Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

TUOMAS NYLUND

Taksin tilaussovelluksen luominen web-ympäristöön

TIETO- JA VIESTINTÄTEKNIIKAN KOULUTUSOHJELMA
2021

Tekijä(t)	Julkaisun laji	Päivämäärä
Nylund, Tuomas	Opinnäytetyö, AMK	01 / 2021
	Sivumäärä 30	Julkaisun kieli Suomi

Julkaisun nimi
Taksin tilaussovelluksen luominen web-ympäristöön

Tutkinto-ohjelma
Tieto- ja viestintätekniikka

Työn tavoitteena oli luoda sovellus, millä voidaan tilata takseja. Samalla sovelluksen oli myös oltava käytettävissä taksikuskilla, minkä kautta kuskilla on mahdollisuus hallinnoida tilauksia.

Aluksi selvitettiin sovelluksen alusta, eli luotaisiinko sovellus verkkoympäristöön vai luotaisiinko siitä puhelinsovellus. Työssä päädyttiin verkkosovellukseen.

Kun alusta oli päätetty, oli selvitettävä vaadittavat ja työn mahdollistavat teknologiat. Työssä päädyttiin käyttämään moderneja verkkosovellusteknologioita, kuten Javascript-kirjastoa Vueta ja niin sanottua Back End as a Service (BaaS) palvelua Firebasea.

Sovelluksesta pyrittiin myös tekemään mahdollisimman yksinkertainen, sekä taksin tilaajalle, että kuskille. Myös käyttöliittymä pyrittiin pitämään yksinkertaisena, mahdollisimman modernina ja responsiivisena.

Selvitettiin myös mahdollisia riskejä ja haittoja, mitä esiintyi työtä luodessa, kuten sovelluksen tietoturva.

Työssä onnistuttiin hyvin. Sovelluksesta saatiin toimiva, käyttäjäystävällinen ja turvallinen.

Avainsanat
Verkkosovellus, Tietokanta, Käyttöliittymä

Author(s) Nylund, Tuomas	Type of Publication Bachelor's thesis	Date 01 / 2021
	Number of pages 30	Language of publication: Finnish

Title of publication

Creating a taxi ordering application for the web environment

Degree program

Bachelor's degree in information and communications technology

The goal of this thesis was to create an application where one could order a taxi. Simultaneously the application would be used by the driver who could manage the orders.

Whether the application was to be a web application or a phone application was chosen in the first phase. The platform ended up being a web-app.

When the platform was chosen, the technologies needed to be determined as well. The application was built with modern web-technologies with the core of the application being a Javascript library Vue and a Back End as a Service (Baas) Firebase.

The goal was also to make the application as simple as possible, for the regular users as well as the driver. The user interface was kept simple, modern and as responsive as possible.

It was also important to research possible risks and harms that the application and its user might face.

The project ended up well made. The application was functional, user friendly and secure.

Web-application, Database, User interface

SISÄLLYS

SANASTO

Angular	Javascript-kirjasto
Back End	Sovelluksen tietokanta ja sen hallinta
CLI	Comman Line Interface. Komentokehote tai komento-rivi
Commit	Commit komentoa käytetään projektin tallennukseen Git-versionhallinnassa
CRUD	Create, read, update, delete. (Luo, lue, päivitä, poista).
CSS	Cascading style sheets. Kuvaa kuinka HTML-elementit näytetään
DOM	Document Object Model tai dokumenttioliomalli. Käytetään kuvaamaan rakenteisen dokumentin, kuten HTML:n rakenne puuna.
Front end	Web-sovellusten käyttöliittymä eli näkyvä osa
Git	Ohjelmistoprojektien versionhallintatyökalu
JSON	Javascript Object Notation, standardoitu tiedosto-muoto
JSON Web Token	Käyttöoikeustietueiden hallinnoimiseen käytettävä menetelmä
Javascript	Ohjelmointikieli

MongoDB	NoSql-tietokantaohjelmisto
NoSql	Tietokantarakenne.
React	Javascrit-kirjasto
URL	Uniform Resource Locator, verkkosivun osoite.
Vue	Javascript-kirjasto

1 JOHDANTO

Opinnäytetyön tavoitteena on luoda taksin tilaussovellus ja se toteutetaan toimeksiantona Porin A Taxit Oy:lle. Sovellus on tarkoitus luoda web-ympäristöön, käyttäen nykyaikaisia web-teknologioita. Nettisivulle pitää pystyä rekisteröitymään, kirjautumaan, jättämään taksin tilauspyyntö ja seurata tilauksen tilaa. Saman sovelluksen on myös toimittava kuskin käytössä. Kuskin on pystyttävä seuraamaan uusia tilauksia ja hyväksymään tai hylkäämään niitä. Sovelluksella on siis myös pysyttävä erottelemaan peruskäyttäjät ja kuskit, sekä eri käyttäjien näkymät.

Opinnäytetyön aluksi on tarkoitus selvittää sovellukselle sopivat teknologiat ja käsitellä, miksi päädyttiin web-pohjaiseen sovellukseen puhelinsovelluksen sijasta. Käydään läpi front-end frameworkin valinta, sen käyttöönotto ja hyödyt. Lisäksi käsitellään back-end teknologian valinta, siihen liittyvät hyödyt ja mahdolliset haittapuolet sekä riskit.

Lopuksi käydään läpi itse työtä. Opinnäytetyössäni vastaan seuraaviin kysymyksiin: miten sovellus rakennettiin uusia teknologioita käyttäen, mitä haasteita kohdattiin työtä tehdessä ja miten ne ratkaistiin.

2 MOBIILISOVELLUKSET JA VERKKOSOVELLUKSET

Ensinäkemältä mobiilisovellukset ja verkkosovellukset voivat näyttää täysin samalta, mutta eroavaisuuksia löytyy kuitenkin paljon. Käyttäjäkokoemuset eroavat toisistaan ja itse sovellusten kehittäminen ja käyttöönotto ovat hyvinkin erilaisia. (Stevens 2018)

On myös tärkeää erotella verkkosovellukset ja perinteiset nettisivut. Nettisivut tai kotisivut ovat staattisia tai harvoin päivitettäviä sivuja, millä voi olla myös hieman interaktiivisuutta. Verkkosovellukset taas toimivat kuin puhelinsovellukset, mutta sen sijaan että sovelluksia tarvitsisi erikseen ladata esimerkiksi sovelluskaupasta, löytyvät ne yksinkertaisesti selaimen kautta. (Stevens 2018).

2.1 Verkkosovellukset

Verkkosovellukset luodaan usein web teknologioiden kuten HTML:n, CSS:n ja Javascriptin avulla ja verkkosovellukset perustuvat usein CRUD tyyppiseen hallintaan. CRUD on lyhenne sanoista create, read, update, delete. Verkkosovelluksissa on siis usein mahdollista luoda uutta tietoa, lukea vanhaa tietoa, päivittää ja poistaa tietoa. Verkkosovellusten käyttäminen tapahtuu selainten, kuten Google Chrome, avulla ja sovelluksilla on usein jonkinlainen kirjautumisjärjestelmä. (Johnston 2020).

2.1.1 Progressiiviset verkkosovellukset

Progressiivisella verkkosovelluksella tarkoitetaan sovellusta, joka pyrkii käyttämään hyväksi responsiivisen verkkosovelluksen, sekä natiivin mobiilikehityksen parhaat puolet. Progressiivisen verkkosovelluksen kehittämisessä keskitytään parantamaan sovelluksen käyttäjäkokemusta mobiililaitteilla ja täten tehdä sovelluksesta enemmän natiivisovelluksen kaltainen. (Itewiki WWW-sivut 2020).

