aina siten reaaliajassa, kun asiakas lisää rahaa pesua varten. Tätä toimintoa ei pystytty kuitenkaan toteuttamaan toistaiseksi johtuen siitä, että minuuttihinnat ovat erilaiset eri pesutoiminnoille kuten esimerkiksi vesi verrattuna pesuaineeseen ja nämä hinnat ovat vielä asemakohtaisia. Jatkokehityksessä asemakohtaiset hinnat voitaisiin lisätä mahdollisesti sovellukseen REST-rajapinnan kautta. Alkuvaiheessa eri pesuvaihtoehtojen hintojen on tarkoitus näkyä asemalle erikseen pystytettävässä fyysisessä taulussa.

Sovelluksen helppokäyttöisyys haluttiin varmistaa pitämällä käyttöliittymä mahdollisimman minimalistisena hyvän käyttökokemuksen takaamiseksi. React Native tarjoaa työkaluja saavutettavuuden huomioon ottamisessa. Sovelluskehyksen natiivielementteihin on implementoitu valmiit saavutettavuus ominaisuudet, joiden käyttöönotto on helppoa [12]. Käyttökokemus on hyvin subjektiivinen kriteeri ja sitä on vaikea mitata tarkasti, koska se vaihtelee aina käyttäjäkohtaisesti. Käyttöliittymä saatiin pidettyä yhdessä näkymässä ja painikkeiden suuri koko oli suuri prioriteetti sujuvan käyttäjäkokemuksen varmistamiseksi, jotta virhepainalluksilta vältyttäisiin. Lisäksi käyttöliittymässä on napit maksun hyväksymistä varten ja hinnan korjaamista varten.

Toisena vaihtoehtona käyttöliittymälle pohdittiin pelkää numerokenttää summan syöttämiseksi. Pajatsomalli koettiin kuitenkin intuitiivisempana ratkaisuna käyttäjäkokemuksen kannalta. Numerokenttä ei tukisi minuuttihinta logiikka läheskään yhtä hyvin. Pajatsomallissa asiakas näkisi pesunajan päivittymisen reaaliajassa ja sitä olisi helpompi lisätä aina tarpeen mukaan.



Kuva 12. *Sovelluksen käyttöliittymä emuloituna Android studion avulla.*

5.3 Maksulogiikan hallinnan toteutus React Hookin avulla

Maksutapahtuman hallinta toteutettiin käyttämällä React Hookia. Hook teknologia mahdollistaa erilaisten React ominaisuuksien käytön kuten funktiokomponentin tilan hallinnan. Hook:lla pystytään korvaamaan erilliset luokkakomponentit, täten vähentäen ylimääräistä boiler plate koodia ja parantaen koodin ylläpidettävyyttä.

```
const [total, setTotal] = useState(0);

function addNumber(a: number) {
  const newTotal = total + a;
  setTotal(newTotal);
}

function reset() {
  const newTotal = 0;
  setTotal(newTotal)
}
```

Kuva 13. *Pesuhinnan seuranta Hook:n avulla.*

Komponentin tila alustetaan kutsumalla useStatea funktiokomponentissa. Maksutapahtumaa varten tila alustetaan 0 arvolla, joka on tässä tapauksessa viittaa maksutapahtuman alkusummaan. Tilaa pystytään muuttamaan setTotal funktiolla, kun halutaan vaihtaa maksusummaa. AddNumber funktiossa lisätään parametrina annettu summa pesutapahtuman kokonaissummaan setTotal funktiolla. Reset funktiota kutsumalla voidaan nollata kokonaissumma. Toteutus soveltuu hyvin maksutapahtuman vaatimaan dynaamiseen tilanhallintaan. Kun maksutapahtuman tilanhallinta on keskitetty sovelluksen pääluokkaan, on sitä helppo hallinnoida sovelluksen sisäisesti. Käyttöliittymäkomponentit voivat vain lukea maksutapahtuman arvon props parametrista, mutta eivät voi muuttaa sitä.

React Hookin käyttö auttaa vastaamaan arkkitehtuurille asetettujen vaatimusten kohtiin 2. ja 4. React Hook on osana vastaamaan modernin ja helposti laajennettavan arkkitehtuurin vaatimuksiin. React Hook mahdollistaa maksudatan dynaamisen tilan helpon hallinnan suoraan ohjelman pääluokassa ja sen tilan jakamisen tarvittaville käyttöliittymäkomponenteille React props parametrin avulla.

