Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

VILLE-MATIAS SÖDERHOLM

Selainpohjainen chat-ohjelma

Tekijä
Söderholm, Ville-Matias

Julkaisun laji
Opinnäytetyö, AMK

Päivämäärä
12/2020

Sivumäärä
31

Julkaisun kieli
Suomi

Julkaisun nimi
Selainpohjainen chat-ohjelma

Tutkinto-ohjelma
Tieto- ja viestintätekniikan koulutusohjelma

Opinnäytetyö käsittelee selainpohjaisen chat-ohjelman luomisen vaiheita ja siihen käytettyjä työkaluja. Tarkoituksena oli luoda selaimessa toimiva reaaliaikainen kommunikointiväline kahdenkeskisille keskusteluille moderneilla työkaluilla.

Opinnäytetyössä käydään läpi yleisesti mitä kaikkea tarvitaan selainohjelmoinnissa ja mihin näitä käytetään. Työssä käydään läpi myös luodun ohjelman toimintaa ja miten ohjelma on toteutettu käytetyillä välineillä.

Lopuksi mielipiteitä käytetyistä välineistä ja millaisissa tilanteissa välineet ovat hyviä valintoja käytettäväksi.

Asiasanat
Vue.js, Firebase, Chat

Author
Söderholm, Ville-Matias

Type of Publication
Bachelor's thesis

Date
12/2020

Number of pages
31

Language of publication:
Finnish

Title of publication
Web based chat program

Degree program
Degree Programme in Information and Communication Technology

This thesis addresses stages in creation of web based chat application and used tools. The goal was to create a web based realtime communication tool to users communicate with each other.

This thesis gives an overview what all are needed in web development and what are they used for. This thesis includes also a section that discusses about functionality of the created application and how it was created with the chosen tools.

The end of thesis addresses the experience using the chosen tools and personal opinions about what were the good things in the used tools and in what situations they good to choices to use.

Key words
Vue.js, Firebase, Chat

SISÄLLYS

1 JOHDANTO

Nykyään on monia eri palveluita, joilla käyttäjät voivat olla yhteyksissä toistensa kanssa. Osa näistä toimii omina sovelluksinaan ja osa toimii selaimessa. Nämä palvelut ovat tulleet tärkeäksi osaksi ihmisten kommunikaatiossa niin liikemaailmassa kuin henkilökohtaisissa kaverisuhteiden ylläpidossa. Varsinkin näinä nykyisinä hetkinä, kun COVID-19 virus jyllää maailmassa on tarve välttää mahdollisimman paljon ihmiskontakteja, niin viestintäpalvelut ovat tullet entistä tärkeämmiksi.

Kommunikointipalveluissa käytetään nykyään monia erilaisia viestintä tapoja. Joissain kuvat ovat pääviestinnän välineitä, toisissa puhelut tai videopuhelut ja toisissa taas normaalit viestit. Osa isoimmista palveluista saattavat myös tarjota näitä kaikkia tai useimpia näistä.

Työni tarkoituksena on läpikäydä selaimessa käytettävän chat-ohjelman kehittämistä moderneilla välineillä ja kertoa näistä yleisesti. Selaimessa toimivaan chat-ohjelmaan päädyin siitä syystä, että ohjelmaa olisi mahdollista käyttää kaikissa selaimen omaavissa laiteissa.

Chat-ohjelmassani on käyttäjien mahdollista käydä kahdenkeskisiä keskusteluja viesteillä. Ohjelmassani käyttäjät eivät voi suoraan keskustella kaikkien kanssa vaan käyttäjien pitää hyväksyä toisensa ystäväkseen aloittaakseen keskustelun keskenään. Työni idea syntyi omasta mietinnästä, miten oikein chat-ohjelmat toimivat ja mitä kaikkea niiden toimintaa varten tarvitsee tehdä.

2 OPINNÄYTETYÖSSÄ KÄYTETYT TEKNOLOGIAT

2.1 Työkalujen valinta

Valitsin Vue.JS yhdeksi ohjelmassani käytettäväksi työkaluksi. Vue.JS valinnan yksi syy oli, että sillä on mahdollista luoda helposti uudelleen käytettäviä komponentteja. Komponenteille voisi asettaa omat funktiot, datat ja ulkoasut. Tätä tulisin tarvitsemaan, kun loisin uudelleen käytettäviä komponentteja ja niiden datat ja funktiot koskisivat vain sitä tiettyä instanssia. Myöskin Vue.JS reaktiivisuus herätti mielenkiintoni, kun ei tarvitsisi itse päivittää muuttunutta dataa ulkoasulle vaan Vue.JS hoitaisi sen automaattisesti. Vue.JS mainostettiin, että siihen on otettu parhaimmat ominaisuudet Angularista ja Reactista, jotka ovat suosituimpia JavaScript frameworkkeja kirjoitushetkellä. Uskon tämän olevan tosi, koska Vue.JS on uusin suosituimmista frameworkeista ja siinä on todennäköisesti opittu Angularissa ja Reactissa tehdyistä virheistä. Syy Vue.JS valintaan oli myös se , että työpaikallani ollaan ottamassa tämä framework käyttöön ja voisin opetella sitä käyttämään tämän opinnäytetyön avulla.

Firebasen käyttöön päädyin testailuiden ja eri tapoihin tutustumisen kautta. Aluksi aloitin luomaan backendiä ohjelmalleni Djangon avulla. Django oli mielestäni hyvä, mutta kaikkien haluamieni ominaisuuksien toteuttaminen tuntui hyvin työläältä, kun jouduin Djangon käytön opettelun lisäksi jo opettelemaan Vue.JS käyttöä. Aluksi Djangoon päädyin siitä syystä, että minua kiinnosti tutustua Python ohjelmointikieleen. Django oli myös suosittu framework ja sitä oli käytetty suurissakin tunnetuissa ohjelmissa. Firebaseen törmäsin opetellessani Djangon käyttöä ja huomasin, että Firebase voisikin sopia hyvin ohjelmani backendiksi ja vähentäsi työn määrää. Firebaseen tutustuessa huomasin, että tarvitsemani asiat olivat Firebasessa yksinkertaisia käyttää ja minun ei tarvitsisi kehittää omaa ohjelmointirajapintaa niiden käyttöä varten. Firebasen helppo käyttöönotto vaikutti myös valintaan.

2.2 HTML

Hyper Text Markup Language (lyhenteeltään HTML) on Tim Berners-Leen luoma merkintäkieli. HTML on tunnettu verkkosivujen luontikielenä. HTML kielellä

kuvataan dokumentin sisällön rakennetta HTML tunnisteiden eli tägien avulla. (BitDegree 2019)

Tägeillä voidaan määrittää elementin tarkoitus, esimerkiksi onko elementti kuva vai onko se normaalia tekstiä tai jotain aivan muuta. Elementit aloitetaan aina aloitus tägeillä ja ne päättyvät usein myös lopetus tägiin, mutta eivät aina. Esimerkiksi kuvaa kuvaavalla tägillä <img> ei ole lopetus tägiä ollenkaan. Useimmiten kuitenkin elementeillä on molemmat tägit. (Mozilla 2020)

HTML tägit eivät rajoitu pelkästään elementin tarkoituksen määrittämiseen, vaan niillä voidaan myös määrittää ulkoasua. Esimerkiksi tägillä voidaan lihavoida tekstiä, joka on sijoitettuna aloitus ja lopetus tägin väliin. Kuitenkin suurimmaksi osaksi HTML dokumentin tyylitykset hoidetaan CSS-tyylityskielen avulla. Tägeillä voidaan myös luoda linkityksiä toisiin dokumentteihin. Esimerkisi tägin <a> avulla voidaan luoda linkki toiselle HTML dokumentille. (Moraes n.d)