Yksi tärkeimpiä ominaisuuksia progressiivisessa verkkosovelluksessa on mahdollisuus käyttää laitteiden sisäänrakennettuja ominaisuuksia, kuten gps-paikannusta tai kameraa. Sovellus pystyy myös lähettämään käyttäjälle push-ilmoituksia, mikä on erittäin tärkeä ominaisuus digitaalisen markkinoinnin kannalta. Progressiivisen verkkosovelluksen käyttäminen ja siellä vierailu tapahtuu kuitenkin yhtä helposti kuin minkä tahansa muun sivun selailu. (Itewiki WWW-sivut 2020).

Progressiivisen verkkosovelluksen termiä käytettiin Googlen insinöörin Alex Russelin toimesta vasta vuonna 2015. Nykyisin kuitenkin progressiiviset verkkosovellukset ovat nouseva ohjelmistotekniikan trendi. Vuonna 2018 käytetyimmät verkkoselaimet tukivat progressiivisia verkkosovelluksia. (Itewiki WWW-sivut 2020).

2.1.2 Verkkosovelluksen hyödyt

Toisin kun mobiilisovellukset, mitkä toimivat vain tietyllä alustalla, kuten Android tai IOS, verkkosovelluksia voidaan käyttää millä laitteella tahansa, kunhan käyttäjällä on verkkoyhteys ja kunhan sovellus tukee käyttäjän käyttämää verkkoselainta. Tämän takia verkkosovelluksen koodistoa on helppo ylläpitää, sillä sama ohjelma toimii monella eri alustalla. Käyttäjän ei myöskään tarvitse päivittää sovellusta erikseen, vaan uusin versio latautuu kaikille käyttäjille ohjelman latauksen yhteydessä. Kehittäjien ei myöskään tästä syystä tarvitse ottaa huomioon eri sovellus, käyttöjärjestelmä, -tai puhelinversioita. Kun yksi koodisto toimii monella eri alustalla ja on vähemmän huomioonotettavaa, pysyvät usein verkkosovellusten kustannukset mobiilisovelluksia alahaisempana. (Clearbridge Mobile WWW-sivut 2020).

Verkkosovelluksien ei myöskään tarvitse selviytyä sovelluskauppojen asettamista standardeista tai vaadi niiden hyväksyntää. Verkkosovellukset voidaan julkaista milloin vain ja missä tahansa muodossa. (Clearbridge Mobile WWW-sivut 2020).

2.1.3 Verkkosovellusten haittapuolet

Verkkosovellukset vaativat selaimen toimiakseen ja täten myös internetyhteyden. Käyttäjän on tehtävä enemmän työtä avatakseen sovelluksensa, kirjoittamalla sovelluksen osoite URL:iin tai etsiä se hakukoneen avulla. Verkkosovellusten on myös otettava huomioon eri selaimet ja miten niiden toiminta eroaa toisistaan. Verrattuna mobiilisovelluksiin, verkkosovellukset voivat myös olla hitaampia ja vähemmän responsiivisia. Mobiilisovellukset pystyvät myös paremmin käyttämään hyväksi laitteen sisäisiä ohjelmistoja ja voivat täten olla interaktiivisempia. Verkkosovellukselle on myös vaikeampi saada näkyvyyttä ilman hyvää markkinointia. (Clearbridge Mobile WWW-sivut 2020).

2.2 Mobiilisovellukset

Mobiilisovellukset ovat pieniä sovellusyksiköitä, joiden toiminta on rajoitettua. Tämä sovellustyyppi on suunniteltu toimimaan mobiililaitteella, kuten älypuhelimella tai tablettitietokoneella. (Mroczkowska 2020).

Toisin kuin pöytätietokonesovellukset, mobiilisovellukset eivät ole integroituja ohjelmia. Sen sijaan jokainen mobiilisovellus tarjoaa erillisen ja rajoitetun toiminnallisuuden. Mobiilisovellus voi olla esimerkiksi peli, laskin tai verkkoselain. (Mroczkowska 2020).

Ensimmäiset mobiilisovellukset välttivät laajaa toiminnallisuutta varhaisten älypuhelimien laitteistorajoituksien takia, mutta mobiilisovellusten toiminta on edelleen hyvin kapeaa, vaikka älypuhelimet ovat kehittyneet huimasti. Mutta kun mobiilisovellusten toiminnallisuus on rajattua, saavat kuluttajat valita haluamansa toiminnallisuudet. (Mroczkowska 2020).

2.2.1 Mobiilisovellusten kehittäminen

Mobiilisovelluksia voidaan kehittää natiivisti tai järjestelmäriippumattomasti. Natiivit mobiilisovellukset on rakennettu tietylle alustalle käyttäen tiettyjä teknologioita. Esimerkiksi Applen iOS käyttöjärjestelmälle luodut sovellukset on kirjoitettu Swift- tai Objective-C- ohjelmointikielillä, kun taas Androidille luodut sovellukset Javalla tai Kotlinilla. Näin ollen Androidille luodut sovellukset eivät toimi iOS järjestelmän laitteissa tai päinvastoin. (Clearbridge Mobile WWW-sivut 2020).

Järjestelmäriippumattomat mobiilisovellukset on suunniteltu toimimaan alustasta riippumatta. Nämä sovellukset voidaan kehittää käyttämällä samaa koodistoa ja täten vältetään saman sovelluksen luominen moneen kertaan eri alustoille. Järjestelmäriippumattomat sovellukset voivat kuitenkin olla suorituskyvyltään heikompia kuin natiivit järjestelmät. (Fullscale WWW-sivut 2020).

2.2.2 Natiivimobiilisovellusten hyödyt

Natiivimobiilisovellusten suorituskyky on parempi kuin verkkosovellusten, sillä applikaatio on suunniteltu ja optimoitu yhdelle alustalle. Sovellus voi käyttää hyväksi laitteen prosessointinopeutta ja koska visuaaliset elementit ovat jo tallennettuja puhelimeen, ovat latausajat nopeita. (Clearbridge Mobile WWW-sivut 2020).

Natiivisovellukset toimivat sulavasti ja niihin on helpompi saada hyvä käyttöliittymäintegraatio. Natiivisovellukset siis luodaan yhtä käyttöjärjestelmätyyppiä ajatellen, esimerkiksi Android, ja täten sovellus voi näyttää ja tuntua siltä kuin se olisi osa järjestelmää. Tämä helpottaa myös sovelluksen oppimista. (Clearbridge Mobile WWW-sivut 2020).

Natiivisovellukset voivat hyödyntää eri käyttöjärjestelmien ominaisuuksia. Sovelluksien on mahdollista käyttää esimerkiksi puhelimen tai tablettilaitteen GPS:ää, kameraa tai vaikka mikrofonia. Myös push-ilmoitusten käyttäminen on helpompaa natiivisovelluksella. (Clearbridge Mobile WWW-sivut 2020).

Natiivisovellusten kehityksessä voi ilmaantua vähemmän virheitä. Koska koodit on eroteltu ja suunniteltu eri alustoille, kehittäjä ei ole riippuvainen eri työkalujen, kuten Xamarinin tai Cordovan, toiminnasta. Natiivisovellusten kehittäjillä on lisäksi pääsy uusimpiin käyttöjärjestelmän toimintoihin nopeasti. (Clearbridge Mobile WWW-sivut 2020).