5.4 Komponenttien modulaarisuus

Sovelluksen käyttöliittymäkomponentit ovat suunniteltu modulaarisen arkkitehtuurin menetelmin. Komponentit ovat tilattomia, funktionaalisia ja niitä on mahdollista

käyttää uudestaan. Täten halutaan varmistaa sovelluksen käyttöliittymäkomponentien helppo räätälöintimahdollisuus ja sovelluksen laajennettavuus tulevaisuudessa, jotka ovat listattu vaatimuksien kohdissa 3 ja 4.

```
import React from 'react';
import { StyleSheet, Text, View, useColorScheme} from 'react-native';
import { Colors } from 'react-native/Libraries/NewAppScreen';


export type Props = {
    title: string;
    total?: number;
  };


  const MySection: React.FC<Props> = ({
    title,
    total = 0
  }) => {

    const isDarkMode = useColorScheme() === 'dark';

    const backgroundStyle = {
      backgroundColor: isDarkMode ? Colors.darker : Colors.lighter,
    };

  return (
    <View style={styles.sectionContainer}>
      <Text
        style={[
          styles.sectionTitle,
          {
            color: isDarkMode ? Colors.white : Colors.black,
          },
        ]}>
        {title}: {total} €
      </Text>
    </View>
    );
};

const styles = StyleSheet.create({
    sectionContainer: {
      marginTop: 32,
      paddingHorizontal: 24,
    },
    sectionTitle: {
      fontSize: 24,
      fontWeight: '600',
```

```
    },
    sectionDescription: {
      marginTop: 8,
      fontSize: 18,
      fontWeight: '400',
    },
    highlight: {
      fontWeight: '700',
    },
  });


export default MySection;
```

Kuva 14. *MySection.tsx käyttöliittymäkomponentin ohjelmakoodi*

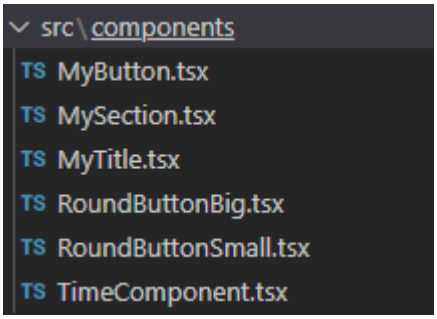
Kuvassa esitetään MySection nimisen käyttöliittymäkomponentin ohjelmakoodi. Sovelluksessa on muitakin custom käyttöliittymä komponentteja, mutta koska ne on toteutettu samalla modulaarisella arkkitehtuurityylillä niin en käy niitä kaikkia tässä erikseen läpi.

MySection komponenttia käytetään pesun kokonaissumman esittämiseen erillisessä tekstikentässä, johon on toteutettu custom CSS tyylit. Komponentti esittää dynaamisesti muuttuvaa pesun kokonaissummaa, käyttämällä Reactin props mallia. Komponentti ei itse pysty muuttamaan saatua props parametrin arvoa, vaan sitä käytetään pelkästään summan esittämiseen sovelluksessa. Kyseisen arkkitehtuurimallin idea on keskittää hinnan tilanhallinta yhteen paikkaan sovelluksessa.

MySection komponentille annetaan parametrit ”title” ja ”total”, joilla esitetään pesun hinta. Tiedoston alkuun määritellään mitä props-muuttujia komponentille tulee antaa, kun se alustetaan käyttöön sovelluksen puolella. TypeScriptiä käytettäessä tulee muuttujat alustaa halutulla tyypillä, jotka ovat tässä tapauksessa string ja number. Itse tarvittu data ja sen hallinta tapahtuu App.tsx luokassa. Komponentilla vain esitetään kyseinen data halutussa formaatissa.

Komponentin nimen täytyy alkaa isolla kirjaimella React konvetion mukaisesti. Seuraavaksi komponentin parametrit alustetaan ja rakennetaan itse haluttu käyttöliittymäkomponentti. Käyttämällä JSX:ää pystymme kirjoittamaan HTML:ää suoraan JavaScript tiedostoon. Luotu JSX-elementti palautetaan App.tsx luokalle esitettäväksi. Komponentin CSS-tyylit on koottu styles nimisen muuttujan alle, mikä helpottaa

huomattavasti käyttöliittymäkomponentien ylläpitoa tulevaisuudessa.