HTML elementeille on myös mahdollista lisätä attribuutteja eli määritteitä. Attribuuttien tarkoitus on kertoa lisää tietoa elementeistä ja mahdollistaa uniikkien elementtien luonnin. Attribuutit lisätään aina elementin aloittavan tägin sisälle. Esimerkiksi aikaisemmin mainittu <a> tägi, eli hyperlinkkiä kuvaava tägi, ei toimi linkkinä lisäämättä siihen href attribuuttia, johon määritetään mihin dokumenttiin linkin pitäisi johtaa. Class ja id attribuuttien avulla voidaan määrittää elementille uniikkeja tunnisteita. Uniikkien tunnisteiden avulla voidaan esimerkiksi määrittää vain tätä elementtiä koskevat CSS-tyylitykset. Tämä tapahtuu viittaamalla elementille lisättyihin id tai class attribuutteihin. Uniikit tunnisteet auttavat myös JavaScriptillä tehtävien asioiden kohdentamisen tiettyyn elementtiin. (Jämsen n.d)

```
<html>
    <head>
        <title>Hello world</title>
    </head>
    <body>
        <p>
            Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
        </p>
    </body>
</html>
```

Kuva 1. Esimerkki HTML dokumentin rakenteesta

2.3 CSS

Cascading Styles Sheets (lyhenteeltään CSS) on tyylityskieli. CSS luotiin HTML dokumenttien tyylittämistä varten, koska HTML:än alkuperäinen tarkoitus ei ollut tyylitellä dokumentteja, vaikka sillä on tyyli-tägejä. CSS:än avulla voidaan määritellä miltä tietyt elementit näyttävät HTML dokumenteilla, esimerkiksi tekstin väri ja sen tausta. CSS:än avulla määritellään myös miten elementit asettuvat dokumentille. CSS:än avulla on mahdollista myös luoda animaatioita dokumenteille. CSS:n ansiosta HTML dokumenteista on mahdollista luoda visuallisesti näyttäviä ja eri näytön kokoille sopeutuvia. (Tutorials point n.d; W3 n.d; Mozilla 2020)

CSS-tyyli määrityksiä voidaan lisätä HTML dokumentille kolmella eri tavalla. Ulkoinen CSS (External CSS) eli linkittämällä ulkoinen CSS-tiedosto HTML dokumentille. Tämä tapahtuu lisäämällä HTML dokumentin <head> tägien sisälle <link> tägi, johon lisätään href attribuutti sisältäen CSS-tiedoston sijainnin. Toinen tapa on lisätä suoraan HTML dokumentille CSS-tyyli määritykset. Tämä tapahtuu lisäämällä HTML dokumentin <head> tägien sisälle <style> tägit, joiden sisälle lisätään CSS-tyyli määritykset. Tätä kutsutaan sisäiseksi CSS:äksi (Internal CSS). Kolmas tapa lisätä CSS-tyyli määritykset on lisäämällä ne suoraan HTML elementille. Tämä tapahtuu lisäämällä style attribuutti tyyliteltävälle elementille ja lisäämällä style attribuuttiin halutut CSS määritykset. Tätä kutsutaan inline CSS:äksi. Jos CSS-tyyli määritykset lisätään ulkoiselta tiedostolta tai ne määritetään <style> tägien sisällä niin tyylien määrittämiset tietyille elementeille tapahtuu viittaamalla elementin HTML tägiin, elementille määritettyihin class:seihin tai id:seen. (W3 n.d)

Se, että CSS tyylejä voidaan lisätä eri tavoilla aiheuttaa erilaisia hyötyjä ja haittoja. Ulkoisen CSS-tiedoston käyttämisen hyötyjä on, ettei HTML dokumenttia tarvitse muuttaa kun halutaan muuttaa dokumentin ulkonäköä ja jos samaa tyylitystä halutaan käyttää useammalla eri dokumentilla, niin ei tarvitse kirjoittaa useampaan dokumenteihin samoja CSS-tyyli määrityksiä. Myöskin, jos samoja tyylityksiä on käytetty useammalla dokumenteilla ja näihin on linkitetty sama CSS-tiedosto ja näitä kaikkia halutaan muuttaa, niin voidaan muuttaa vain CSS-tiedostoa. Ulkoisen CSS:än haittana on kuitenkin se, että HTML dokumentti renderöidään vasta CSS-tiedoston lataamisen jälkeen. Jos tyylitykset määritellään suoraan HTML dokumenttiin niin dokumentti renderöidään heti. Yleensä on kuitenkin järkevämpää luoda ulkoinen CSS-tiedosto helpottaakseen suurten tyyli määritys määrien ylläpitämistä ja muokkaamista.

```
<html>
    <head>
        <title>Hello world</title>
        <style>
            p {
                background-color: black;        Internal CSS
            }
        </style>
        <link rel="stylesheet" href="mycss.css">    External CSS link
    </head>
    <body>
        <p style="color: white;">          Inline CSS
            Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
        </p>
    </body>
</html>
```

Kuva 2. CSS:n lisäys tavat

```
p {
    font-size: large;
    font-family: Arial, Helvetica, sans-serif;
}
```

Kuva 3. Ulkoinen CSS tiedosto

2.4 JavaScript (JS)

JavaScript (JS) on yleisimmin verkkosivuilla käytetty oliopohjainen ohjelmointikieli. JavaScriptillä voi muokata verkkosivujen ulkonäköä tapahtumien perusteella, luoda pelejä, videosoittimia ja muita kompleksisia ominaisuuksia verkkosivuille. JavaScript on dynaamisesti tyypitetty kieli, mikä tarkoittaa sitä, että muuttujien tyypit päätetään koodin suorituksen aikana. JavaScript koodi ei tarvitse ollenkaan kääntäjää, vaan tietokone pystyy suoraan tulkitsemaan sitä ja tekemään koodin haluamat asiat. JavaScript koodi suoritetaan yleensä käyttäjän laitteessa, mutta JavaScript koodia on nykyään myös mahdollista suorittaa palvelimen puolella (Node.js). JavaScriptin käyttö ulottuu nykyään myös selainohjelmoinnin ulkopuolelle työpöytä- ja mobiilisovellusten kehitykseen. (Mozilla 2019; Mozilla 2020)

JavaScriptin ansiosta kaikkia verkkosivuilla tapahtuvia asioita ei enää tarvitse tehdä palvelimella, mikä nopeuttaa verkkosivujen toimintaa ja vähentää palvelimien kuormitusta. JavaScriptin avulla verkkosivuista saadaan paljon vuorovaikutteisempia, kun voidaan muokata verkkosivuja käyttäjän tekemien asioiden perusteella ja käyttäjälle voidaan myös heti antaa palautetta tekemilleen asioille. JavaScript mahdollistaa myös sen, ettei verkkosivuja tarvitse aina ladata uusiksi ulkonäköä päivittääkseen. Ilman JavaScriptiä selaimeen ei voitasi luoda monimutkaisia työpöytämaisia sovelluksia. (Tutorials point n.d; Morris 2012; Kantor 2020)

JavaScript koodin lisääminen verkkosivuille tapahtuu liittämällä se HTML dokumentin <script> tägien sisälle. JavaScriptille on myös mahdollista luoda oma tiedosto ja liittää se HTML dokumenttiin asettamalla tiedoston sijainnin <script> tägien sisälle. (Mozilla 2020)

2.5 Framework

Framework eli ohjelmistokehys on runko, jonka päälle kehittäjä rakentaa oman applikaationsa. Ohjelmistokehyksien tarkoitus on helpottaa ja nopeuttaa applikaatioiden kehitystyötä. Ohjelmistokehykset nopeuttavat ohjelmistokehitystä siinä, että niissä on valmiiksi kehitetty applikaatioille välttämättömiä perustoimintoja, jotka kehittäjän olisi muutenkin hyvä tai pakko ohjelmoida ja suunnitella ilman ohjelmistokehyksen

käyttöä. Valmiin ohjelmistokehyksen käytön muita hyötyjä on, varsinkin suosittujen, että ne ovat todennäköisesti kohtuu bugittomia, nopeita ja tietoturvallisia laajan käyttäjäkunnan takia, joten virheet ja ongelmat ovat todennäköisesti jo löydetty. Ohjelmistokehykset ovat yleensä luotu myös niin, että varsinkin laajoissa ohjelmistoissa ne selkeyttävät ja helpottavat koodin luettavuutta ja ylläpitoa. (Singh 2020; Zviagin 2020)

Ohjelmistokehyksiä löytyy monelle eri ohjelmointikielelle ja moneen eri käyttötarkoitukseen. Esimerkiksi työssäni käsiteltävä Vue.js on JavaScript ohjelmistokehys web front-end kehittämiseen.