2.2.3 Natiivisovellusten haittapuolet

Natiivisovelluksilla on myös haittapuolensa. Sovellusta ladattaessa täytyy käydä läpi monta eri kohtaa: etsiä applikaatio sovelluskaupasta, asentaa sovellus, avata se ja kirjautua sisään. Kaikilla käyttäjillä ei ole aikaa tai kärsivällisyyttä käydä tätä prosessia läpi ja näin ollen jättävät sovelluksen asentamisen kesken. (Uzair Khan 2018).

Natiivisovellukset vaativat myös joko laajaa osaamista kehittäjiltä tai mahdollisesti jopa kaksi eri tiimiä kehittämään sama sovellus Android- ja iOS- alustoille. Tämä saattaa nostaa sovelluksen kehittämiskuluja. (Uzair Khan 2018).

Natiivisovelluksissa on lisäksi otettava huomioon eri laitteet ja käyttöjärjestelmien versiot. Käyttäjien on myös ladattava päivityksiä manuaalisesti, joten kehittäjien on huomioitava oman sovelluksensa eri versiot. (Clearbridge Mobile WWW-sivut 200).

3 VUE.JS

Vue.js, paremmin tunnettu nimellä Vue, on ex-Google- työntekijän Evan Youn luoma Javascript-kirjasto. Työskennellessään Googlella You käytti töissään Googlen omaa Angular Javascript-kirjastoa. You piti Angularista, mutta hänen mielestään se oli liian raskas hänen käyttötarpeisiinsa. You päätti ottaa pitämänsä asiat Angularista ja luoda oman kevyemmän kirjastonsa. Pienien askelien myötä Vue kehittyi ensimmäiseen versioonsa vuonna 2013. Myöhemmin vuonna 2014 You päätti julkaista Vuen myös muiden käytettäväksi. (Cromwell 2017).

Youn mukaan Vue on kuitenkin lähempänä Facebookin kehittämää ja ylläpitämää React Javascript-kirjastoa, kuin Angularia. Samoin kuin Reactissa, Vuen ydin on datan sitominen ja komponenttien käyttäminen. Vue on kuitenkin helpommin lähestyttävä niille kehittäjille, joilla on jo kokemusta kokemusta HTML:ta, CSS:ta ja Javascriptista, Reactiin verrattuna. (Cromwell 2017).

Vue on kevyt, sillä se tarjoaa vain pienen osan ominaisuuksista core-kirjaston mukana ja osan ominaisuuksista erillään. Erillisiä ominaisuuksia ovat esimerkiksi tilan hallinta (state management) tai reititys (router). (Cromwell 2017).

3.1 DOM ja Vuen virtuaalinen DOM

Document Object Model (DOM) on HTML- ja XML-dokumenttien ohjelmointirajapinta. Se edustaa verkkosivua siten, että eri ohjelmat voivat muuttaa dokumentin rakennetta, tyyliä ja sisältöä. Verkkosivu on siis dokumentti ja tämä dokumentti voidaan näyttää joko selaimessa tai HTML-lähteenä. Dokumentti on kuitenkin sama molemmissa tapauksissa. DOM esittää tämän dokumentin sellaisessa muodossa, että sitä voidaan muokata ohjelmointikielien, kuten Javascript, avulla. (Mozilla WWW-sivut 2020).

Monet Javascript kirjastot tai frameworkit, kuten Vue, React tai Ember käyttävät niin sanottua virtuaalista DOM:ia. Virtuaalinen DOM parantaa verkkosivun nopeutta ja tehokkuutta, muiden hyötyjen lisäksi. (Gore 2017).

Tavallisen DOM:in päivittäminen on raskasta. Selaimen on löydettävä mahdollisesti tuhansien solmujen(node) tai elementtien joukosta haluttu kohde ja tehdä halutut muutokset. Virtuaalisessa DOM:ssa taas solmut voidaan ajatella samoin kuin Javascriptin objektit. Jos muokkaus taas virtuaalisen DOM:in avulla, muokataan vain Javascript-objektia ja tämä on nopeampaa kuin jos muokkauksessa jouduttaisiin kutsumaan DOM-rajapintaa. Virtuaalinen DOM ei vain paranna suorituskykyä, mutta tarjoaa myös eri funktionalisuuksia. (Gore 2017).

3.2 Vuen käyttöönotto

Vuen käyttöönotto on helppoa. Vuen käyttö voidaan aloittaa yksinkertaisesti lisäämällä se script-tagien sisälle. Kuvassa 1 on kuvattu yksinkertainen Vue ohjelma. Ohjelma yksinkertaisesti tulostaa sivulle lauseen ”Hei maailma!”, käyttäen Vuen data attribuuttia sekä Vuessa käytettävää aaltosulkusyntaksia tai ”viiksisyntaksia”.

```
<html>
  <body>
    <div id="esimerkki">
      <p>{{ hei }}</p>
    </div>
    <script src="https://unpkg.com/vue"></script>
    <script>
        new Vue({
            el: '#esimerkki',
            data: { hei: 'Hei maailma!' },
        })
    </script>
  </body>
</html>
```

Kuva 1. Esimerkki yksinkertaisesta Vue ohjelmasta.

Jos tarkoituksena on tehdä isompikin projekti Vuella, niin suosituksena on ladata Vue NPM:n avulla. Vuella on myös virallinen CLI(Command Line Interface), mikä mahdollistaa Vue-komentojen käytön terminaalissa ja näiden komentojen, kuten ”vue create” avulla, voidaan nopeasti luoda uusia projekteja. (Vue.js WWW-sivut 2020).

Vue tarjoaa myös kehittäjilleen oman selainlaajennuksensa, Vue Devtoolsin. Vue Devtools on laajennus, josta voit tutkia esimerkiksi oman projektisi komponentteja, niiden vuorovaikutusta ja sivulla olevaa dataa. (Copes 2018).

3.3 Vuen komponentit ja tilan hallinta

Vuen komponentin ovat uudelleenkäytettäviä instansseja, mitä voidaan käyttää missä tahansa Vuen sivulla tai osana sivua. Ja koska komponentit ovat uudelleenkäytettäviä, on niillä myös omat esimerkiksi omat metodit, data attribuutit ja elinkaaret. (Vue.js WWW-sivut 2020).

Kuvassa 2 on esimerkki yksinkertaisesta Vue komponentista. Siinä luodaan komponentti ”buttonCounter” ja tämä lisätään esimerkkisivulle. Komponenttia voidaan käyttää hyvin samakaltaisesti, kuin tavanomaista html elementtejä kirjoittamalla <button-counter>Paina tästä</button-counter>. Kuvassa 3 näkyy tämän esimerkin sivu. Sivulla näkyy komponentin avulla luotu nappi ja nappia on painettu kaksi kertaa, joten siinä lukee ”Painoit minua 2 kertaa”. Tätä nappikomponenttia voitaisiin käyttää myös monta kertaa samalla sivulla ja vaikka komponentteja olisikin monta, on niillä silti omat data-attribuuttinsa. Omat attribuutit mahdollistavat sen, että kaikilla ”buttonCounter” komponenteilla on omat ”count” -arvonsa ja täten näyttävät omia lukemiaan.