Kuva 15. *Käyttöliittymäkomponentit components kansiossa.*

5.5 Flexbox

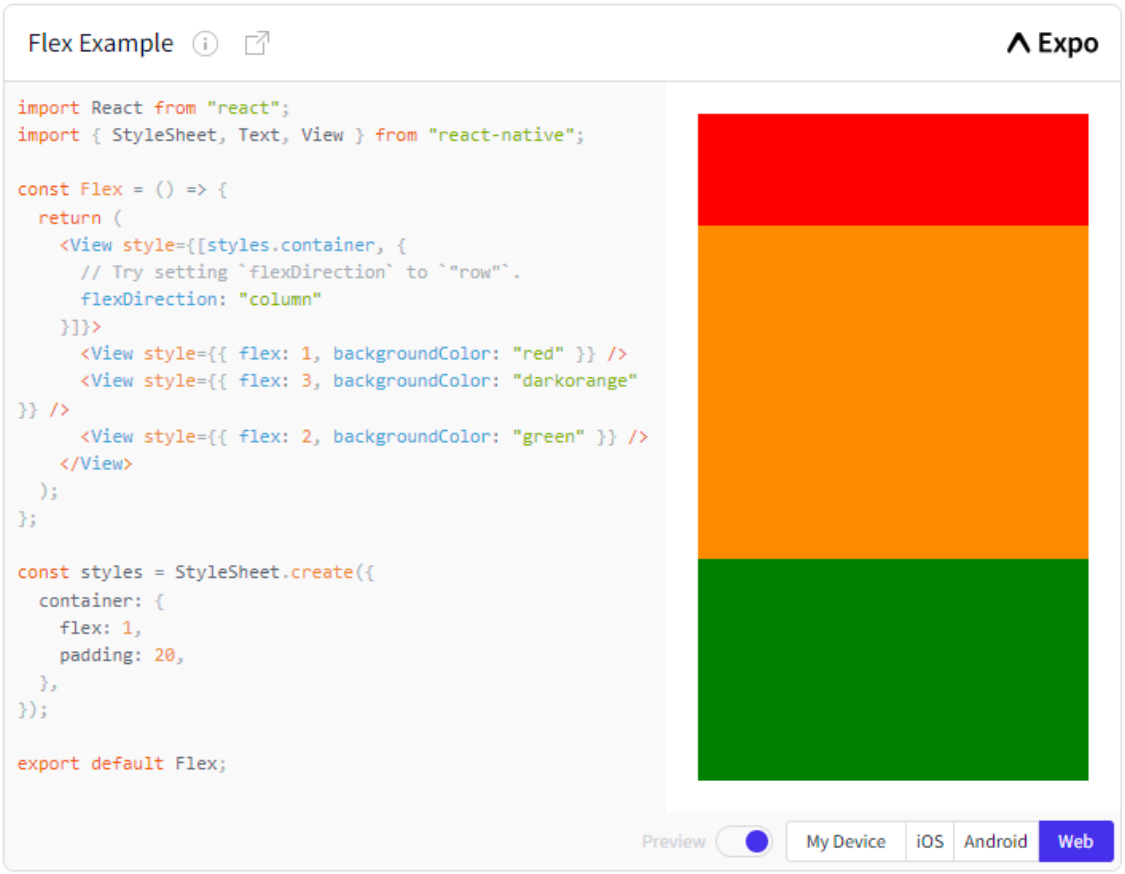
Sovelluksen käyttöliittymäkomponentien asettelu toteutettiin Flexboxin avulla. Flexbox on moderni tapa jaotella näkymä itsenäisesti hallittuihin lohkoihin. Flexboxin algoritmi osaa responsiivisesti tehdä jaon riippumatta minkä kokoista näyttöä käytetään.

```
    <View style={[styles.container, {
      flexDirection: "column"
    }]}>

    <View style={{ flex: 1}} >
      <MyTitle title='Autopesu'/>
    </View>

    <View style={{ flex: 3}} >
      <RoundButtonSmall title='2€' clickFunction={() => addNumber(1)}/>
      <RoundButtonBig title='5€' clickFunction={() => addNumber(2)}/>
    </View>

    <View style={{ flex: 2}} >
      <MySection title='Yhteensä' total={total}/>
      <MyButton title='Maksa' clickFunction={() => Alert.alert('Purchase
')}/>
      <MyButton title='Nollaa hinta' clickFunction={() => reset()}/>
    </View>

    </View>
```

Kuva 16. *Sovelluksen käyttöliittymäkomponentien asettelu Flexboxin avulla.*

Sovelluksessa Flexbox jaettiin kuuteen osaan 1+3+2=6. Lohkot jaettiin vaakatasossa, eli flexDirection asetettiin ”column” muotoon. Flexbox jaettiin kolmeen osaan, joiden koot olivat 1/6, 3/6 ja 2/6 osaa näytöstä. Keskellä oleville painikkeille haluttiin varata isoin tila näytöltä. Sovelluksen yläosassa sijaitsevalle otsikolle riitti

pieni yhden kolumnin tila. Alaosassa sijaitseville komponenteille jäi loput jäljellä olevasta tilasta näytöllä eli 2/6. Alla olevassa kuvassa tätä jakoa havainnollistetaan vielä tarkemmin värikoodien avulla.