2.6 Vue.JS

Vue.JS on avoimen lähdekoodin JavaScript framework eli ohjelmistokehys, jonka kehityksen aloitti Evan You. Ensimmäinen versio Vue.JS:tä julkaistiin vuoden 2014 helmikuussa. Vue.JS:n kehityksen takana ei ole mikään iso yritys niin kuin monilla muilla suosituilla JavaScript frameworkeillä esimerkiksi React.js (Facebook) ja Angular (Google) vaan Vue.JS:sää ylläpitää ja kehittää Vue:n yhteisö. (Caputa 2019)

Vue.JS käytetään front-end kehityksessä ja sillä.voidaan luoda single-page ja multi-page applikaatioita. Vue.js käytetään pääasiassa web front-end-kehityksessä, mutta sitä on mahdollista myös käyttää mobiilikehityksessä. (Mamani 2019)

Vue.JS etuihin kuuluvat sen hyvä skaalautuvuus ja helppo liitettävyys jo olemassa olevaan koodiin, eli käyttäjän ei tarvitse kirjoittaa uusiksi koko koodikantaa, jos haluaa ottaa Vue.JS käyttöön projektissa. Vue.JS on myös tiedostokooltaan pieni, koska kaikkia Vue.JS tarjoamia ominaisuuksia ei tarvitse ladata kerralla, jollei niitä tarvitse. Vue.JS:sän mukana tulee myös graafinen käyttöliittymä, jolla voidaan luoda vue-projektin pohjia, jollei halua näitä luoda komentoriviltä. (Vue.js n.d)

Vue.JS käyttää komponenttipohjaista arkkitehtuuria, eli Vue.JS:llä luodut applikaatiot koostuvat pienemmistä komponenteista. Esimerkiksi verkkosivulla jokin nappula voisi olla oma komponentti. Tämän arkkitehtuurin hyviä puolia on se, että valmiit

komponentit on helppo liittää useampaan paikkaan applikaatiossa ja tämän ansiosta ei tarvitse kirjoittaa samaa koodia useampaan paikkaan.
Vue komponentit sisältävät template osan, mikä määrittelee komponentin ulkonäön. Templatessa käytetään normaalia HTML:llää ja CSS:ää. Tämä helpottaa Vue.JS oppimista varsinkin, jos web-kehittäminen on jo tuttua. Komponenttiin voi sisältyä myös omia funktioita, nämä lisätään komponentilla olevien <script> tägien sisälle. Komponentille on mahdollista lisätä myös watcher funktioita, jotka ajetaan aina kun komponentille annettava data muuttuu. Komponenteille voi myös lisätä funktiota mitkä suoritetaan komponentin tietyssä elinkaaren kohdassa, esimerkiksi kun komponentti luodaan. Komponentilla voi olla myös omia muuttujia, joihin voidaan asettaa dataa. Muuttujiin voidaan viitata suoraan templatella, mikä näyttää muuttujassa sisältävän datan ulkoasulla. Vue.JS reaktiivisuuden ansiosta ulkoasulla näytettävä data päivittyy automaattisesti muuttujan datan muuttuessa. (Vue.js n.d)

Alla olevassa koodiesimerkissä komponentilla näytetään aluksi yksi nappula ja arvo 0. Kun nappulaa painetaan, niin suoritetaan funktio incrementCount, joka korottaa komponentilla olevaan count muuttujaa aina yhdellä. Count muuttujaan on viitattu komponentin templatessa, joten muutos päivittyy suoraan komponentin ulkoasulle.

```
<template>
    <div>
        <button v-on:click="incrementCount">Click me</button>
        <p>{{count}}</p>
    </div>
</template>

<script>
export default {
  data() {
    return {
     count: 0
    };
  },

  methods: {
      incrementCount() {
        this.count++;
      }
  }
};
</script>

<style>
    p {
        color: red;
    }
</style>
```

Kuva 4. Vue.js komponentti esimerkki

2.7 Firebase

Firebase on Googlen kehittämä backend-as-service palvelu eli BaaS, joka toimii pilvessä. Backend-as-service on palvelu, joka tarjoaa valmiiksi luodun backendin, jonka kehittäjä voi liittää oman ohjelmaansa tallentamaan esimerkiksi tietoa. Firebase tarjoaa käyttäjän autentikoinnin, reaaliaikaisen tietokannan, pilvi funktioiden luonnin, joilla voi luoda kustomoituja toimintoja Firebaseen, housting palveluita, koneoppimista, tallennustilaa ja erilaisia analytiikkaa esimerkiksi ohjelman kaatumisesta. (Esplin 2016; Stevenson 2018; Sutch 2020)

Firebasen tarjoamat tietokannat ovat NoSQL-tietokantoja. Firebasen tietokannassa luodaan kokoelmia, joihin lisätään dokumentteja, jotka sisältävät tietoa ja ne saattavat

myös sisältää toisia kokoelmia. Esimerkiksi voisi olla ”users” kokoelma, joka sisältää käyttäjä dokumentteja. Dokumentit sisältäisivät taas tiettyjen käyttäjien tietoja. (Stevenson 2018)

Firebasea käyttäessä ei tarvitse pystyttää eikä ylläpitää omia palvelimia vaan backend pyörii Googlen palvelimilla. Firebasea käyttäessä käyttäjän ei myöskään tarvitse kehittää omaa ohjelmistorajapintaa, jolla keskustella backendin kanssa. Firebase tarjoaa myös valmiita metodeja, joilla voi käyttää Firebasen tarjoamia palveluita. Firebase on siis helppo ottaa käyttöön, kun ei tarvitse kehitellä omia ratkaisuja hallitakseen ja keskustellakseen Firebasen kanssa. (Esplin 2016)

Firebasen palveluiden hallinnointi tapahtuu selaimessa toimivalla Firebase konsolilla, jonka kautta voidaan ottaa uusia palveluita käyttöön ja hallinnoida tietokantoja ja säätää niiden oikeuksia. (Google n.d)

Firebase voi käyttää iOS, Android, web, Unity C++ sovelluksissa. Kaikkia palveluita tosin ei ole saatavilla kaikille alustoille.

2.7.1 Tietokannat

Tietokanta on järjestelty ja jäsennelty kokoelma dataa. Tietokantaan voidaan lisätä uutta dataa, muuttaa olemassa oleva dataa, poistaa dataa ja sieltä voidaan noutaa dataa. Tietokannat on luotu helpottamaan suuren data määrän nopeaa hakua, käsittelyä ja säilyttämistä. Tietokantoja on monia erityyppisiä, suosituimpia näistä ovat relaatiotietokannat ja NoSql-tietokannat. (Javatpoint n.d; Oracle 2020)

Relaatiotietokannoissa käytetään taulurakenteeseen perustuvaa tietorakennetta. Data siis organisoidaan tauluihin, sarakkeisiin ja riveihin. Esimerkiksi voisi olla ”users” taulu, josta löytyvät kaikkien verkkokaupan asiakkaiden tiedot. Sarakkeissa voisi olla esimerkiksi asiakkaan osoite ja nimi. Eri rivit olisivat eri asiakkaita. Relaatiotietokannoissa tietokannan rakenne määritellään etukäteen. Tätä voidaan toki muuttaa myöhemminkin, mutta tällöin se on haastavampaa. Relaatiotietokannoissa pyritään välttämään saman datan esiintymisen useammalla eri taululla. Relaatiotietokantoja hallitaan

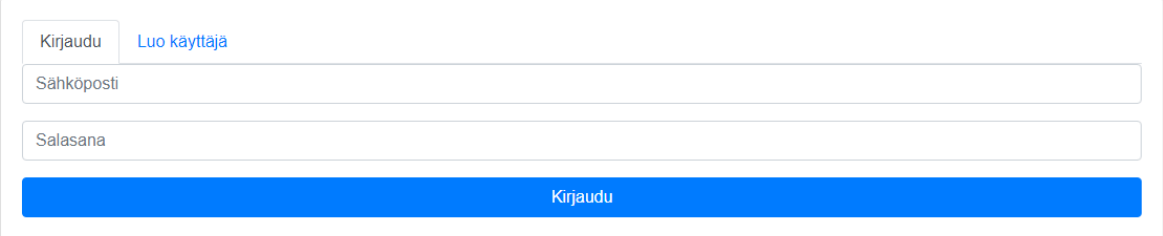
SQL-kielellä. SQL:llä muokataan dataa, lisätään dataa ja luodaan kyselyt, joilla noudetaan dataa. Jos halutaan noutaa dataa useammasta eri taulusta samassa kyselyssä niin SQL kielen avulla voidaan yhdistää eri tauluja asetetuilla ehdoilla. (Javatpoint n.d; Konkka 2016)