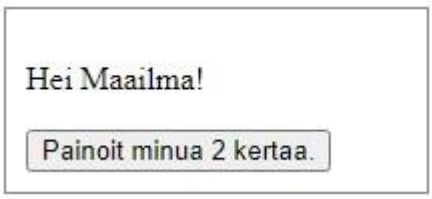
```
<html>
    <body>
        <div id="esimerkki">
            <p> {{ hei }} </p>
            <button-counter>Paina tästä</button-counter>
        </div>
        <script src="https://unpkg.com/vue"></script>
        <script>


            var buttonCounter = Vue.component('buttonCounter', {
                data: function () {
                    return {
                        count: 0
                    }
                },
                template: '<button v-on:click="count++">Painoit minua {{ count }} kertaa.</button>'
            })

            new Vue({
                el:"#esimerkki",
                data: {hei: "Hei Maailma!"},
                components: {
                    'button-counter': buttonCounter,
                }
            })

        </script>
    </body>
</html>
```

Kuva 2. Esimerkki Vuen komponentista

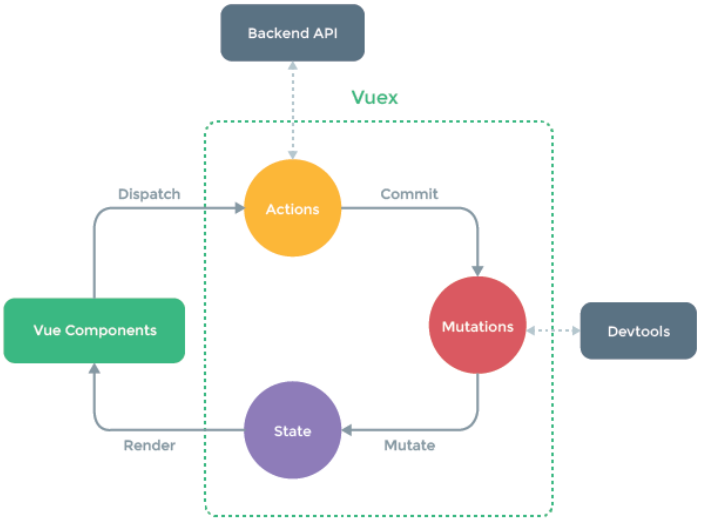


Kuva 3. Vue -sivu mihin lisätty nappi komponentti.

Vuen komponenteilla on siis oma tilansa, jota hallinnoidaan käyttämällä sen data-attribuuttia. Vuen data-attribuutti on hyvin monikäyttöinen ja sitä voidaan käyttää esimerkiksi käyttöliittymän hallintaan. Esimerkiksi onko alasvetovalikko avattuna tai ollaanko hakemassa dataa. Kun on puhe komponenttien tilasta, puhutaan paikallisesta tilasta, sillä vaikutus on vain kyseiseen komponenttiin. (Rakowski 2020).

Vue tarjoaa myös tilanhallintakirjaston Vuex:n. Se mahdollistaa jaetun datan käytön komponenttien välillä, käyttämättä perinteistä props-menetelmää. Vuex on hyödyllinen työkalu, jos samaa dataa joudutaan käyttämään ja hallinnoimaan eri

komponenteissa. Vuex:n käyttö lisää kompleksisuutta sovelluksen kehittämiseen, mutta sen avulla sovellus pysyy myös paremmin organisoituna, sillä se vähentää liikennettä komponenttien välillä. (Flaviocopes WWW-sivut 2020). Kuvassa 4 kuvataan miten Vuex kommunikoi Vuen komponenttien, backendin ja DevToolsien, kanssa.



Kuva 4. Visuaalinen kuvaus Vuex:n toiminnasta (Vue.js)

3.4 Vue router

Vuen yksi parhaimpia ominaisuuksia on Single Page Applications (SPA) tai yhden sivun sovelluksien luominen. Yhden sivun sovelluksilla tarkoitetaan sitä, että kaikki sivun sisältö ladataan yhdellä latauksella ja uusia sivun latauksia ei vaadita, kun näkymä muuttuu. Tällöin käyttäjän käyttökokemus on sulava ja nopea. Jotta Vuella voidaan luoda yhden sivun sovelluksia, tarvitaan Vuen tarjoama lisäkirjasto Vue Router. (Maribojoc 2020).

Vue routerin asennus tapahtuu helposti Vue-cli:n avulla tai manuaalisesti asennettuna npm -komentona. Voimme terminaaliin kirjoittaa komennon ”npm install vue-router” ja kun lisäkirjasto on ladattu voimme lisätä sen Vue sovellukseemme. Lopulta luomalla Javascript -tiedoston, voimme määritellä routerin asetukset. (Maribojoc 2020).

Kuvassa 5 on esimerkki yksinkertaisesta Vue router javascript -tiedostosta, jossa on määritelty eri asetuksia. Ensiksi tiedostoon tuodaan itse Vue -kirjasto sekä sivulla käytetyt näkymät tai komponentit kirjautuminen ja etusivu. Routes taulukkoon on lisätty halutut navigointikohteet. Eli jos web-osoitteen tai URL:in perään ei lisätä mitään, ohjaa sivu käyttäjän kirjautumissivulle. Jos osoitteen perään taas lisätään /etusivu, niin Vue ohjaa käyttäjän tai renderöi sivulle etusivunäkymän. Routerin linkkejä voidaan nyt käyttää yksinkertaisesti luomalla Vuen oma elementti ”<router-link to=’/etusivu’></router-link>”.

```
import Vue from "vue";
import VueRouter from 'vue-router'
import Kirjautuminen from './views/Login.vue'
import Etusivu from './views/Login.vue'


Vue.use(VueRouter);

const routes = [
    {
        path: "/",
        name: "kirjautuminen",
        component: Kirjautuminen
    },
    {
        path: "/etusivu",
        name: "etusivu",
        component: Etusivu
    }
];


const router = new VueRouter({
    routes: routes,
    base: process.env.BASE_URL,
    mode: 'history'
  });
```

Kuva 5. Esimerkki yksinkertaisesta Vue Router javascript -tiedostosta.

4 SERVERLESS

Terminä ”serverless” tai palvelimeton voi olla hieman hämmentävä, mutta se ei tarkoita sitä, että serverless-teknologiaa käyttävät sovellukset eivät tarvitse palvelimia. Sovellukset edelleen tarvitsevat palvelimia, mutta jokin kolmas osapuoli huolehtii palvelimien hallinnasta ja skaalauksesta, eikä kehittäjän näin ollen tarvitse huolehtia näistä. Serverless-teknologia yleensä rakentuu kahdesta komponentista ”Function as a service” (FaaS) ja ”Backend as a service” (BaaS). Serverless-teknologiassa siis keskitytään vain koodin kirjoittamiseen, eikä siihen miten tai missä koodi suoritetaan. (Baez 2020).

4.1 Serverless-teknologian hyödyt

Serverless-teknologian käyttäminen on halvempaa, sillä maksat vain siitä mitä käytät. Et joudu maksamaan laitteistosta tai ylimääräisiä kustannuksia, kun sovelluksesi ei ole käytössä. Ja koska kolmas osapuoli on vastuussa ja hallinnoi palvelinpuolta, ei myöskään ole tarvetta huolehtia esimerkiksi uusista tietoturvapäivityksistä. Tämän lisäksi sovellus skaalautuu helposti. Kun käyttöä tulee lisää, skaalautuu palvelinpuoli vaaditun määrän. (Fisher 2018).

