jamk

Web-pohjainen asiakaspalvelun tukisovellus

Tieto- ja viestintätekniikka

Jusa Myrskog

Opinnäytetyö, AMK
Kesäkuu 2021
Tieto- ja viestintätekniikka
Insinööri (AMK), ohjelmistotekniikka

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Kuvailulehti

Myrskog Jusa

Web -pohjainen asiakaspalvelun tukisovellus

Jyväskylä: Jyväskylän ammattikorkeakoulu. **Kesäkuu 2021**, 43 sivua

Insinööri (AMK), tieto- ja viestintätekniikan tutkinto-ohjelma

Julkaisun kieli: suomi

Verkkojulkaisulupa myönnetty: kyllä

Tiivistelmä

Opinnäytetyön tavoitteena oli suunnitella ja toteuttaa web-pohjainen asiakaspalvelun tukisovellus. Sovellus toteutettiin JavaScript-kielellä käyttäen React-sovelluskehystä tukipuolen sovelluksen kehityksessä, jQueryä sekä vanilla JavaScriptiä asiakaspuolen moduulin kehityksessä, Node.js-ajoympäristöä backend-kehityksessä ja MySQL-tietokantaa keskustelujen tallentamista ja hakemista varten.

Sovelluksen vaatimuksina oli toteuttaa keskustelusovellus, jossa tukipuolella voi olla useita tukihenkilöitä. Keskusteluja pystyi luokittelemaan ja tallentamaan tietokantaan sekä tukipuolen käyttäjät pystyivät keskustelemaan samanaikaisesti yhden tai useamman asiakkaan kanssa. Itse keskusteluominaisuudet toteutettiin käyttämällä JavaScriptin Socket.IO-kirjastoa.

Sovelluksen kehitys tapahtui paikallisessa kehitysympäristössä ja uusia versioita siirrettiin lähes viikoittain asiakkaan testipalvelimelle, jolloin palautetta ja kehitysideoita saatiin jatkuvasti lisää.

Työn toimeksiantajana toimi Oululainen it-palvelu- ja konsultointialan yritys Indalgo Oy, ja itse sovellus toimitettiin Indalgon asiakkaalle. Asiakaspuolen moduuli otettiin käyttöön jo Indalgon aiemmin asiakkaalle toimittamassa sovelluksessa ja tukipuolen sovellus asennettiin omana prosessinaan asiakkaan palvelimelle. Aiemmin mitään tällaista kommunikointikanavaa ei Indalgon sovelluksissa ollut.

Opinnäytetyön tuloksena saatiin aikaiseksi moderni, helppokäyttöinen ja asiakkaan vaatimukset täyttävä sovellus, joka toivottavasti tuo myös lisäarvoa Indalgon kehittämille sovelluksille.

Avainsanat (asiasanat)

Web-kehitys, React, JavaScript, Node.js, MySQL, full stack, frontend, backend, Socket.IO

Muut tiedot (salassa pidettävät liitteet)

Esim. opinnäytetyön liitteen salassapitoperuste, ks. raportointiohjeen luku 4.1.2

jamk

Description

Myrskog Jusa

Web-based customer service support application

Jyväskylä: JAMK University of Applied Sciences, June 2021, 43 pages

Engineer (AMK), information and communications technology

Permission for web publication: Yes

Language of publication: Finnish

Abstract

The assigned goal of this thesis was to plan and implement a web-based customer service support application. The application was implemented with JavaScript, using the React framework in the development of the support-side application, jQuery and vanilla JavaScript in the development of the client-side module, Node.js runtime in the development of the backend application and MySQL for saving and querying conversations.

The requirements for the application were to implement a chat application in which the support-side could handle multiple logged-on support personnel, conversations could be classified and saved to a database and the support-side users could chat with multiple clients simultaneously. The chat functionalities themselves were implemented using a JavaScript library called Socket.IO.

The application development was done in a local development environment and new versions of the application were installed on the client's test server almost weekly, which supported the development process with constant feedback and development ideas.

The assignor for the application was an IT-service and consultation company from Oulu, called Indalgo Oy and the application itself was delivered to a client of Indalgo. The client-side module of the application was deployed in an application previously delivered to the client by Indalgo and the support-side application was deployed on the client's server as its own process. Previously no such communication channel was available in Indalgo's applications.