NoSQL-tietokantoihin voidaan tallentaa tietoa millaisessa rakenteessa tahansa. NoSQL-tietokannoissa ei vältetä saman tiedon tallentamista useampaan paikkaan niin kuin relaatiotietokannoissa. Näiden ansioista NoSQL-tietokannat ovat suorituskykyisiä, kun ei tarvitse yhdistellä eri tauluja. NoSQL-tietokannat skaalautuvat myös paremmin kuin relaatiotietokannat. NoSQL-tietokantojen datan rakennetta voidaan muuttaa helposti. (Lauren Schaefer n.d; Konkka 2016)

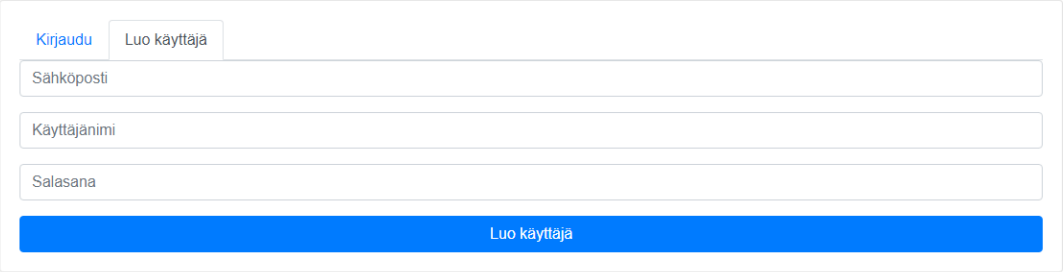
3 OHJELMAN KÄYTTÖ

Ohjelmani tarkoitus on mahdollistaa reaaliaikaisen keskustelun kahden henkilön kesken. Henkilöiden pitää ensiksi hyväksyä toisensa ystäväkseen, jotta pystyisivät keskustella toistensa kanssa.

Ensiksi ohjelmaan pitää luoda käyttäjä ja kirjautua tällä sisälle aloittaakseen ohjelman käytön. Ohjelma tarkastaa, että käyttäjä on kirjautunut, muuten käyttäjä ei pääse muille sivuille kuin kirjautumissivulle.

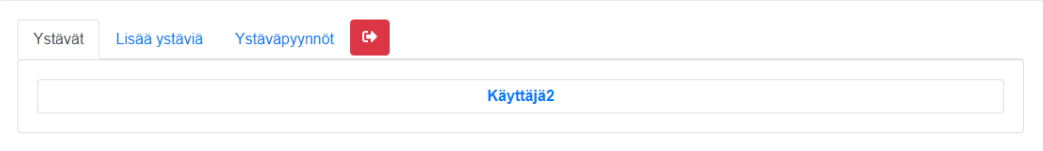


Kuva 5. Kirjautuminen



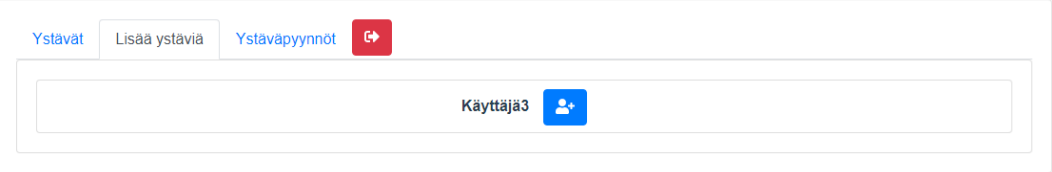
Kuva 6. Käyttäjän luonti

Kun käyttäjä kirjautuu sisälle, niin avautuu näkymä, jossa näkyy kaikki henkilön ystävät, joiden kanssa käyttäjä voi aloittaa keskustelun.

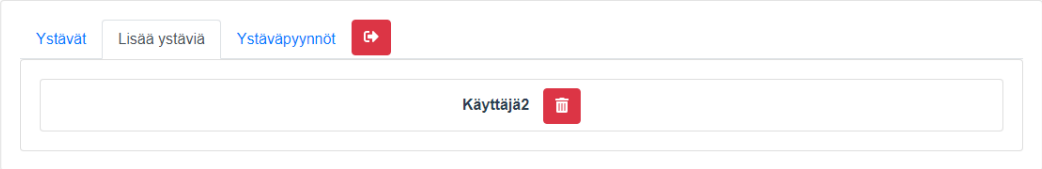


Kuva 7. Ystävät-näkymä

”Lisää ystäviä”-nappulaa painaessa avautuu näkymä, jossa näkyy kaikki ohjelmaan luodut käyttäjät, joita sisäänkirjautunut ei ole lisännyt ystäväkseen. Käyttäjä voi halutessaan lähettää haluamalleen henkilölle ystäväpyynnön painamalla käyttäjän nimen viereisestä sinisestä nappulasta, jossa on lisää ystäväksi ikoni. Tällöin ystäväpyynnön vastaanottajan ystäväpyynnöt-näkymälle tulee näkyville ystäväpyyntö. Kun käyttäjä on lähettänyt jollekin käyttäjälle ystäväpyynnön, tulee vastaanottajan nimen viereen punainen roskakorinappula. Tätä painamalla voidaan peruuttaa lähetetty ystäväpyyntö. Ystäväpyyntö katoaa tällöin vastaanottajan ystäväpyynnöt listalta.

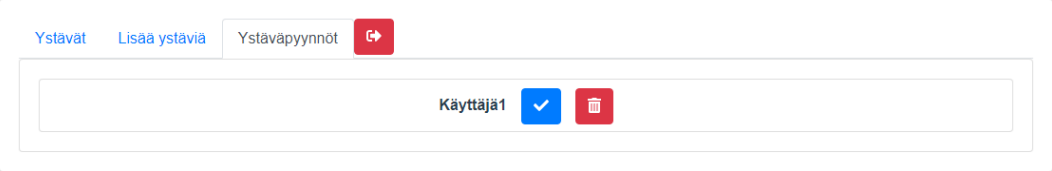


Kuva 8. Lisää ystäväksi näkymä



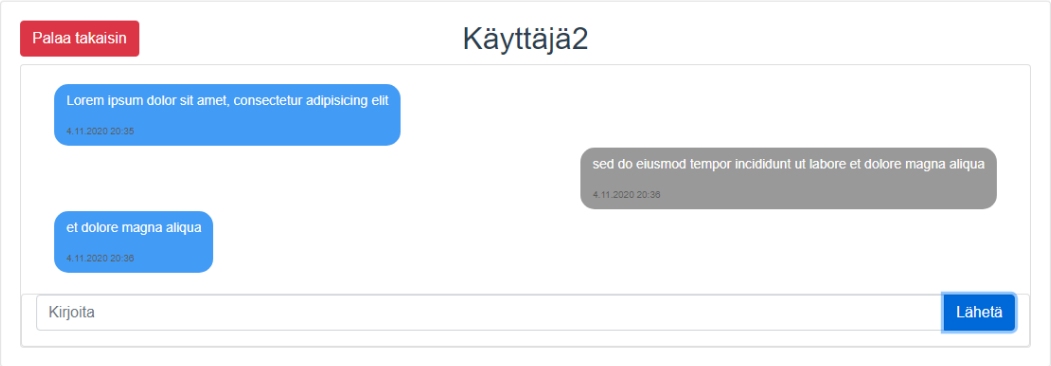
Kuva 9. Ystäväpyyntö lähetetty

Ohjelman yläpalkissa näkyy myös ystäväpyynnöt-nappula ja tätä painamalla avautuu näkymä, jossa näkyy kaikki käyttäjälle lähetetyt ystäväpyynnöt. Käyttäjä voi halutessaan hyväksyä ystäväpyynnön painamalla sinistä hyväksy-nappulaa lähettäjän nimen vieressä. Kun käyttäjä hyväksyy ystäväpyynnön niin molempien käyttäjien ystävät-näkymälle ilmestyy toistensa nimet ja he voivat aloittaa keskustelun toistensa kanssa. Jos käyttäjä ei halua hyväksyä ystäväpyyntöä, niin käyttäjä voi poistaa ystäväpyynnön painamalla punaista roskakorinappulaa.