Kuva 17. *Havainnollistus Flexboxin toiminnallisuudesta [14].*

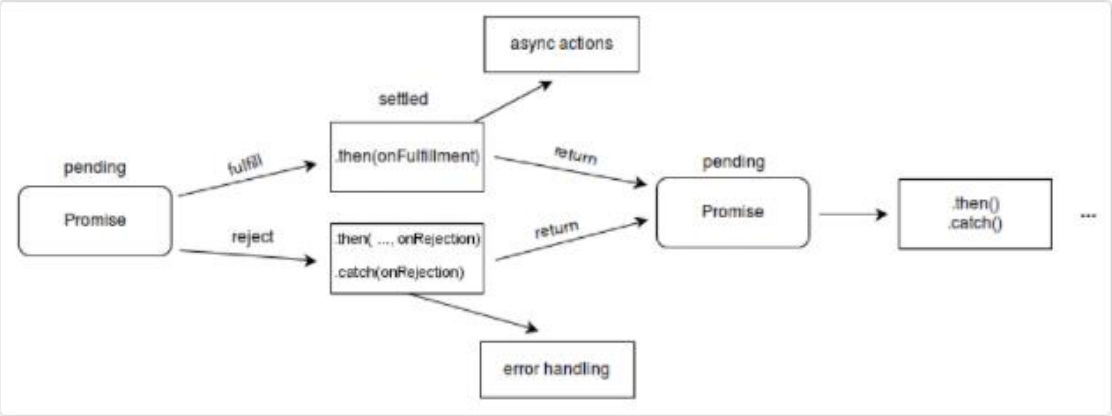
5.6 Rest-rajapinta tuki

Sovelluksen oli alun perin tarkoitus kommunikoida taustajärjestelmän kanssa REST-rajapinnan yli. Tätä rajapintaa ei kuitenkaan työnantajan puolelta ehditty toteuttamaan tämän työn aikataulussa. Työssä kuitenkin toteutettiin tuki REST-rajapinnalle, jotta laajentaminen taustajärjestelmän kanssa kommunikointiin olisi mahdollisimman helppoa tulevaisuudessa. Työhön suunniteltiin luurankometodi, joka voidaan myöhemmin ottaa käyttöön implementoimalla kyseiseen metodiin validin REST-rajapinnan parametrit.

React Native tarjoaa Fetch API rajapinnan HTTP-kutsuja varten, jota käytettiin tässä työssä. Fetch API rajapinnalla pystytään lataamaan sisältöä URL verkko-osoitteesta.

Fetch funktiolle siis annetaan haluttu URL, josta json data haetaan. Sovelluksen tapauksessa REST-kutsulla haettaisiin asemakohtaiset hinnat json datana. Paluuliikenteenä tehtäisiin katevaraus pesutapahtumaa varten taustajärjestelmään.

Työssä json datan lataus on toteutettu getPaymentData nimiseen funktioon. GetPaymentData on asynkroninen funktio, mikä on tyypillistä, kun ollaan tekemisissä verkkoliikenteen kanssa. Funktio toisin sanoen jää odottamaan vastausta palvelimelta niin kutsutun Promisen muodossa. Promisea käytetään asynkronisten kutsujen kanssa `f = () => expression` notaatiolla. Promiselle on määritelty kolme eri tilaa *pending, fulfilled ja rejected*.



Kuva 18. *Promise tilakaavio asynkronisille funktioille [8].*

UseEffect Hook kutsutaan sivunlatauksen yhteydessä, ja se kutsuu getPaymentData funktiota. Se vastaa React:sta tuttua componenDidMount metodia, mikä kertoo, että komponentti on alustettu oikein. Funktio on toteutettu try-catch lohkon sisälle, jotta mahdolliset virhetilanteet saadaan kiinni. Response muuttujaan haetaan fetch kutsulla json data. Await parametri kertoo funktiolle, että sen täytyy odottaa vastausta palvelimelta. SetData funktio asettaa vastauksena saadun json datan aiemmin määriteltyyn data nimiseen Hook muuttujaan. SetLoading Hook on aluksi asetettu tosi tilaan osoittamaan, että json datan lataus on vielä kesken. Latauksen lopuksi muuttuja asetetaan epätosi tilaan merkiksi siitä, että lataus on valmistunut.

```
const [isLoading, setLoading] = useState(true);
const [data, setData] = useState([]);


const getPaymentData = async () => {
  try {
    const response = await fetch('https://test.json');
    const json = await response.json();
```

```
    setData(json.data);
  } catch (error) {
    console.error(error);
  } finally {
    setLoading(false);
  }
}

useEffect(() => {
 getPaymentData();
}, []);
```

Kuva 19. *Rest-rajapinta tuki.*

Toiminnallisuus on toteutettu App.tsx tiedostoon React Hook:ia käyttämällä, jotta saatua dataa voidaan hallita keskitetysti yhdessä paikassa. App.tsx tiedostosta data pystytään lähettämään käyttöliittymä komponenteille props parametrien avulla näytettäväksi käyttöliittymäkomponenteissa.