4.2 Serverless-teknologian haittapuolet

Koska sovelluksen palvelinpuolen hoitaa kolmas osapuoli, on sinun pelattava heidän sääntöjensä mukaan. Et voi vaikuttaa laitteistoon, päivityksiin tai ajoaikoihin ja tästä voi aiheutua johdonmukaisuusongelmia tai tämä voi rajoittaa käytettävissä olevia resursseja. Jos taas haluat vaihtaa palveluntarjoajaa, on vaihdon suorittaminen vaikeaa ja voit joutua uudelleenkirjoittamaan sovelluksesi koodia tai muokkaamaan sen toimintatapaa. (Fisher 2018).

Jos sovelluksen tarpeena on suorittaa pitkiä funktioita tai tehtäviä, on mahdollista, että kustannukset nousevat. Samoin jos funktioita suoritetaan harvoin, voi

sovelluksen toimintanopeus kärsiä, niin sanotun kylmäkäynnistyksen (”Cold start”) vuoksi. (Fisher 2018).

5 FIREBASE

Firebase on Googlen tarjoama BaaS (Backend as a service) -palvelu. Firebase siis vapauttaa sovelluskehityksen niin, ettei kehittäjän tarvitse kirjoittaa omia tietokantarajapintoja, käyttää aikaa itse tietokannan rakentamiseen tai palvelimien ylläpitämiseen. Tällöin voidaan käyttää enemmän aikaa käyttöliittymän luomiseen sekä käyttäjäkokemuksen parantamiseen. Firebase tarjoaa useita työkaluja helpottaakseen kehittäjien työskentelyä. (Batschinski n.d).

5.1 Firebasen reaaliaikainen tietokanta

Reaaliaikainen tietokanta, Firebase Realtime Database, on yksi Firebasen palveluista. Reaaliaikaisella tietokannalla tarkoitetaan sitä, kun tietokannassa tapahtuu muutos, niin muutos ja siihen liittyvä data lähetetään myös niille käyttäjille, jotka ovat liitettyinä tähän tietovirtaan. Esimerkkinä voidaan pitää keskustelusovellusta. Kun yksi käyttäjistä lähettää viestin sovelluksen avulla, päivittyy tietokanta ja muut keskusteluun osallistuneet henkilöt näkevät tietokannan muutokset uusina viesteinä. (Heddings 2020).

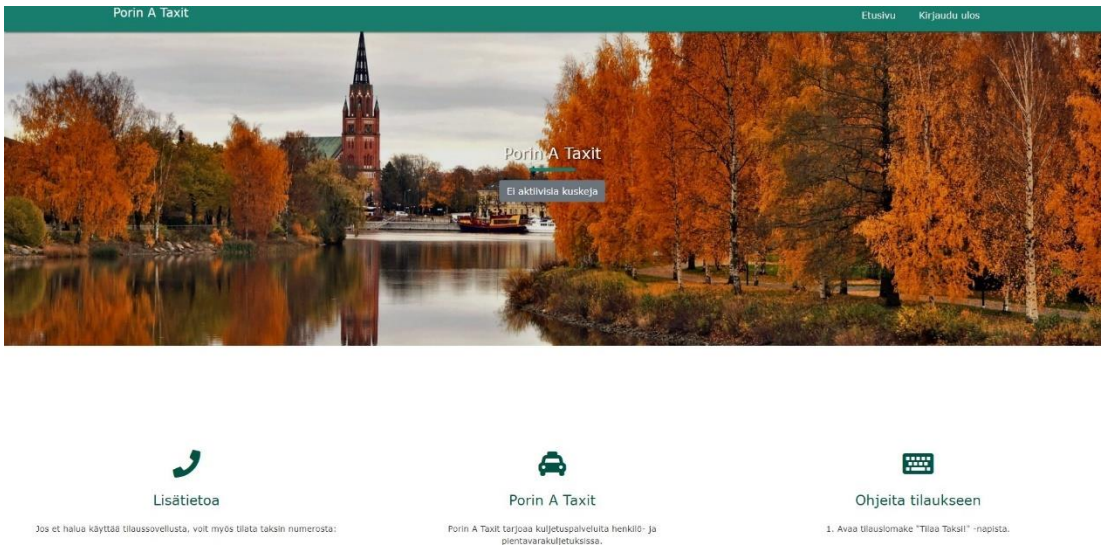
Firebasen tietokanta on dokumenttipohjainen järjestelmä, mikä muistuttaa hyvin paljon muita NoSql tietokantarakenteita, kuten esimerkiksi MongoDB:tä. Firebasen tietokanta on skeematon, jolloin voit tallentaa dataa missä tahansa muodossa. Data on tallennettuna isossa JSON-puussa. Jokaista puun oksaa voidaan muokata haluamallaan tavalla ja käyttäjä voi seurata muutoksia tietyissä oksissa. (Heddings 2020).

Firebase tarjoaa myös toista hyvin samankaltaista palvelua nimeltä Firebase Firestore. Molemmat Realtime Database ja Firestore ovat dokumenttipohjaisia NoSql tietokantoja. Edellä mainituista Firestore on kuitenkin enemmän jäsennelty järjestelmä. Ison JSON-puun sijaan, Firestore tallentaa tiedot eri dokumentteihin, joilla voi olla omat rakenteensa. Firestore pystyy myös samoin reaaliaikaisiin päivityksiin ja sen rakenne on parempi sovelluksille, jotka vaativat edistyneempiä kyselyitä. (Heddings 2020).

6 KÄYTTÖLIITTYMÄN SUUNNITTELU

Käyttöliittymää suunniteltaessa pyrittiin siihen, että sivu ja sen käyttö pysyy mahdollisimman yksinkertaisena sekä käyttäjän että kuskin näkökulmasta. Myös sivumäärät pyrittiin pitämään alhaisena. Verkkosivun tarkoituksena oli vain toimia tilaussovelluksena, joten yleiset tiedot yrityksestä ja taksikyydeistä pidettiin alkuperäisellä verkkosivulla. Lopulta peruskäyttäjälle näkyviin tuli ainoastaan neljä näkymää: kirjautumissivu, rekisteröitymissivu, sovelluksen etusivu ja tilauksen seurantasivu. Etusivulla taas on linkki tai nappi tilauslomakkeen avaamista varten. Tilauslomake avautuu modaali-ikkunana nappia painettaessa. Tällöin vähennetään käyttäjän tarvetta vaihtaa sivuja sovelluksessa. Tilauslomakkeen avausnappi sijoitettiin mahdollisimman näkyvälle paikalle, tämän ollessa oleellisin osa sivua ja jotta se on helppo löytää. Jos käyttäjä on tilannut taksin, samasta napista pääsee myös seuraamaan tilauksen tilannetta.