Kuva 10. Ystäväpyynnöt-näkymä

Ystävät-näkymällä ystävän nimeä painaessa ohjelma siirtyy keskustelunäkymään, jossa käyttäjä voi keskustella reaaliaikaisesti kyseisen henkilön kanssa kahden kesken. Käyttäjän lähettämät viestit näkyvät sinisellä pohjalla ja keskustelukumppanin näkyvät harmaalla pohjalla. Viestikuplan alaosassa näkyy viestin lähetysaika. Keskustelunäkymän alaosassa näkyy laatikko, johon käyttäjä voi kirjoittaa haluamansa viestin. Käyttäjän painaessa enter tai lähetä-nappulaa viesti lähetetään ja tämä ilmestyy keskustelukumppanin keskustelunäkymälle näkyville. Keskustelunäkymän vasemmassa yläreunassa on punainen ”Palaa takaisin”-nappula, jota painamalla käyttäjä pääse takaisin Ystävät-näkymälle.



Kuva 11. Keskustelu-näkymä

4 OHJELMAN TOTEUSTUS

4.1 Käyttäjän luonti

Päädyin käyttämään ohjelmassani Firebasen valmista käyttäjän autentikointia. Firebase tarjoaa monia valmiita autentikointimenetelmiä. Firebasen tukemia autentikointimenetelmiä ovat esimerkiksi sähköposti ja salasana autentikointi, puhelin autentikointi ja muiden palveluiden tunnuksilla autentikointi. Päädyin käyttämään ohjelmassani sähköposti ja salasana autentikointia, koska useimmilla henkilöillä on jo käytössään sähköpostiosoite ja käyttäjän ei tarvitsisi luoda applikaatiotani varten käyttäjää muuhun palveluun. Firebasen autentikoinnissa olisi ollut myös mahdollista sallia monen eri kirjautumismenetelmän yhtäaikaisen käytön, mutta mielestäni pelkkä sähköposti ja salasana autentikointi on riittävä applikaatiotani varten.

Uuden käyttäjän luomiseen applikaatioon loin funktion nimeltä signUp, jota kutsutaan, kun käyttäjä on syöttänyt tarvittavat tiedot ja painaa Luo käyttäjä nappulaa. SignUp funktiossa kutsutaan Firebasen createUserWithEmailAndPassword metodia, jolle annetaan argumenteiksi email ja password muuttujat, joihin on tallennettuna käyttäjän syöttämät sähköpostiosoite ja salasana. Email ja password muuttujat ovat Vue komponentin data muuttujia. Sähköposti ja salasana inputeille asetin v-model=”muuttujan nimi” attribuutin, joka määrittää mitä komponentin muuttujaa päivitetään, kun inputtiin syötetään tekstiä. Kun CreateUserWithEmailAndPassword metodi suoritetaan se

tarkistaa annettu sähköpostiosoite, että täyttääkö validin sähköpostiosoitteen kriteerit ja ettei sähköpostiosoitetta ole jo liitetty jo luodulle käyttäjälle. Metodi tarkistaa myös, että annettu salasana on vähintään 6 merkkiä pitkä. Jos kriteerit eivät täyty niin käyttäjän luonti epäonnistuu ja metodi palauttaa virheen. Jos sähköpostiosoite on jo käytössä metodi palauttaa virheeksi ”auth/email-already-in-use” ja jos salasana on liian heikko niin virheeksi tulee ”auth/weak-password”. Virhe napataan catch:illa. Virheen syy tutkitaan ja virheestä ilmoitetaan käyttäjälle asettamalla errorSign muuttujaan virhettä kuvaava teksti, joka ilmestyy näkyville luo käyttäjä sivulle. Virheviesti ilmestyy automaattisesti komponentille näkyviin Vue.JS reaktiivisuuden ansiosta, koska errorSign muuttujaan on viitattu komponentin template osassa. Jos käyttäjän luonti onnistuu niin metodi palauttaa objektin, mikä sisältää tietoja luodusta käyttäjästä. Objekti sisältää esimerkiksi tiedon käyttäjälle luodusta uid:sta eli unique identifier:ista, joka on uniikki tunnistetieto. Tällä voidaan tunnistaa juuri tietty käyttäjä, vaikka olisi useampia saman nimisiä käyttäjiä.

SignUp funktiossa luodaan myös Firestoreen eli Firebasen tietokantaan ”users” kokoelma, johon luodaan jokaiselle luodulle käyttäjälle oma dokumentti. Dokumentin nimeksi asetetaan createUserWithEmailAndPassword metodin palauttamalta objektilta saatu käyttäjän uid, joka varmistaa, ettei samannimistä dokumenttia ole samassa kokoelmassa. Tämä myös helpottaa oikean käyttäjän tietojen hakemista, kun ei tarvitse käydä useampaa dokumenttia läpi löytääkseen halutun käyttäjän dokumentin vaan voidaan suoraan hakea dokumenttia käyttäjän uid:lla. Käyttäjälle luodulle dokumentille lisätään käyttäjän käyttäjänimi, sähköpostiosoite ja uid tunniste. Tiedot tallennetaan siitä syystä, että kaikkien käyttäjien tiedot olisivat myös käytettävissä, kun ollaan kirjauduttu muille käyttäjille. Esimerkiksi kun halutaan liittää käyttäjät toistensa ystäväksi, niin pitää tietää molempien käyttäjien uid:t, jotta voidaan varmasti liittää oikeat käyttäjät toistensa ystäväksi.

Kun luodun käyttäjän tiedot on tallennettu tietokantaan niin käyttäjä ohjataan ystävät-näkymälle. Tämä tapahtuu this.$router.replace({ name: ”Chats”}); avulla, joka antaa Vue:n routerille tiedon mihin komponentille pitäisi siirtyä seuraavaksi. Tässä tapauksessa Chats nimiselle komponentille, joka on ystävät-näkymä.

```
async signUp() {
  // tries to create new user to firebase
  try {
    await firebaseApp
      .auth()
      .createUserWithEmailAndPassword(this.email, this.password)
      .then(cred => {
        // if user creation succeed new user will be added to firebase users collection
        return firestore
          .collection("users")
          .doc(cred.user.uid)
          .set({
            username: this.username,
            email: this.email,
            uid: cred.user.uid
          })
          .catch(function(e) {
            console.log(e);
          });
      });
    this.$router.replace({ name: "Chats" });
  } catch (e) {
    // shows errors in user creation
    if (e.code === "auth/email-already-in-use") {
      this.errorSignin = "Sähköposti on jo käytössä";
    } else if (e.code === "auth/weak-password") {
      this.errorSignin = "Salasanan pitää olla vähintään 6 merkkiä pitkä";
    }
  }
}
```

Kuva 12. signUp funktio

4.2 Kirjaudu sisään

Sisään kirjautumiselle loin funktion nimeltä login. Funktiota kutsutaan, kun käyttäjä painaa kirjaudu nappulaa. Funktion sisällä kutsutaan Firebasen metodia signInWithEmailAndPassword. Tällä metodilla pystytään kirjautumaan applikaatioon sisälle. Metodille annetaan argumenteiksi muuttujat email ja password mitkä sisältävät käyttäjän syöttämät sähköpostiosoite ja salasana. Näihin muuttujiin päivittyvät tiedot samalla tavalla kuin käyttäjän luonnissa. Metodi tarkistaa aluksi onko sähköpostiosoitteella luotu käyttäjä jo applikaatioon ja onko syötetty salasana ja sähköposti kombinaatio oikein. Jos kirjautumista ei hyväksytä metodi palauttaa virheen, mikä tarkastetaan catchissa. Jos virhekoodi on joko auth/user-not-found eli sähköpostilla ei ole luotu käyttäjää tai auth/wrong-password eli salasana on väärin, niin annetaan errorLogin muuttujalle teksti, jossa ilmoitetaan virheestä. Kun muuttuja saa virhetekstin niin tämä näytetään heti kirjaudu sisään-näkymässä. Jos käyttäjän syöttämät sähköposti ja

salasana ovat oikein, niin käyttäjä siirretään ystävät-näkymälle käyttämällä jälleen this.$router.replace({ name: ”Chats”});.

```
async login() {
  // tries to log in to firebase
  try {
    await firebaseApp
      .auth()
      .signInWithEmailAndPassword(this.email, this.password);
    this.$router.replace({ name: "Chats" });
  } catch (e) {
    // shows errormessage
    if (
      e.code === "auth/user-not-found" ||
      e.code === "auth/wrong-password"
    ) {
      this.errorLogin = "Sähköposti tai salasana on väärin";
    }
  }
},
```

Kuva 13. login funktio

4.3 Näkymältä toiselle siirtyminen

Näkymältä toiselle siirtymiseen käytin Vue:n router lisäosaa, jolla ohjataan komponenteille siirtymisiä. Vue router tiedostossa määritellään kaikki mahdolliset komponentit, joille voidaan siirtyä ohjelmassa.