6. TULOKSET JA NIIDEN ARVIOINTI

Lopuksi arvioidaan tutkimuksen ja prototyyppisovelluksen avulla saatuja tuloksia neljännessä kappaleessa sille asetettuja vaatimuksia vasten. Prototyyppisovelluksena toteutettiin React Nativella itsepesupalvelun maksusovellus. Työssä selvitettiin uusien teknologioiden soveltuvuutta osana modernia arkkitehtuurimallia. Prototyyppisovelluksen avulla pystytään vastaamaan suurimpaan osaan asetuista kysymyksistä, mutta osa kysymyksistä jää selvitettäväksi jatkotutkimuksen puolelle.

Taulukko 4. *Tutkimuksen tulokset listattuna.*

Vaatimus	Selvitys	Tulos
1. **Arkkitehtuurin tulee tukea alustariippumatotonta kehitystä sekä Android että iOS alustoilla.**	React Native arkkitehtuurissa on sisäänrakennettu tuki sekä Android, että iOS alustalle. Tätä selvitettiin rakentamalla prototyyppisovellus ja tutkimalla aiheeseen liittyvää kirjallisuutta ja edeltäviä tutkimuksia React Nativesta.	Arkkitehtuurilla on React Nativen toimesta tuki sekä Android, että iOS alustalle.
2. **Arkkitehtuuria pitää olla mahdollista laajentaa tulevaisuudessa esimerkiksi lisäämällä se osaksi toista sovellusta verrattain pienellä työmäärällä.**	Sovelluksen arkkitehtuuri ja sen komponentit suunniteltiin modulaarisuusperiaatteen mukaisesti. Se mahdollistaa teoriassa toteutuksen liittämisen osaksi toista React Native toteutusta käyttämällä tässä työssä luotuja käyttöliittymäkomponentteja ja niiden logiikkaa.	Käyttöliittymäkomponentien modulaarisuus mahdollistaa arkkitehtuurin laajentamisen ja ylläpidon verrattain pienellä työmäärällä.
3. **Käyttöliittymän elementtien tulee huomioida sovelluksen**	Tutkimuksessa toteutettiin JSX:n ja TypeScriptin	Käyttöliittymäkomponentit ovat laajasti räätälöitäviä ja saavutettavia, täten

laaja käyttäjäkunta mahdollistaakseen sujuvan käyttökokemuksen.	avulla räätälöivät käyttöliittymäkomponentit modulaarisen arkkitehtuurimallin mukaisesti. React Nativen syntaksin todettiin myös antavan kehittäjälle tarpeelliset työkalut saavutettavuus vaatimusten täyttämiseksi kirjallisuuden pohjalta.	helppokäyttöisyys on toteutettavissa.
4. Arkkitehtuurin tulee olla moderni, vikasietoinen ja tietoturvallinen.	Arkkitehtuuri toteutettiin moderneilla teknologioilla ja alan parhaiden käytäntöjen mukaisesti. Arkkitehtuuri nojaa näiden teknologioiden laaja-alaisen käytön tuomaan suojaan ja käytettävyyteen. Sovellus on käyttöliittymä maksutapahtumalle. Kokonaisuutta arvioiden tietoturvan toteutumisen tarkempi tarkastelu on oleellisempaa palvelinpuolen teknisentoteutuksen ja REST-rajapinnan teknisentoteutuksen osalta, jotka rajautuvat tämän työn ulkopuolelle.	Tietoturvapoikkeamia tai muita ongelmia ei havaittu. Kattavampi selvitys olisi tarpeen jatkotutkimusta varten.

Ensimmäisenä vaatimuksena työlle asetettiin. että arkkitehtuurin täytyy tukea alustariippumatonta kehitystä sekä Android että iOS alustoilla. Työ kehitettiin Windows-alustalla ja maksupäätelaite toimii Android-alustalla, mikä oli yrityksen kannalta yksiköprioriteettina. IOS-laitteella testaus olisi vaatinut erillisen MacOS pohjaisen tietokoneen, jota ei ollut saatavilla tutkimusta varten. React Native sovelluskehys on kuitenkin lisännyt iOS-kehitystä varten tarvittavat riippuvuudet ja sen ottaminen mukaan kehitykseen ei vaatisi enää lisäkehitystyötä.