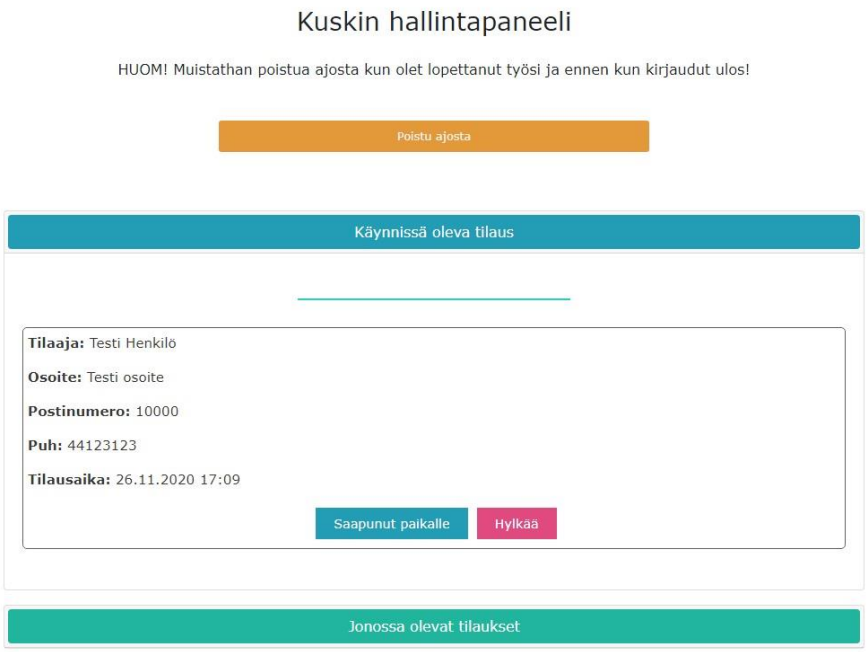
Kuvassa 6 on kuva etusivun käyttöliittymän yläosasta. Tilausnappi on deaktivoituna, sillä kuskeja ei ole aktiivisena. Etusivulla on myös yhteystietoja, linkki yrityksen kotisivuille sekä yksinkertaiset ohjeet tilauksen suorittamiseen.



Kuva 6. Verkkosovelluksen etusivu.

Kuskille luotiin oma hallintapaneeli ja siitä pyrittiin luomaan mahdollisimman yksinkertainen. Kuskin hallintapaneeliin ei lisätty mitään ylimääräistä, vaan käytettävyys pidettiin mahdollisimman helppona. Hallintapaneelissa on vain yksinkertaisesti mahdollisuus seurata tilauksia, hyväksyä ja hylätä niitä sekä siirtyä ajoon tai pois ajosta.

Kuvassa 7 on esimerkki kuskin hallintapaneelista. Ylhäällä on nappi ajoon ja ajosta pois siirtymiseen. Tämän alla on Bootstrapin kaksi ”accordion” tai haitarikomponenttia. Ylemmässä komponentissa on näkyvissä aktiivinen tai hyväksytty tilaus ja alemmassa komponentissa on uudet, hyväksymättömät tilaukset.



Kuva 7. Kuskin hallintapaneeli.

6.1 Responsiivisuus

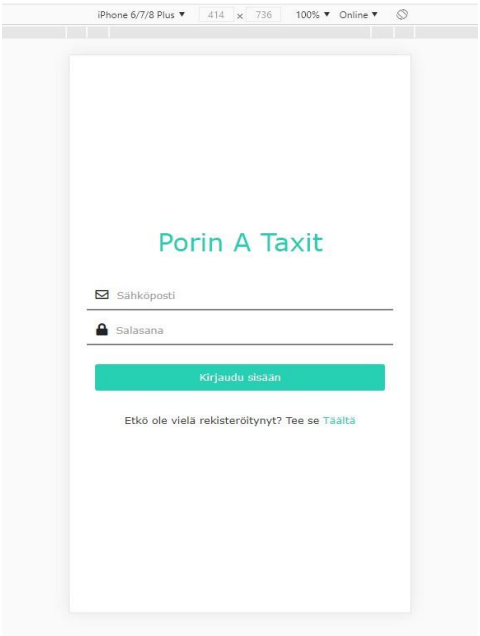
Sovelluksen käyttöliittymän suunnittelussa otettiin ensisijaisesti huomioon mobiiliystävällisyys eli käyttöliittymä suunniteltiin niin sanotusti ”mobile first” menetelmällä. Käytännössä tämä tarkoitti sitä, että jokainen sovelluksen sivu ja komponentti suunniteltiin ensin näyttämään hyvältä ja niin, että ne ovat responsiivisia pienilläkin näytöillä. Kun komponentti tai näkymä sopi hyvin eri puhelinleveyksille, siirryttiin vaiheittain isompiin näyttöihin. Oli tärkeää, että verkkosivu on responsiivinen, sillä oli

oletettavaa, että suurin osa käyttäjistä käyttää sovellusta puhelimella. Myös sovellusta käyttävät kuskit käyttäisivät sovellusta puhelimella.

Responsiivisuus saatiin toteutettua verkkosivulla täysin CSS:n avulla. Käyttämällä CSS:n ”breakpointteja” esimerkiksi ”@media only screen and (min-width: 768px)”, voidaan asettaa eri CSS tyylejä, kun päätelaitteen näytön koko on vähintään 768 pikseliä. Koska käyttöliittymä suunniteltiin ensisijaisesti mobiilinäkökulmasta, asetettiin nämä breakpointit käyttämään minimileveysasetusta. Sivulla käytettiin 768, 992 ja 1200 pikselin breakpointteja. Näin saatiin aseteltua sivulle tyylittelyjä myös tablettitietokoneille.

Käyttöliittymän responsiivisuuden testaaminen oli helppoa Google Chromen kehittäjän työkalujen avulla (DevTools). Työkaluista voidaan asettaa tietokoneen näyttö mukailemaan tiettyjen puhelinten tai tablettitietokoneiden näytön kokoa.

Kuvassa 8 on esimerkki responsiivisuuden testauksesta sovelluksen kirjautumissivulla. Kehittäjän työkaluista on asetettu näyttö mukailemaan iPhone 6/7/8 plus puhelimien näyttöjä.



Kuva 8. Sivun responsiivisuuden testaus kirjautumissivulla.

7 TIETOKANNAN SUUNNITTELU

Tietokantana käytettiin aiemmin mainittua Firebasea, joten itse tietokantaa tai esimerkiksi kirjautumis- tai rekisteröitymisjärjestelmiä ei tehty, sillä nämä kuuluvat Firebasen tarjontaan. Firebaseen oli kuitenkin lisättävä tarvittavat dokumenttikokoelmat, joita käytettiin tietojen tallentamiseen. Lisäksi tuli asettaa tiettyjä sääntöjä tietokannan hallitsemiseen

7.1 Kokoelmat

Sovellukseen luotiin neljä eri kokoelmaa: activeDrivers, drivers, orders ja users, eli aktiiviset kuskit, kuskit, tilaukset ja henkilöt. Henkilökokoelma ei ollut pakollinen tieto, sillä rekisteröityneet käyttäjät voidaan tallentaa ilman kokoelmaa, mutta henkilökokoelmaan saatiin kuitenkin tietoa, mikä helpottaa sovelluksen käyttöä. Henkilökokoelmaan lisättiin rekisteröinnin yhteydessä henkilön etunimi, sukunimi, sähköpostiosoite ja puhelinnumero. Näitä tietoja pystyttiin käyttämään uutta tilausta tehdessä, asettamalla esimerkiksi nimitiedot automaattisesti tilauslomakkeelle. Drivers tai kuskikokoelma oli hyvin samankaltainen kuin henkilökokoelma, mutta siihen lisättiin lisäkenttä kertomaan, onko kuski aktiivinen vai ei.

Orders tai tilauskokoelmaan lisättiin henkilöiden tilaukset. Yhteen tilausdokumenttiin lisättiin tilaajan haluama osoite, postinumero, etunimi, sukunimi, puhelinnumero, tilauksen aika ja tilauksen tilanne.