Router tiedostoon voidaan myös luoda navigation guard:iksi kutsuttu tarkistus, joka ajetaan aina kun yritetään siirtyä toiselle komponentille. Tällä voidaan estää komponenteille siirtyminen tietyissä tilanteissa. Esimerkiksi applikaatiossani piti käyttää tätä, koska applikaatiossani haluttiin sallia vain kirjautuneiden käyttäjien siirtymien tietyille komponenteille, joten loin tarkistuksen navigation guard:in avulla kirjautumattomien käyttäjien komponenteille siirtymisen estämiseksi. Kirjautumattomana käyttäjä pääsee ainoastaan sisäänkirjautumiskomponentille. Asetin myös toisen tarkistuksen navigation guard:ille, koska pitää myös tarkastaa voiko käyttäjä siirtyä tietylle keskustelulle. Käyttöliittymällä ei ollut mahdollista siirtyä toisen käyttäjän keskusteluihin, mutta jos tiesi keskustelun uid:n niin oli mahdollista siirtyä muiden käyttäjien

keskusteluihin lisäämällä uid url:iin. Tämä piti estää, koska keskusteluun kuulumattomien ei pidä nähdä toisten henkilöiden keskusteluja.

Kun halutaan applikaatiossa siirtyä toiselle komponentille niin koodin lisätään this.$router.replace({ name: 'komponentin nimi'}); , mikä ilmoittaa routerille mihin komponentille pitää siirtyä jos navigation guard sallii tämän.

```
router.beforeEach((to, from, next) => {
  const requiresAuth = to.matched.some(record => record.meta.requiresAuth);
  const isAuth = firebaseApp.auth().currentUser;

  if (requiresAuth && !isAuth) {
    // if user is not signed in return to auth
    next("/");
  } else {
    // checks if if conversation_id is real and user belongs to conversation then allow user to go
    // else return to chats
    if (to.name == "Chat") {
      firestore
        .collection("conversations")
        .doc(to.params.id)
        .get()
        .then(function(doc) {
          if (
            doc.exists &&
            (doc.data().user_a === isAuth.uid ||
              doc.data().user_b === isAuth.uid)
          ) {
            next();
          } else {
            next("/chats");
          }
        });
    } else {
      next();
    }
  }
});
```

Kuva 14. Navigation guard

4.4 Ystävät-näkymä

Firebasella on kaksi tapaa, joilla voidaan noutaa dataa Firestore tietokannasta, get ja onSnapshot metodit.

Get metodin avulla voidaan noutaa viimeisintä dataa tietokannasta aina kun metodia kutsutaan. Tämä ei siis hae dataa uusiksi, jos tietokannan tietoihin tulee muutoksia noudon jälkeen. Ystävät-näkymällä olevalle ystävät-listalle on mahdollista tulla uusia ystäviä milloin vain, joten get metodia ei voisi suoraan käyttää. Jos haluaisin käyttää get metodia tässä tilanteessa, niin pitäisi luoda funktio, joka tarkastaisi tietyn ajan

välein onko tietokannan tietoihin tullut muutoksia. Jos on tullut muutoksia, niin silloin suoritettaisiin get metodi uuden datan noutamiseksi. Alla olevassa kuvassa on esimerkki get() metodin käytöstä, jossa ”users” kokoelmaan tallennetusta dokumentista haetaan dataa ja se asetetaan loggedUser muuttujaan.

```
firestore
  .collection("users")
  .doc(firebaseApp.auth().currentUser.uid)
  .get()
  .then(function(doc) {
    self.loggedUser = doc.data();
  })
```

Kuva 15. get metodi esimerkki

OnSnapshot metodilla noudetaan reaaliaikaisesti aina uudet tiedot tietokannasta, kun tietokantaan tulee muutoksia. Tämän metodin käyttö sopii hyvin ystävät-listan datan noutamiseen, koska lista voi päivittyä milloin tahansa. Alla olevassa kuvassa haetaan onSnapshot metodilla kirjautuneen käyttäjän dokumentilla olevalta ”friends” kokoelmalta kaikki käyttäjän ystävät ja ystäväpyynnöt. Jos ystävä on pending tilassa, niin se lisätään friendRequest listalle, johon lisätään ystäväpyynnöt. Jos ystävä on accepted tilassa niin tällöin se lisätään friends listaan, joka sisältää kaikki käyttäjän ystävät. Jos ”friends” kokoelmaan tulisi muutoksia niin onSnapshot metodi hakisi uudet tiedot automaattisesti ja lisäisi ne uusiksi haluttuihin listoihin.

```
// Load friend requests and friends
firestore
  .collection("users")
  .doc(firebaseApp.auth().currentUser.uid)
  .collection("friends")
  .onSnapshot(function(querySnapshot) {
    self.friends = [];
    self.friendRequests = [];
    querySnapshot.forEach(function(doc) {
      var data = doc.data();
      // if friend is pending push it to friendRequest array
      if (data && data.status === "pending") {
        self.friendRequests.push(doc.data());
        // if friend is accepted push it to friend array
      } else if (data && data.status === "accepted") {
        self.friends.push(doc.data());
      }
    });
  });
```

Kuva 16. onSnapshot metodi

Kun datat lisätään friendRequest ja friends listoille niin komponenttien template:illa eli ulkoasuilla, joissa dataa käytetään, käydään tallennetut datat läpi ja näiden avuilla luodaan uudet elementit, jotka kuvaavat ystävää ystävät-listalla tai ystäväpyyntöä ystäväpyynnöt-listalla. Vue:n reaktiivisuuden ansiosta, kun data muuttuu komponentin muuttujalla, niin muutos näkyy heti myös komponentin ulkoasulla. Komponenttiin ei siis tarvitse luoda toimintoa päivittämään näkymää, kun muuttujan data muuttuu. Alla olevassa kuvassa näkyy, miten ystävät-lista on toteutettu ystävät komponentissa. Router-link tägin avulla luodaan linkki, jonka avulla applikaatio pystyy siirtymään tietyllä keskustelunäkymälle linkkiä painettaessa. Linkki johtaa keskustelunäkymälle, johon kuuluvat kirjautunut käyttäjä ja elementin ystävä. Router-link tägille on määritetty mihin komponentille halutaan siirtyä, kun tätä painetaan.

```
<div class="friends">
  <div v-for="friend in friends" :key="friend.uid">
    <router-link :to="{ path: 'chat/' + friend.chat_uid }">
      <div class="card friend-box">
        <strong>{{ friend.name }}</strong>
      </div>
    </router-link>
  </div>
```