The outcome of the thesis was a modern, easy-to-use application that satisfied the client's requirements and hopefully gives more value to Indalgo's applications.

Keywords/tags (subjects)

Web-development, React, JavaScript, Node.js, MySQL, full stack, frontend, backend, Socket.IO

Miscellaneous (Confidential information)

For example, the confidentiality marking of the thesis appendix, see Project Reporting Instructions, section 4.1.2

Sisältö

Kuviot

1 Johdanto

1.1 Toimeksiantaja

Työn toimeksiantajana toimi Indalgo Oy. Indalgo Oy on Oulussa toimiva, vuonna 2010 perustettu IT-konsultointi- ja IT-palvelualan yritys. Indalgo Oy:n liikevaihto vuonna 2020 oli 1-2 milj. euroa, josta liikevoittoprosentti oli 31,6 %. Indalgolla oli 10 työntekijää opinnäytetyön kirjoittamisen hetkellä vuonna 2021.

1.2 Tavoitteet

Opinnäytetyön tavoitteena oli toteuttaa web-pohjainen asiakaspalvelun tukisovellus eräälle Indalgon asiakkaalle. Sovellus sisälsi asiakaspuolen moduulin, joka otettiin käyttöön jo Indalgon aiemmin asiakkaalle toimittamassa web-sovelluksessa, sekä tukipuolen moduulin, jolle luotiin oma verkkosivu. Sovelluksen tuli mahdollistaa kommunikointi asiakaspuolen sovelluksen ja tukipuolen sovelluksen käyttäjien välillä ongelmatilanteiden ratkaisun helpottamiseksi. Aiemmin mitään tällaista kommunikointikanavaa ei sovelluksessa ollut. Opinnäytetyön tuotoksen avulla tukipuoli pystyy keskustelemaan usean eri asiakkaan kanssa samanaikaisesti ja käytyjä keskusteluja pystyy luokittelemaan keskustelun aiheen mukaan ja tallentamaan tietokantaan.

Opinnäytetyön kirjallisen osuuden tavoite oli käydä läpi sovelluksessa käytettyjä teknologioita ja toteutusprosessia.

2 Vaatimukset

Sovelluksen tuli olla web-sovellus JavaScript-kielellä toteutettuna. Toteutukseen valittiin vanilla JavaScript ja jQuery asiakasmoduulin kehittämiseen ja React-sovelluskehys tukipuolen moduulin kehittämiseen. React valittiin koska Indalgolla on ollut aiempaa kiinnostusta Reactin käyttöönottamisesta projekteissa ja koska opinnäytetyön tekijä itse ehdotti Reactia, sekä on tehnyt web-kehitystä aiemmin Reactia käyttäen. Backend puolelle käyttöön otettiin node.js ja express.js palvelinohjelmistoksi.

Viestit tuli pystyä tallentamaan tietokantaan. Tietokantaohjelmiston valinta rajautui kahteen vaihtoehtoon – MySQL tai Mongo. Lopulta valinta rajautui MySQL:n, koska keskustelujen luokittelu ja luokiteltujen viestien hakeminen relaatiotietokannasta on tehokkaampaa kuin NoSQL kannasta.

Alla esitellään web-sovelluksen vaatimukset ja toiminnot.

Chat

Sovelluksen pääasiallinen toiminnallisuus on itse chat-toiminto. Chat-toiminnot on jaettu kahteen eri osaan: asiakaspuolen toimintoihin ja tukipuolen toimintoihin. Asiakaspuolen piti pystyä kommunikoimaan tukipuolen kanssa reaaliajassa ja tukipuolen piti pystyä luokittelemaan sekä arkistoimaan käytyjä keskusteluja. Tukipuolen piti saada Windows-ilmoitus kun asiakas aloitti keskustelun. Kaikkien keskustelujen asiakkaan kanssa tuli säilyä tukipuolen näkymässä 3kk:n ajan, ellei niitä poistettu tarkoituksella.

Kirjautuminen

Tukipuolen sovellukseen piti pystyä kirjautumaan sisään asiakkaan tunnuksilla. Sovellukseen otettiin käyttöön Indalgon jo aiemmin kehittämä Windows Active Directory -kirjautuminen.