ActiverDrivers tai aktiiviset kuskit -kokoelma taas on yksi yksinkertainen dokumentti, joka kertoo käyttäjälle, onko aktiivisia kuskeja. Aktiiviset kuskit -kokoelma tehtiin, ettei käyttäjien tarvitse käydä jokaista kuskikokoelman dokumenttia läpi selvittääkseen onko aktiivisia kuskeja.

7.2 Firebase rules

Firebase tarjoaa mahdollisuuden asettaa sääntöjä tietokannan kokoelmiin. Nämä säännöt ovat hyvin tärkeitä, sillä ilman sääntöjen asettamista, kuka vain voi lukea tai jopa poistaa minkä tahansa dokumenttikokoelman. Sääntöjen asettaminen on kohtalaisen helppoa, kun käytetään Firebasen Firestorea. Firestoren sääntöjen hallintapaneeliin voidaan yksinkertaisesti kirjoittaa esimerkiksi ”Allow read: if request.auth != null;”. Tällä pystyttiin varmistumaan siitä, että pyynnön lähettäjä voi lukea dokumenttia tai dokumenttikokoelmaa, jos hän on kirjautunut sisään. Hallintapaneelissa voitiin myös tehdä yksinkertaisia funktioita, millä pystyttiin selvittämään, onko käyttäjä esimerkiksi kirjautunut sisään tai tilauksen ”omistaja”, ja näitä voitiin taas uudelleen käyttää sääntöjen eri osioissa.

Tässä projektissa sääntöihin asetettiin eri asetuksia, kuten esimerkiksi mahdollistettiin tilauksien lukeminen vain, jos kyseessä on tilauksen luoja tai kuski. Vaikka tietokantaan ei tallennettu sensitiivistä dataa, oli kuitenkin tärkeää saada estettyä tietojen vuotaminen.

Firestoren sääntöjä voidaan myös käyttää, ja on myös tärkeää, että niitä käytetään datan validoinnissa. Vaikka erilaisten lomakkeiden lähetyksen yhteydessä voidaan validoida lomakkeen osia. Esimerkiksi uutta taksia tilatessa validoitiin kaikki lomakkeen kentät, jotta käyttäjä ei voi asettaa vääränlaista tietoa eri kenttiin. Asettamalla Firestoren sääntöihin ”request.resource.data.Phone is number” voitiin validoida, että saapuvan datan ”Phone”, eli puhelinnumero, sisältää vain numeroita. Tai asettamalla ”request.resource.data.FirstName.matches(’[A-Öa-ö]’)”, varmistettiin että käyttäjän asettamassa etunimessä on vain kirjaimia. Nämä validoinnit tietysti tehtiin myös käyttöliittymän puolella, mutta siihen ei kuitenkaan voida luottaa täysin, sillä käyttöliittymässä tehdyt validoinnit ovat kierrettävissä.

8 TOIMINNALLISUUS

8.1 Kirjautuminen ja rekisteröityminen

Koska kyseessä oli verkkosovellus, rekisteröityminen ja sisäänkirjautuminen olivat oleellisessa osassa. Ilman sisäänkirjautumista ei sivulla pystynyt tekemään mitään. Rekisteröinti pysyttiin suorittamaan yksinkertaisella lomakkeella. Itse käyttäjän luomiseen ei tarvittu kuin sähköpostiosoite ja salasana, kun käytettiin Firebasen ”createUserWithEmailAndPassword(email, password)” -metodia. Asettamalla käyttäjän lisäämä sähköpostiosoite ja salasana metodin parametreiksi, Firebase rekisteröi käyttäjän ja näillä arvoilla oli heti mahdollista kirjautua sisään sovellukseen. Lomakkeen yhteyteen lisättiin kuitenkin vielä käyttäjän eri tietojen, kuten nimen ja puhelinnumeron, tallentaminen tietokantaan.

Sovellukseen kirjautuminen tapahtui myös hyvin samankaltaisesti käyttämällä tällä kertaa Firebasen ”signInWithEmailAndPassword(email, password)” -metodia. Lomakkeen ja kyseisen metodin avulla lähetetään tietokantaan käyttäjän sähköpostiosoite ja valitsema salasana ja jos tiedot vastaavat rekisteröityjä tietoja, palauttaa Firebase käyttäjälle JSON Web Tokenin ja käyttäjä kirjautuu sisään. Jos kirjautuminen onnistuu, niin käyttäjä ohjataan automaattisesti verkkosivun etusivulle.

8.2 Peruskäyttäjän ja kuskin erottelu

Peruskäyttäjän ja kuskin erottelu toteutettiin käyttämällä Firebasen mukautettuja autentikointi valtuuksia (Custom Tokens). Admin-käyttäjällä on mahdollisuus asettaa eri käyttäjille joko admin valtuudet tai driver eli kuskin valtuudet. Käyttämällä näitä valtuuksia, voidaan käyttöliittymä jakaa eri käyttäjille erilaiseksi. Esimerkiksi linkki kuskin ohjauspaneeliin saatiin näkyville navigaatiopalkkiin vain, jos kirjautunut käyttäjä on kuski käyttämällä Firebasen metodia ”getIdTokenResult”, mikä palauttaa kirjautuneen käyttäjän kustomoidut valtuudet, jos niitä on asetettu. Käyttäjä voitiin myös ohjata pois sivulta, jos hänellä ei ole tiettyjä valtuuksia. Näitä samoja

valtuuksia voitiin myös käyttää Firestoren säännöissä siten, että esimerkiksi vain kuskin valtuudet omaavat käyttäjät voivat lukea kaikkia tehtyjä tilauksia.

8.3 Vuex store ja tietokannan muutosten seuranta

Sovelluksessa käytettiin hyväksi Vuen tarjoamaa tilanhallinta-kirjastoa Vuex:ää. Peruskäyttäjän kirjautuessa sisään ja siirtyessä etusivulle, juostaan erilaisia storeen asetettuja metodeja. Storeen tallennetaan näiden metodien avulla käyttäjän tietoja Vuex:n tilaan, kuten nimet ja puhelinnumero, ja niitä voidaan täten helposti käyttää tilauslomakkeella ja asettaa näitä tietoja valmiiksi lomakkeen kenttiin. Kirjautumisen yhteydessä tarkistetaan myös, onko käyttäjällä aktiivinen tilaus. Vuex:n storeen lisättiin myös metodeja seuraamaan muutoksia käyttäjän tilauksessa ja aktiivisissa kuskeissa Firestoren reaaliaikaisen tietokannan avulla. Kun käyttäjä kirjautuu sisään, nämä metodit aktivoidaan ja ne seuraavat muutoksia tietokannassa. Kun esimerkiksi kuski siirtyy ajoon, saa sovellus siitä tiedon ja käyttöliittymää voidaan muokata halutulla tavalla ja mahdollisesti antaa myös ilmoitus käyttäjälle. Vuen reaktiivisuuden avulla voitiin esimerkiksi automaattisesti asettaa deaktivoitu taksin tilausnappi aktiiviseksi ja toimivaksi, kun activeDrivers-kokoelmassa huomataan muutoksia, ilman tarvetta päivittää sivua. Samoin tilauksen seurannassa voitiin tehdä käyttäjälle ilmoitus automaattisesti, kun taksi on hyväksytty ja kun kuski on saapunut paikalle hakemaan käyttäjää.