Kuva 17. Ystävä elementti

4.5 Ystäväpyynnön lähettäminen

Mahdolliset henkilöt, joille voidaan lähettää ystäväpyyntö, haetaan myös aikaisemmin mainitulla onSnapshot metodilla, koska uusia käyttäjiä voidaan luoda koska vain. Kun applikaatioon luodut käyttäjät on noudettu, niin niistä luodaan käyttäjäelementtejä lisää ystäviä komponentin template osassa. Tämä tehdään lähes samalla tavalla kuin ystävät komponentilla. Ainoat erot ovat, että elementille lisätään nappula, josta voidaan lähettää ystäväpyyntö ja elementiltä puuttuu linkki keskustelunäkymälle.

```
<template>
  <div class="card">
    <div class="fr-box">
      <strong class="fr-name">{{ friendRequest.name }}</strong>
      <button
        v-on:click="acceptFriendRequest(friendRequest)"
        class="btn btn btn-primary fr-button"
      >
        <font-awesome-icon icon="check" />
      </button>
      <button
        v-on:click="removeFriendRequest(friendRequest)"
        class="btn btn-danger fr-button-remove"
      >
        <font-awesome-icon icon="trash-alt" />
      </button>
    </div>
  </div>
</template>
```

Kuva 18. Käyttäjä elementti

Kun käyttäjä painaa elementillä henkilön nimen viereisestä lisää ystäväksi nappulasta niin silloin kutsutaan sendFriendRequest niminen funktio. Funktio luo molempien osapuolien ”users” kokoelman käyttäjien dokumentteihin ”friends” kokoelman, johon tallennetaan kaikki käyttäjän lähettämät ystäväpyynnöt, vastaanottamat ystäväpyynnöt ja ystävät.

Lisätäkseni uutta dataa tietokantaan käytin Firestoren set metodia, jolla voidaan lisätä tai päivittää dataa määritetyssä dokumentissa. Set metodi osaa myös luoda uuden kokoelman ja dokumentin määritetyn polun avulla, jollei näitä ole jo luotu valmiiksi.

```
sendFriendRequest(user, loggedUser) {
  // Add receiver to sender's friends collection
  const ref = firestore
    .collection("users")
    .doc(firebaseApp.auth().currentUser.uid)
    .collection("friends")
    .doc(user.uid);
  ref.set({
    name: user.username,
    status: "requested",
    uid: user.uid
  });
  // Add sender to receiver's friends collection
  firestore
    .collection("users")
    .doc(user.uid)
    .collection("friends")
    .doc(firebaseApp.auth().currentUser.uid)
    .set({
      name: loggedUser.username,
      status: "pending",
      uid: firebaseApp.auth().currentUser.uid
    });
},
```

Kuva 19. sendFriendRequest funktio

4.6 Viestien lähetys

Kun käyttäjä hyväksyy ystäväpyynnön, niin samalla luodaan uusi dokumentti ”conversations” kokoelmaan. Conversations kokoelma sisältää kaikkien käyttäjien keskustelut. Dokumenttiin lisätään keskusteluun kuuluvien käyttäjien uid:t, jotta voidaan tarkastaa router:issa olevalla navigation guard:illa, että kuuluuko kirjautunut käyttäjä keskusteluun yrittäessään avata tätä. Loin käyttäjille yhteisen keskustelu dokumentin siitä syystä, ettei tarvitsisi päivittää useampaa dokumenttia viestin lähetyksessä.

Conversations kokoelmaan luotuun keskustelu dokumenttiin lisätään myös messages kokoelma, joka sisältää kaikki keskustelussa lähetetyt viestit. Jokainen lähetetty viesti on oma dokumentti messages kokoelmassa. Viesti dokumentti sisältää viestin lähettäjän uid:n, viestin sisällön ja lähetysajan.

Kun käyttäjä on keskustelunäkymällä ja kirjoittaa viestin viesti-inputtiin ja painaa lähetä nappulaa niin silloin suoritetaan sendMessage niminen funktio. Funktio poistaa syötetyn tekstin edestä ylimääräiset välilyönnit, etteivät käyttäjät voisi lähettää pelkkiä välilyöntejä sisältäviä viestejä. Tämän jälkeen tarkistetaan, että viesti on pituudeltaan pidempi kuin 0 merkkiä ja jos viesti läpäisee tarkistuksen niin luodaan uusi dokumentti messages kokoelmaan Firestoren add metodin avulla. Luodulle dokumentille lisätään lähettäjän uid, viestin sisältö ja viestin lähetysaika.

Add metodi eroaa set metodista sillä, että add metodi luo uuden dokumentin metodin luomalla uid:llä, kun taas set metodia käyttäessä dokumentin nimi pitää olla jo tiedossa. Add metodilla ei myöskään pysty päivittämään dokumenttia niin kuin set metodilla, koska add metodi luo aina uuden dokumentin. Add metodi on hyvä viestien tallentamiseen, koska viesteillä olisi hyvä olla yksilöivä uid tietyn viestin tunnistamiseksi. Set metodilla olisi myös mahdollista luoda viesti dokumentti, mutta silloin pitäisi luoda uid koodissa ja asettaa tämä luotavan dokumentin nimeksi.

```
sendMessage() {
  const loggedInUser = firebaseApp.auth().currentUser.uid,
    trimmedMessage = this.message.trim();
  // Trimmed message so that user can't send empty messages.
  if (trimmedMessage.length > 0) {
    // Adds new message to messages collection
    firestore
      .collection("conversations")
      .doc(this.conversation_id)
      .collection("messages")
      .add({
        user_id: loggedInUser,
        message: trimmedMessage,
        timestamp: Date.now()
      })
      .catch(function() {
        console.log("Sending message failed.");
      });
  }
  // We need to clear messagebox when message is sent
  this.message = null;
}
```

Kuva 20. sendMessage funktio

4.7 Viestien näyttäminen

Viestien noutaminen tapahtuu aiemmin mainitulla onSnapshot metodilla, joka hakee automaattisesti uudet vastaanotetut viestit. Kun viesti noudetaan, niin tästä luodaan uusi objekti, joka lisätään messages-listalle. Messages sisältää kaikki keskustelussa lähetetyt viestit. Vue:n reaktiivisuuden ansiosta messages-listan datan muuttuessa luodaan uusi viestielementti, joka sisältää viestin sisällön ja lähetysajan. Elementin luonti tapahtuu samalla tavalla kuin ystävät-näkymällä ystäväelementin luonti. Luotu elementti lisätään automaattisesti keskustelunäkymälle.

```
// Load all messages from conversation and pushes them to messages array
firestore
  .collection("conversations")
  .doc(this.$route.params.id)
  .collection("messages")
  .orderBy("timestamp")
  .onSnapshot(function(querySnapshot) {
    querySnapshot.docChanges().forEach(function(change) {
      var data = change.doc.data();
      if (data) {
        self.messages.push({
          user_id: data.user_id,
          message: data.message,
          timestamp: moment(data.timestamp).format("D.M.YYYY H:mm ")
        });
      }
    });
  });
```

Kuva 21. Viestien nouto

Viestien lähettäjän tunnistamiseksi loin kaksi eri luokkaa, joiden CSS-tyylitykset eroavat toisistaan. Käyttäjän omien viestien pohjavärinä on sininen ja viestit sijaitsevat keskustelulaatikon vasemmassa laidassa, kun taas vastaanotettujen viestien pohjavärinä on harmaa ja ne sijaitsevat oikeassa laidassa. Luokan asettamiseksi käytin hyväksi komponentin templatella luokan asetus tarkistusta, jolla voidaan asettaa tietty luokka, jos ehto on tosi tai epätosi. Loin tarkistukseen ehdon, jossa tarkastetaan, onko viestille asetettu lähettäjän uid sama kuin kirjautuneella käyttäjällä. Jos lähettäjä on sisäänkirjautunut henkilö, niin asetetaan luokaksi sent ja jos ei ole niin asetetaan received. Alla olevassa kuvassa näkyy viestielementin luonti. Kuvassa näkyy myös luokan valinta ehto.

```
<div
  class="messages"
  v-chat-scroll="{ always: false, smooth: true }"
>
  <div v-for="message in messages" :key="message.id">
    <div
      class="message"
      :class="[
        message.user_id == user_id ? 'sent' : 'received'
      ]"
    >
      <p class="message-text">{{ message.message }}</p>
      <div class="message-time">{{ message.timestamp }}</div>
    </div>
  </div>
</div>
```

Kuva 22. Viesti-elementti

5 LOPUKSI

Ennen opinnäytetyöni aloittamista Vue:JS oli täysin tuntematon minulle muuten kuin nimeltään ja käyttötarkoitukseltaan. Vue:ta käyttäessäni huomasin kuinka kätevä ja työtä helpottava työkalu se on.