Luokittelu ja arkistointi

Käytyjä keskusteluja tuli pystyä luokittelemaan etukäteen määriteltyjen kategorioiden mukaisesti. Jokainen chat-istunto eli käyty keskustelu asiakkaan yhteydenotosta asiakkaan poistumiseen saakka tuli pystyä luokittelemaan. Kaikki yli 3kk vanhat sessiot tuli tallentaa tietokantaan luokittelutietojen kera tai jos luokittelua ei oltu tehty kolmessa kuukaudessa, session viestit tuli poistaa.

3 Käytetyt teknologiat

3.1 JavaScript

JavaScript on kevyt, tulkittu tai ”Just in time” -käännetty ohjelmointikieli. JavaScript on parhaiten tunnettu verkkosivujen ohjelmointikielenä, mutta myös monet ei-selainympäristöt käyttävät sitä,

kuten esim. Node.js, Apache CouchDB ja Adobe Acrobat. JavaScript on yksisäikeinen, dynaaminen, olio-ohjelmointia tukeva ja dekaratiivinen ohjelmointikieli (JavaScript. 2021).

JavaScriptiä voidaan suorittaa nykyään missä vain ympäristössä, johon on asennettu JavaScript engine. Esimerkkejä JavaScript engineistä ovat V8 (Google Chrome) ja SpiderMonkey (Firefox) (An Introduction to JavaScript. N.d).

JavaScriptin yksisäikeisyys tarkoittaa sitä, että vain yksi komento voidaan suorittaa kerrallaan. Koodi suoritetaan siis järjestyksessä rivi riviltä eikä ohjelman suorittaminen (tietokoneen prosessori) voi hypätä kesken tehtävän suoritusta toiseen tehtävään käyttäen toista säiettä, kuten esim. Java ohjelmointikielessä. JavaScript -kielessä voidaan kuitenkin suorittaa koodia niin kuin se olisi monisäikeistä, käyttäen asynkronista ohjelmointia, hyväksikäyttäen call stackiä, call back queueta, web APIa ja event looppia. (Siddique S. 2020.)

JavaScript luotiin alun perin, jotta verkkosivut tuntuisivat olevan enemmän ”elossa”. Ohjelmia, jotka on kirjoitettu JavaScriptillä, kutsutaan skripteiksi. JavaScriptiä voidaan kirjoittaa suoraan verkkosivun HTML -tiedostoon ja se ajetaan automaattisesti, kun sivu latautuu. JavaScript verkkoselaimissa voi tehdä mitä vain verkkosivun sisällön manipulointiin liittyvää sekä olla vuorovaikutuksessa käyttäjän ja palvelimen kanssa, kuten reagoida käyttäjän klikkauksiin ja lähettää HTTP pyyntöjä. (An Introduction to JavaScript. N.d.)

JavaScript on ”turvallinen” ohjelmointikieli. Se ei tarjoa alhaisen tason pääsyä muistiin tai prosessoriin, koska JavaScript oli alun perin kehitetty selaimille, jotka eivät sellaista pääsyä tarvitse (An Introduction to JavaScript. N.d).

JavaScript-kieltä kehitetään ECMAScript-standardin mukaisesti. Yleensä standardin näkee lyhennettynä muodossa ES6, ES7 jne. Viimeisin ECMAScript-versio on ES11, joka julkaistiin vuonna 2020. ECMA International määrittelee ECMAScript-standardit. (JavaScript language resources. 2021.)

JavaScriptiä kirjoitetaan usein suoraan verkkosivujen HTML-tiedostoihin, script-tagien sisään, alla olevan kuvion mukaisesti (ks. kuvio 1). Nykyään JavaScript on kuitenkin modulaarisempaa, eli JavaScriptiä kirjoitetaan useimmiten omiin tiedostoihinsa.

```
<!DOCTYPE HTML>
<html>

<body>

  <p>Before the script...</p>

  <script>
    alert( 'Hello, world!' );
  </script>

  <p>...After the script.</p>

</body>

</html>
```

Kuvio 1. Yksinkertainen JavaScript-esimerkki HTML tiedostossa

Kun JavaScriptiä on kirjoitettu omaan tiedostoonsa, voidaan se liittää HTML-tiedostoon käyttäen script-tagin, src-attribuuttia (ks. kuvio 2).

```
<script src="/path/to/script.js"></script>
```

Kuvio 2. JavaScript-tiedoston liittäminen HTML tiedostoon

3.2 React

React on JavaScript-kirjasto käyttöliittymien kehitystä varten. React on Facebookin ja Instagramin kehityksessä alkunsa saanut projekti. React kehitettiin mahdollistamaan suurten sovellusten kehittäminen muuttuvan datan kanssa. Reactin kehitys alkoi vuonna 2011 ja sen lähdekoodista tuli avointa 2013. Reactin versio tämän opinnäytetyön kirjoittamisen hetkellä oli 17.0.2. (Krill P. 2014.)