Toisena vaatimuksena työlle asetettiin, että työn tulee olla laajennettava tulevaisuudessa. Yrityksen toiveena oli, että maksusovellus voitaisiin tulevaisuudessa lisätä osaksi heidän olemassa olevaan React Native mobiilisovellukseen. Arkkitehtuurin modulaarisuus mahdollistaa sovelluksen lisäämisen osaksi toista sovellusta verrattain pienellä työmäärällä. Sovelluksen komponentit ovat myös responsiivisia ja modulaarisia, mikä tukee laajentamista tulevaisuudessa. Komponenttien modulaarisuus helpottaa sovelluksen ylläpitoa ja komponentit ovat uudelleen käytettäviä mikä vähentää duplikaattikoodia.

Käyttöliittymää suunniteltaessa tärkeäksi seikaksi nousi sen laaja käyttäjäkunta. Myös vanhempien ei diginatiivien tulisi pystyä käyttämään sovellusta. Tämä huomioiden sovelluksen käyttöliittymä haluttiin pitää mahdollisimman minimalistisena. Käyttöliittymäkomponentit suunniteltiin mahdollisimman suurikokoisiksi ja kaikki ylimääräinen karsittiin käyttöliittymästä. Käyttöliittymä onnistuttiin pitämään yhdessä näkymässä, mikä osaltaan selkeyttää käyttökokemusta. Käyttökokemus on kuitenkin loppujen lopuksi hyvin subjektiivinen kriteeri mikä vaikeuttaa sen arviointia. Käyttäjäkokemus vaihtelee aina käyttäjäkohtaisesti.

Arkkitehtuurin neljänneksi vaatimukseksi asetettiin, että sen tulee olla moderni, vikasietoinen ja tietoturvallinen. Prototyyppisovelluksella haluttiin demota uusia sovelluskehityksen teknologioita. Prototyyppisovellus toteutettiin TypeScriptillä pelkän JavaScriptin sijaan. TypeScript osaltaan nykyaikaistaa sovelluskehitystä tuomalla JavaScript kielen jo kauan kaipaaman tyyppitarkastuksen osaksi modernia tämän päivän sovelluskehitys standardia parantaen osaltaan sovelluksen vikasietoisuutta.

Modernissa sovelluskehityksessä on tärkeää pyrkiä hyvään koodin jäsentelyyn käyttämällä loogisesti valideja rakenteita luokkien ja kansio rakenteiden muodostamista varten. Näin sovelluksen ohjelmarakenne pysyy paremmin hallittavissa ja jatkokehittäminen helpottuu, kun vältytään liiallisen teknisen velan muodostumiselta. Tähän lopputulokseen päästiin modulaarisen arkkitehtuurin suunnittelumallilla.

React kirjastosta uutena teknologiana testattiin erityisesti Hook teknologian soveltuvuutta. Hook malli osoittautui toimivaksi ratkaisuksi maksusumman dynaamisessa hallinnoinnissa ja esittämisessä käyttöliittymäkomponentissa. Se onnistui vähentämään osaltaan tarvetta ylimääräiselle boiler plate koodille ja yksinkertaistamaan sovelluslogiikkaa, kun ylimääräistä luokkaa ei tarvinnut rakentaa. Käyttöliittymä puolella testattiin Flexboxin soveltuvuutta osana modernia kehitys prosessia. Flexboxin avulla saatiin jaoteltua käyttöliittymä komponentit joustavasti haluttuun malliin. Jako omiin segmentteihin oli helppoa ja luontevaa. Sovelluksen kehityksessä käytetty

ESLint kirjasto on staattinen koodin analysaattori. Se auttaa osaltaan vastaamaan neljänteen vaatimukseen sovelluksen vikasietoisuudesta.

Taulukko 5. *Toiminnalliset vaatimukset.*

Vaatimus	Selvitys	Tulos
1. **Sovelluksen tulee pystyä muodostamaan yhteys taustajärjestelmään REST-rajapinnan välityksellä.**	Sovellukselle luotiin REST-rajapintatuki.	Sovelluksella on valmius REST-rajapinta kommunikointiin.
2. **Sovelluksen tulee pystyä käsittelemään taustajärjestelmältä saatuja JSON-dataobjekteja.**	TypeScript sisältää tuen JSON-dataobjektien käsittelylle.	Sovellus pystyy käsittelemään JSON-dataobjekteja.
3. **Sovelluksen tulee pystyä lähettämään katevaraus taustajärjestelmään.**	Aikataulusyistä selvitys jäi tutkimuksen ulkopuolelle.	Ei toteutettu tämän tutkimuksen piirissä.

Sovellukseen luotiin REST-rajapintatuki jatkokehitystä varten. Sovellukseen suunniteltiin ja toteutettiin valmis luurankometodi myöhemmin käytettäväksi ja täydennettäväksi oikean REST-rajapinnan parametreilla. Sovelluksen initalisointivaiheessa REST-rajapinnasta ladataan asemakohtaiset arvot, jotka välitetään yksittäisille komponenteille esitystä varten. Sovelluksen käyttämä TypeScript-kieli mahdollistaa JSON-objektien käsittelyn, joita REST-rajapinnan on tarkoitus palauttaa. Maksuvarauksen tekeminen jäi tämän tutkimuksen ulkopuolelle aikataulullisista syistä.