Pidin Vue:n komponenttipohjaisesta arkkitehtuurista, eli ohjelmassa näytetyistä elementeistä on mahdollista luoda omia komponentteja, joilla on omat ulkoasut, funktiot ja datat. Komponenttien helppo luominen ja niiden helppo liittäminen toisille komponenteille paransi koodikantaa sillä tavalla, että ohjelma tuli helpommin jaettua pienempiin osiin ja tämä auttoi koodin ylläpidossa ja selkeydessä. Tästä syystä tiedostoista ei tullut massiivisia sekasotkuja, jotka tekevät vähän kaikkea joka puolelle. Vue:sta on mielestäni suuri hyöty ja helpotus varsinkin suurten koodikantojen ylläpitämisessä.

Pidin myös Vue:n reaktiivisuudesta, eli komponentille päivittyneet tiedot päivittyivät suoraan myös niittä näyttäville näkymille. Tämä helpotti työtä, kun ei tarvinnut luoda omia funktioita tämän hoitamiseen.

Aikaisemmin mainitut asiat löytyvät jo monesta muustakin frameworkista, joten nämä asiat eivät erota Vue:ta kilpailijoistaan, mutta Vue:n helppo liitettävyys jo olemassa olevaan koodiin on Vue:n erottava tekijä muihin frameworkeihin verrattuna. Uskon, että tämä tekijä vaikuttaa Vue:n suosioon tulevaisuudessa, kun yritykset muuttavat frameworkejään moderneimpiin ja eivät halua kaikkea koodia päivittää tai kirjoittaa uusiksi. Vue:n hyötyihin näen myös sen hyvän ja selkeän dokumentaation.

Uskon, että Vue:n käytöstä työssäni oli suuri apu ja tulen myös käyttämään sitä tulevaisuudessa omissa projekteissa. Suosittelen Vue:ta varsinkin kaikille, jotka tarvitset modernia JavaScript frameworkia jo olemassa olevaan koodikantaan, jota ei ole mahdollista päivittää kokonaan.

Toiseksi työkalukseni valitsin Firebasen ja uskon, että se oli oikea valinta työhöni, koska se tarjosi kaikki tarvitsemani ominaisuudet ja se säästi paljon aikaani kun ei tarvinnut luoda omaa backendiä. Firebase oli mielestäni helppokäyttöinen ja helppo liittää omaan projektiin. Firebase tarjosi myös valmiit metodit sen ominaisuuksien käyttämiseen.

Pidin Firebasen tarjoamasta Firestore tietokanta ratkaisusta. Itselläni ei ollut aikaisempaa kokemusta NoSQL-tietokannoista, mutta se oli mielestäni selkeä ja helppokäyttöinen. Tietokantaa oli helppo päivittää ja hakea tietoa nopeasti Firestoren valmiilla metodeilla. Pidin Firestore dokumenttien helposta luonnista, kun ei tarvinnut valmiiksi suunnitella rakenneta, vaan sitä pystyi helposti muokkaaman projektin edetessä. Ainoa mikä tuntui NoSQL-tietokannoissa työläältä ja vaikealta oli muistaa päivittää data jokaiseen dokumenttiin, jotka sisälsivät saman datan. Relaatiotietokannoissa sama data löytyy todennäköisesti vain yhdestä taulusta ja riittää vain tämän päivittäminen.

Firebase on mielestäni hyvä varsinkin pienille kehitystiimeille, joilla ei ole välttämättä paljon rahaa ja aikaa kehittää omaa backendiä ja pystyttää omia palvelimia. Jos projektin backendissä ei tarvitse olla mitään kovin erikoista niin Firebase sopii hyvin

projektiin. Jos kuitenkin backend vaatisi jotain erikoisempaa niin tekisin kyllä oman backendin, koska voisi rakentaa juuri sellaisen, jonka tarvitsee. Firebaseen toki pystyy luomaan kustomoituja funktioita, mutta näillä ei voida tehdä aivan kaikkea. Voisin hyvinkin käyttää Firebasea uusiksi projekteissani, jos projekti olisi kohtuu yksinkertainen toiminnoiltaan ja tarvitsisin helpon tavan hakea dataa tietokannasta.

LÄHTEET

Mozilla. 2020. HTML basics. Viitattu 22.10.2020. https://developer.mozilla.org/en-US/docs/Learn/Getting_stated_with_the_web/HTML_basics

BitDegree. 2019. What is HTML: Introducing HTML Basics. Viitattu 22.10.2020. https://www.bitdegree.org/learn/what-is-html

Jämsen, P. n.d. HTML Attribuutit. Viitattu 23.10.2020. https://peda.net/p/jamspe/omat-atk-t/html5/5-html-attribuutit

Moraes, F. n.d. HTML For Beginners The Easy Way: Start Learning HTML & CSS Today. Viitattu 23.10.2020. https://html.com/

W3. n.d. CSS Introduction. Viitattu 23.10.2020. https://www.w3schools.com/css/css_intro.asp

Mozilla. 2020. What is CSS?. Viitattu 24.10.2020. https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/What_is_CSS

W3. n.d. How To Add CSS. Viitattu 24.10.2020. https://www.w3schools.com/css/css_howto.asp

Tutorials point. n.d. What is CSS?. Viitattu 24.10.2020. https://www.tutorialspoint.com/css/what_is_css.htm

Mozilla. 2020. About JavaScript. Viitattu 30.10.2020. https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript

Mozilla. 2019. Back to the Server: Server-Side JavaScript On The Rise. Viitattu 30.10.2020. https://developer.mozilla.org/en-US/docs/Archive/Web/Server-Side_JavaScript/Walkthrough

Mozilla. 2020. What is JavaScript?. Viitattu 30.10.2020. https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

Kantor, I. 2020. An Introduction to JavaScript. Viitattu 30.10.2020. https://javascript.info/intro

Tutorials point. n.d. JavaScript – Overview. Viitattu 30.10.2020. https://www.tutorialspoint.com/javascript/javascript_overview.htm

Morris, S. 2012. TECH 101: What Is JavaScript?. Viitattu 30.10.2020. https://skillcrush.com/blog/javascript/

Mamani, M. 2019. What is Vue.js and How do we Use It?. Viitattu 12.11.2020. https://www.avantica.com/blog/what-is-vue.js-and-how-do-we-use-it

Caputa, M. 2019. 8 Reasons Why Vue.js is Worth Considering for Your Next Project. Viitattu 12.11.2020. https://www.netguru.com/blog/why-vue-js

Vue.js. n.d. Vue documentation. Viitattu 12.11.2020. https://vuejs.org/v2/guide/

Vue.js. n.d. Comparison with Other Frameworks. Viitattu 17.11.2020. https://vuejs.org/v2/guide/comparison.html

Singh, V. 2020. What is Frameworks?. Viitattu 15.11.2020. https://hackr.io/blog/what-is-frameworks

Zviagin, I. 2020. What is Framework in Software Engineering?. Viitattu 15.11.2020. https://gbksoft.com/blog/what-is-framework/

Oracle. 2020. Database defined. Viitattu 22.11.2020. https://www.oracle.com/database/what-is-database.html

Javatpoint. n.d. What is Database?. Viitattu 22.11.2020. https://www.javatpoint.com/what-is-database

Schaefer, L. n.d. NoSQL vs SQL Databases. Viitattu 22.11.2020. https://www.mongodb.com/nosql-explained/nosql-vs-sql

Konkka, P. 2016. Tietokannat: NoSQL ja MongoDB. Viitattu 22.11.2020. https://petrikonkka.com/fi/pilvipalvelujen-tietokannat-nosql-mongodb/

Stevenson, D. 2018. What is Firebase? The complete story, abridged. Viitattu 22.11.2020. https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0

Esplin, C. 2016. What is Firebase? Viitattu 24.11.2020. https://howtofirebase.com/what-is-firebase-fcb8614ba442

Google. n.d. Firebase docs. Viitattu 24.11.2020. https://firebase.google.com/docs

Sutch, C.2020. What is Firebase? Viitattu 24.11.2020. https://dev.to/caelinsutch/what-is-firebase-1acj