3.2.1 Komponentit ja propertyt

Reactia kirjoitetaan jakamalla käyttöliittymän koodi itsenäisiin, uudelleenkäytettäviin osiin, joita kutsutaan komponenteiksi. Konseptuaalisesti komponentit ovat kuten JavaScript-funktioita, jotka ottavat vastaan syötteitä, eli funktion argumentteja. Näitä syötteitä kutsutaan nimellä ”props” (properties). Komponentit palauttavat React-elementtejä, jotka kuvaavat mitä näytölle pitää piirtää. (Components and Props. N.d.)

Komponentteja voidaan kirjoittaa ihan kuten JavaScript-funktioita ja yksinkertainen komponentti propseilla on esimerkiksi funktio, joka palauttaa käyttäjän nimen (ks. kuvio 3).

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

Kuvio 3. Yksinkertainen Hello world React -komponentti

Tätä komponenttia voitaisiin ”kutsua” eli se voitaisiin piirtää verkkosivulle yksinkertaisesti (ks. kuvio 4).

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

const element = <Welcome name="Sara" />;
ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Kuvio 4. React-komponentin renderöinti

Kuten mainittu aiemmin, komponentteja voidaan myös käyttää uudelleen, joten yllä luotua Welcome-komponettia voitaisiin käyttää useiden viestien renderöintiin. Alla olevassa esimerkissä on

luotu myös uusi App-komponentti, joka renderöi Welcome-komponentin kolme kertaa, tervehtien eri henkilöitä (ks. kuvio 5).

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

function App() {
  return (
    <div>
      <Welcome name="Sara" />
      <Welcome name="Cahal" />
      <Welcome name="Edite" />
    </div>
  );
}

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

Kuvio 5. React komponentin uudelleenkäyttäminen

3.2.2 State ja lifecycle

React-komponentteja voidaan kirjoittaa myös tilallisina komponentteina eli komponentille määritellään tilamuuttuja. Aina kun komponentin tila muuttuu, se renderöidään uudelleen ja näin käyttöliittymään piirretty sisältö vastaa ohjelman todellista tilaa.

Esimerkki yksinkertaisesta tilallisesta komponentista voisi olla esimerkiksi komponentti, joka näyttää montako kertaa käyttäjä on klikannut jotain elementtiä sivulla. Tällöin klikkausten määrää pidettäisiin komponentin tilamuuttujassa ja aina kun tila päivitetään, uusi senhetkinen klikkausten määrä piirretään sivulle.

Normaali funktiokomponentti voidaan muuttaa tilalliseksi komponentiksi kahdella eri tavalla – käyttämällä ES6-komponenttia tai uutta React 16.8 -versioon lisättyä ”hook”-ominaisuutta, jolla

voidaan muuttaa normaali funktionaalinen komponentti tilalliseksi komponentiksi (Using the State Hook. N.d).

React-tilamuuttujaa tulee käsitellä kuten se olisi muuttumaton (immutable), eli sitä ei tule muuttaa suoraan asettamalla arvoa muuttujaan, vaan kaikki muutokset tulee tehdä React API:n setState -metodin kautta, jolloin muutokset renderöidään aina oikein. Esimerkki tilallisesta ES6-komponentista, joka laskee ja piirtää näytölle napin painallusten määrän, esitetään alla (ks. kuvio 6). Aina kun käyttäjä painaa nappia, napin onClick-käsittelijässä kutsutaan Reactin setState -metodia, joka inkrementoi tilamuuttujaa ”count”.

```
class Example extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }

  render() {
    return (
      <div>
        <p>You clicked {this.state.count} times</p>
        <button onClick={() => this.setState({ count: this.state.count + 1 })}>
          Click me
        </button>
      </div>
    );
  }
}
```

Kuvio 6. Tilallinen ES6-komponentti

React-komponenteissa voi käyttää myös elinkaari (lifecycle) -metodeja, joita kutsutaan komponentin elinkaaren eri vaiheissa. Esimerkkejä tällaisista metodeista ovat componentDidMount, jota kutsutaan aina kun komponentti on renderöity sivulle ja componentWillUnmount, jota kutsutaan aina kun komponenttia ollaan poistamassa sivulta. (State and Lifecycle. N.d.)