6.1 Arkkitehtuurin heikkoudet ja jatkotutkimus

Tutkimuksessa toteutetun alustariippumattoman arkkitehtuurin suurin heikkous on, että se nojaa täysin React Nativen tukeen. Jos Meta tekee suuria muutoksia sovelluskehykseen tai lakkauttaa sen aiheuttaa se väistämättä suuria ongelmia projektille. Toisaalta sama ongelma on millä tahansa arkkitehtuurilla, joka nojaa johonkin tiettyyn teknologiaan tai sovelluskehykseen. Siksi onkin tärkeää arvioida ennen teknologian valitsemista arkkitehtuurille, sen uskottavuutta pidemmällä aikavälillä. React Nativella on laajakäyttäjäkunta ja tuki ja onkin hankala nähdä suuria uhkia sen käyttöön liittyen lähitulevaisuudessa.

Tutkimuksessa jäi vielä muutamia aiheita, joita voisi selvittää jatkotutkimuksessa. Sovelluksen toimintaa voitaisiin vielä erikseen arvioida Applen kehitysekosysteemissä. Apple on rajoittanut sen käytön heidän oman macOS käyttöjärjestelmän ja

Xcode kehitysympäristön taakse. Applen toimintapa on hyvin vaikea Windows alustalla toimivalle kehittäjälle, kun iOS alusta on poissuljettu, vaikka omistaisi iOS pohjaisen puhelimen tai tabletin.

Sovelluksen tarkoituksena oli myös mahdollinen integrointi osaksi toista järjestelmää. Teoriassa ohjelmakoodi pitäisi olla liitettävissä osaksi toista React Native projektia. Tämä voi olla kuitenkin käytännössä hankalampaa kuin mitä aiemmin suunniteltiin, johtuen koodin muuttujien päällekkäisyyksistä ja kahden eri sovelluksen käyttöliittymän tyylien eroavaisuuksista.

Katevarauksen tekeminen taustajärjestelmään olisi myös mielenkiintoinen jatkotutkimusaihe. Tämä jäi kuitenkin toteuttamatta, koska sen vaatima REST-rajapinta ei ehtinyt toteutua tämän tutkimuksen aikataulussa ja tutkimus rajautui React Nativen tutkimiseen.

7. YHTEENVETO

Diplomityön aiheena oli tutkia React Native ohjelmistokehystä osana alustariippumatonta ohjelmistokehitystä eri mobiililaitteille. Tutkimuksessa suunniteltiin sovelluksen arkkitehtuuri ja toteutettiin sen pohjalta prototyyppisovellus mobiilimaksamista varten auton itsepesupalvelulle. Tutkimuksessa käytettiin konstruktiivista tutkimusmenetelmää, jossa ensiksi tutustuttiin teoriaan ja sitten sen pohjalta innovoitiin ratkaisu tosielämän ongelmaan.

Tutkimuksessa luotiin responsiivinen alustariippumaton mobiilisovellus, jossa käytettiin monia uusia teknologioita. Sovellus toteutettiin TypeScriptillä paremman hallittavuuden takaamiseksi. Muuttujien tyyppitarkastus verrattuna tavalliseen JavaScript toteutukseen tuo osaltaan paremman vikasietoisuuden sovellukselle. React Hook teknologiaa käytettiin kokonaismaksusumman tilan hallintaan ja REST-rajapintatuen tekoon. React Hook teknologia yksinkertaistaa toteutusta vähentämällä ylimääräistä boiler-plate koodia. Käyttöliittymän asetteluun käytettiin Flexbox teknologiaa. Se on moderni tapa hallita näkymän asettelua jakamalla näyttö paloihin. Flexbox hoitaa itse näkymän skaalauksen eri kokoisille näytöille. Käyttöliittymäkomponentit toteutettiin modulaarisesti omiin luokkiinsa JSX teknologialla. Komponenttien modulaarisuus helpottaa sovelluksen ylläpitoa ja komponentit ovat uudelleenkäytettäviä mikä vähentää duplikaattikoodia.