Kuskin puolen toiminnallisuus toteutettiin myös käyttämällä Vuex:ää ja Firestoren reaaliaikaista tietokantaa hyväksi. Kuskin siirtyessä ohjauspaneeliin, tarkistetaan ensin, onko kuski aktiivinen. Jos kuski on aktiivinen, niin haetaan mahdolliset uudet tilaukset ja käynnissä oleva tilaus, jos sellainen on. Käyttämällä reaaliaikaista tietokantaa voitiin uudet tilaukset päivittää heti, kun niitä saadaan, ilman tarvetta päivittää sivua.

8.4 Kuskin toiminnallisuudet

Kuskilla on erilaisia toiminnallisuuksia, mitä tarvitaan kuskin tilan sekä tilausten hallitsemiseen. Kuskin ohjauspaneelissa kuski voi siirtyä ajoon ja ajoista pois. Jos ajosta siirrytään pois, niin yksinkertaisesti tarkistetaan tietokannasta, onko muita aktiivisia kuskeja ja jos niitä ei ole, se asettaa aktiiviset kuskit kokoelman dokumentin epätodeksi. Siirryttäessä ajoon sama dokumentti asetetaan todeksi.

Kuskit pystyivät myös hallitsemaan tilauksia. Kuskilla on mahdollisuus hyväksyä sekä hylätä tilauksia. Hyväksytyn tilauksen tilaa voidaan myös niin sanotusti puskea eteenpäin, eli tilauksen hyväksymisen jälkeen voidaan asettaa tietokantaan tilauksen tilaksi saapunut paikalle ja tilaus suoritettu.

9 YHTEENVETO

Työn tavoitteena oli luoda sovellus, millä on mahdollista tilata takseja. Samalla sovellus oli tarkoitus saada käytettäväksi myös kuskeille, joilla on taas mahdollisuus hallita tilauksia. Opinnäytetyötä kirjoittaessani sovellus ei ollut vielä täysin valmis, mutta pääpiirteittäin onnistuin tavoitteissa. Käyttäjät voivat rekisteröityä, kirjautua ja lisätä tilauksia järjestelmään. Sain myös kuskin toiminnan toimimaan halutulla tavalla.

Alussa päätin, loisinko verkkosovelluksen vai puhelinsovelluksen ja lopulta päädyin verkkosovellukseen. Tämä vähensi työn määrää, sillä ei tarvinnut ottaa huomioon eri puhelimien käyttöjärjestelmiä, sovelluskauppojen rajoituksia tai vaatimuksia, eikä jouduttu luomaan erillistä verkkosivua. Näin koettiin verkkosovelluksen olevan parempi ratkaisu, varsinkin kun kyseessä oli vain yksittäinen projekti.

Pääteknologioiksi valitsin Vue.js, sekä Firebase. Front end teknologiaksi olisi luultavasti käynyt mikä tahansa moderni ja tuettu Javascript-kirjasto, mutta päädyin Vueen sen ollessa yksinkertaisempi esimerkiksi Reactiin verrattuna, sekä minulla oli siitä myös hieman kokemusta. Firebase taas tarjosi monia hyviä palveluita, kuten yksinkertaisen rekisteröintityökalun, mikä nopeutti ja helpotti työn etenemistä. Ilmaiseksi käytettynä Firebase kuitenkin asettaa tiettyjä rajoituksia järjestelmälle, mitkä oli otettava huomioon.

Sain mielestäni pidettyä käyttöjärjestelmän ja sen toiminnallisuuden yksinkertaisena, sekä sen toimivuuden ja responsiivisuusden eri laitteilla hyvänä. Responsiivisuuden luominen eri näytöille aiheutti lisätyötä, mutta nykypäivänä kuitenkin käyttäjät odottavat sovelluksien ja sivujen toimivan ja näyttävän yhtä hyvältä eri alustoilla.

LÄHTEET

Baez 2020. ’What is serverless architecture and when should you use it?’. Scalyr Blog 14.5.2020. Viitattu 15.11.2020 https://www.scalyr.com/blog/serverless-architecture/

Batschinski n.d. ’What is Firebase? The secrets unlocked…’. Back4App Blog. Viitattu 9.11.2020.
https://blog.back4app.com/what-is-firebase/

Clearbridge Mobile WWW-sivut 2020. Viitattu 30.9.2020.
https://clearbridgemobile.com/

Copes 2018. ’The Vue Handbook: a thorough introduction to Vue.js’. Freecodecamp Blog. Viitattu 21.10.2020.
https://www.freecodecamp.org/news/the-vue-handbook-a-thorough-introduction-to-vue-js-1e86835d8446/

Cromwell 2017. ’Between the wires: An interview with Vue.js createor Evan You’. Freecodecamp Blog. Viitattu 10.10.2020.
https://www.freecodecamp.org/news/between-the-wires-an-interview-with-vue-js-creator-evan-you-e383cbf57cc4/

Fisher 2018. ’What is serverless architecture? Pros, cons, and how to get started’. LearnToCodeWithMe Blog. Viitattu 15.11.2020.
https://learntocodewith.me/posts/serverless-architecture/

Flaviocopes WWW-sivut 2018. ’Vuex, the Vue.js State Manager’. Viitattu 31.10.2020. https://flaviocopes.com/

Full Scale WWW-sivut 2020. ’What is cross platform app developement’. Viitattu 30.9.2020.
https://fullscale.io/

Gore 2017. ’What’s the deal with Vue’s virtual DOM’. Vue.js Developers blog. Viitattu 17.10.2020.
https://vuejsdevelopers.com/2017/02/21/vue-js-virtual-dom/

Heddings 2020. ’How does Google’s Firebase realtime database work?’ CloudSavvy It Blog. Viitattu 15.11.2020.
https://www.cloudsavvyit.com/4763/how-does-googles-firebase-realtime-database-work/

ite wiki WWW-sivut n.d. ’Progressive web application (PWA) / Progressiivinen verkkosovellus’ Viitattu 23.9.2020
https://www.itewiki.fi/

Johnston 2020. ’A beginners guide to web application development’ Budibase Blog. Viitattu 23.9.2020
https://www.budibase.com/blog/web-application-development/

Khan 2018. ’The pros and cond of native apps’. Clutch Blog. Viitattu 30.9.2020.

Maribojoc 2020. ’A closer look at Vue Router’. Vue.js Developers Blog. Viitattu 31.10.2020.
https://vuejsdevelopers.com/2020/01/27/closer-look-at-vue-router/

Mozilla WWW-sivut 2020. ’Introduction to the DOM’. Viitattu 17.10.2020.
https://developer.mozilla.org/en-US/

Mroczkowska 2020. ’What is a mobile app? Aoo devekionebt basics for businesses’. The droid on roids Blog. Viitattu 30.9.2020.
https://careerfoundry.com/en/blog/web-development/what-is-the-difference-between-a-mobile-app-and-a-web-app/

Rakowski 2020. ’Deep dive into Vue state management’. Viitattu 31.10.2020.
https://vueschool.io/articles/vuejs-tutorials/deep-dive-into-vue-state-management/

Stevens 2018. ’What is the difference between a mobile app and a web app?’. Viitattu 23.9.2020.
https://careerfoundry.com/en/blog/web-development/what-is-the-difference-between-a-mobile-app-and-a-web-app/

Vue CLI WWW-sivut 2020. Viitattu 21.10.2020
https://cli.vuejs.org/

Vue.js WWW-sivut 2020. Viitattu 17.10.2020
https://vuejs.org/