3.3 Node.js

Node.js on asynkroninen, tapahtumalähtöinen (event-driven) JavaScript-ajoympäristö. Node.js on suunniteltu skaalautuvien web-sovellusten kehittämiseen (Introduction to Node.js. N.d).

Node.js suorittaa JavaScript-koodia selaimen ulkopuolella. Node.js käyttää Googlen V8 JavaScript engineä, joka tekee siitä erittäin suorituskykyisen (Introduction to Node.js. N.d).

Node.js sovellukset ajetaan yhdessä prosessissa, ilman uusien säikeiden luontia jokaiselle HTTP-pyynnölle. Kun Node.js suorittaa esimerkiksi I/O-operaation, kuten esim. tietokantakyselyn, Node.js ei odota vastausta tuhlaten näin prosessoriaikaa, vaan Node.js jatkaa kyseisiä operaatioita vasta, kun vastaus tietokannasta on valmistunut. Tämä mahdollistaa Node.js:n käsittelemään tuhansia samanaikaisia yhteyksiä ilman, että se vaatii sovelluskehittäjää painimaan säikeistämisen ja rinnakkaisuuden kanssa, joka voi olla suuri bugien lähde sovelluksissa. (Introduction to Node.js. N.d.)

Node.js:ää käytetään usein verkkosivujen palvelinohjelmistona ja koska sen ohjelmointikieli on JavaScript, tekee se backend-ohjelmoinnista helpompaa myös frontend-kehittäjille, jotka ovat tottuneet käyttämään JavaScriptiä. (Introduction to Node.js. N.d.)

Yksinkertainen esimerkki Node.js sovelluksesta on web-palvelin. Alla oleva esimerkki (ks. kuvio 7) käynnistää web-palvelimen, joka lähettää vastauksena tekstin ”Hello World!”, kun käyttäjä lähettää HTTP-pyynnön osoitteeseen 127.0.0.1, porttiin, joka on määritelty prosessin ympäristömuuttujiin.

```
const http = require('http')

const hostname = '127.0.0.1'
const port = process.env.PORT

const server = http.createServer((req, res) => {
  res.statusCode = 200
  res.setHeader('Content-Type', 'text/plain')
  res.end('Hello World!\n')
})

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`)
})
```

Kuvio 7. Node.js web-palvelinkoodi

Yllä luotu web-palvelinskripti voitaisiin ajaa komentoriviltä komennolla ”node server.js”, jossa skriptin nimi on ”server.js”.

Node.js-ekosysteemissä on mahdollista käyttää yli miljoonaa eri avoimen lähdekoodin koodikirjastoa, hyödyntäen npm (node package manager) -sovellusta. Yksi suosituimmista kirjastoista on esim. express.js, joka tarjoaa erittäin yksinkertaisen, mutta tehokkaan tavan luoda web-servereitä. (Introduction to Node.js. N.d.)

3.4 Socket.IO

Socket.IO on kirjasto, joka mahdollistaa reaaliaikaisen, kaksisuuntaisen ja tapahtumiin perustuvan kommunikoinnin selaimen ja palvelimen välillä. Socket.IO-kirjasto on käännetty useille eri ohjelmointikielille. (What Socket.IO is. N.d.)

Socket.IO-yhteys alustetaan oletuksena WebSocket-yhteytenä, jos mahdollista, ja HTTP ”long polling” -yhteytenä, jos WebSocket-yhteys epäonnistuu (What Socket.IO is. N.d).

WebSocket on kommunikaatioprotokolla, joka mahdollistaa full-duplex- sekä pienen latenssin yhteyden selaimen ja palvelimen välillä (What Socket.IO is. N.d).