Tutkimuksessa toteutuneelle prototyyppisovelluksen arkkitehtuurille asetettiin neljä vaatimusta ja toiminnallisia vaatimuksia asetettiin kolme kappaletta. Arkkitehtuurille asetetut vaatimukset saatiin täytettyä ja todistettua prototyyppisovelluksen avulla suurilta osin. Sovelluksen tietoturvan ja vikasietoisuuden testaamiseen olisi voitu käyttää lisää aikaa, mutta se ei ollut tämän tutkimuksen kriittisin piste. Tutkimus keskittyi uusien teknologioiden soveltuvuuden arvioimiseen. Arkkitehtuurin vaatimusten täyttyminen osaltaan puoltaa näiden uusien teknologien käytännöllisyyttä. Tutkimuksen toiminnalliset vaatimukset täyttyivät niiltä osin, miten niitä pystyttiin tutkimaan tämän työn puitteissa. React Nativella toteutetulla ohjelmistolla ei ole erityisiä esteitä REST-rajapintojen hyödyntämisessä.

LÄHTEET

[1] Asahi Technologies, 5 Benefits of cross-platform mobile app development, 2020. Saatavissa https://www.asahitechnologies.com/blog/5-benefits-of-cross-platform-mobile-app-development/

[2] B. Skuza, D. Włodarczyk, A. Mroczkowska (2021). Droids On Roids, Flutter vs. React Native – What to Choose in 2022?, Saatavissa https://www.the-droidsonroids.com/blog/flutter-vs-react-native-what-to-choose-in-2021

[3] C. Mariana, Back4App, Top 4 Alternatives to React Native (#3 is amazing), Saatavissa https://blog.back4app.com/react-native-alternatives/

[4] Castles Technology, https://www.castlestech.com/products/saturn1000-e-upt/

[5] J. Kaneko, Understanding React Native Architecture, 2020. Saatavissa: https://dev.to/goodpic/understanding-react-native-architecture-22hh

[6] K. Lukka, Konstruktiivinen tutkimusote, Metodix, 2001. Saatavissa https://metodix.fi/2014/05/19/lukka-konstruktiivinen-tutkimusote/

[7] M. Bohara (2020), Comparison between React Native and native application development platform, Saatavissa https://urn.fi/URN:NBN:fi:tuni-202007076342

[8] Mdn web docs, Promise, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

[9] Node.js, https://nodejs.org Saatavissa https://nodejs.org/en/

[10] Node.js, Introduction to Node.js, https://nodejs.dev/. Saatavissa https://nodejs.dev/learn

[11] npm, www.npmjs.com, Saatavissa https://www.npmjs.com/

[12] P. Ronja, Mobiilisovelluksesta saavutettava: kädet saveen Saatavissa https://www.eficode.com/fi/blog/mobiilisovelluksesta-saavutettava-kadet-saveen

[13] React Native, “React Native, Learn once, write anywhere: Build mobile apps with React,” Saatavissa: https://reactnative.dev/

[14] React Native, Layout with Flexbox https://reactnative.dev/docs/flexbox

[15] React, Introducing Hooks, reactjs.org. https://reactjs.org/docs/hooks-intro.html

[16] React, reactjs.org. Saatavissa https://reactjs.org/

[17] S. Shaibana, The Pros and Cons of Cross-Platform Apps, 2021. Saatavissa https://www.focaloid.com/the-pros-and-cons-of-cross-platform-apps/

[18] T. Kunnamo, Alustariippumaton arkkitehtuuri Web- ja Mobiilisovelluksille, 2018. Saatavissa https://urn.fi/URN:NBN:fi:tty-201805241853

[19] The Open Source Definition, https://opensource.org/about, 2007. Saatavissa https://opensource.org/docs/osd

[20] TypeScript, TypeScript for the New Programmer, typescriptlang.org. Saatavissa https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html

[21] TypeScript, typescriptlang.org/. Saatavissa https://www.typescriptlang.org/

[22] W3schools, Node.js Introduction, w3schools.com. Saatavissa https://www.w3schools.com/nodejs/nodejs_intro.asp

[23] W3schools, React Hooks, w3schools.com. Saatavissa https://www.w3schools.com/react/react_hooks.asp

[24] W3schools, What is npm?, w3schools.com. Saatavissa https://www.w3schools.com/whatis/whatis_npm.asp

[25] Wikipedia, Alustariippumattomuus, 2020. Saatavissa https://fi.wikipedia.org/wiki/Alustariippumattomuus

[26] Wikipedia, Deklaratiivinen ohjelmointi wikipedia.fi. Saatavissa https://fi.wikipedia.org/wiki/Deklaratiivinen_ohjelmointi

[27] Wikipedia, JavaScript, wikipedia.fi, 2021. Saatavissa https://fi.wikipedia.org/wiki/JavaScript

[28] Wikipedia, Open source, wikipedia.fi, 2022. Saatavissa https://en.wikipedia.org/wiki/Open_source

[29] Wikipedia, React (JavaScript library), wikipedia.fi. Saatavissa https://en.wikipedia.org/wiki/React_(JavaScript_library)

[30] Wikipedia, TypeScript, wikipedia.fi. Saatavissa https://en.wikipedia.org/wiki/TypeScript