Esimerkki (ks. kuvio 8) yksinkertaisesta palvelinohjelmistosta Socket.IO:lla luotuna, joka odottaa asiakkaan yhteydenottoja, vastaanottaa kaksi eri tapahtumaa ”message” ja ”salutations” ja kirjoittaa vastaanotetun viestin konsoliin.

```
io.on("connection", socket => {
    socket.on("message", data => {
        console.log(data);
    });

    socket.on("salutations", (elem1, elem2, elem3) => {
        console.log(elem1, elem2, elem3);
    });
});
```

Kuvio 8. Socket.IO-palvelinohjelmisto

Vastaavasti asiakaspuolella eli selaimessa, voidaan luoda lähes identtisellä koodilla vastaavat tapahtumat (ks. kuvio 9).

```
const socket = require("ws://localhost:3000");

socket.on("connect", () => {
    socket.on("message", data => {
        console.log(data);
    });

    socket.on("salutations", (elem1, elem2, elem3) => {
        console.log(elem1, elem2, elem3);
    });
});
```

Kuvio 9. Socket.IO-selainohjelmisto

Kun asiakas tai selain haluaa lähettää viestejä, yksinkertaisimmin se lähetetään johonkin olemassa olevaan nimettyyn tapahtumaan (ks. kuvio 10).

```
socket.emit("salutations", "Hello!");
```

Kuvio 10. Socket.IO-viestin lähetys "salutations" tapahtumaan

Socket.IO ei ole kuitenkaan WebSocket-implementaatio. Vaikka Socket.IO käyttää WebSocket-yhteyttä kun mahdollista, se lisää myös jokaiseen pakettiin metadataa, jonka takia WebSocket-clientti ei pysty yhdistämään Socket.IO-palvelimeen eikä Socket.IO-clientti pysty yhdistämään tavalliseen WebSocket-palvelimeen. (What Socket.IO is. N.d.)

3.5 Parcel

Parcel on web-sovellusten niputtaja (bundler), joka pystyy käyttämään useita ytimiä eikä vaadi ollenkaan konfigurointia (How It Works. N.d).

Parcel muuntaa puun assetteja puuksi bundleja. Parcel aloittaa työn ottamalla vastaan yhden entry-resurssin (asset), joka voi olla esim. JavaScript-, HTML-, CSS- tai kuvatiedosto, jäsentää (parse) tiedostot, etsii assettien riippuvuudet ja muuntaa ne niiden koottuun (compiled) muotoon – puuksi assetteja. (How It Works. N.d.)

Kun assettipuu on rakennettu, assetit siirretään bundlepuuhun. Bundle luodaan jokaiselle entry assetille (esim. HTML, JS tai CSS tiedosto) ja lapsibundle luodaan jokaiselle dynaamiselle importille. (How It Works. N.d.)

Bundlepuun rakentamisen jälkeen jokainen bundle kirjoitetaan tiedostoon käyttäen tiedostotyypin paketoijaa (packager). Paketoijat tietävät miten yhdistää koodia kustakin assetista yhteen koottuun tiedostoon, joka ladataan selaimessa. (How It Works. N.d.)

Yksi Parcelin kehitystyötä helpottavista ominaisuuksista on ”Hot Module Replacement” eli automaattinen moduulien päivitys selaimeen ilman, että sivua pitää ladata uudelleen. Kumpiakin, JavaScript- ja CSS-tiedostoja voi siten muuttaa ja muutosten vaikutuksen voi nähdä selaimessa lähes

välittömästi tiedoston tallentamisen jälkeen. Kun tiedosto tallennetaan, Parcel rakentaa tarpeelliset assetit uudelleen ja lähettää päivityksen käynnissä oleville asiakkaille. (Hot Module Replacement. N.d.)

Kun sovellus on valmis produktioon ja siitä luodaan produktioversio Parcelia käyttäen, Parcel ottaa pois käytöstä hot module reloading -ominaisuuden ja rakentaa sovelluksen. Produktioversiossa parcel minifioi kaikki output bundlet tiedostokoon pienentämiseksi ja ottaa pois käytöstä myös koodikirjastojen development mode -ominaisuudet. Esim. Reactissa on debugging-ominaisuuksia, joiden ottaminen pois käytöstä johtaa nopeampaan sovelluksen rakentamiseen ja pienempään lopulliseen tiedostokokoon. (Production. N.d.)

4 Sovelluksen toteutus

4.1 Yleistä

Opinnäytetyön aiheesta luotu sovellus kehitettiin pääasiassa aiemmin läpikäytyjä teknologioita käyttäen. Käytössä on myös muita, mutta ei niin suuressa osassa olevia koodikirjastoja. Alla käydään läpi sovelluksen backend- ja frontend-kehitystyötä.

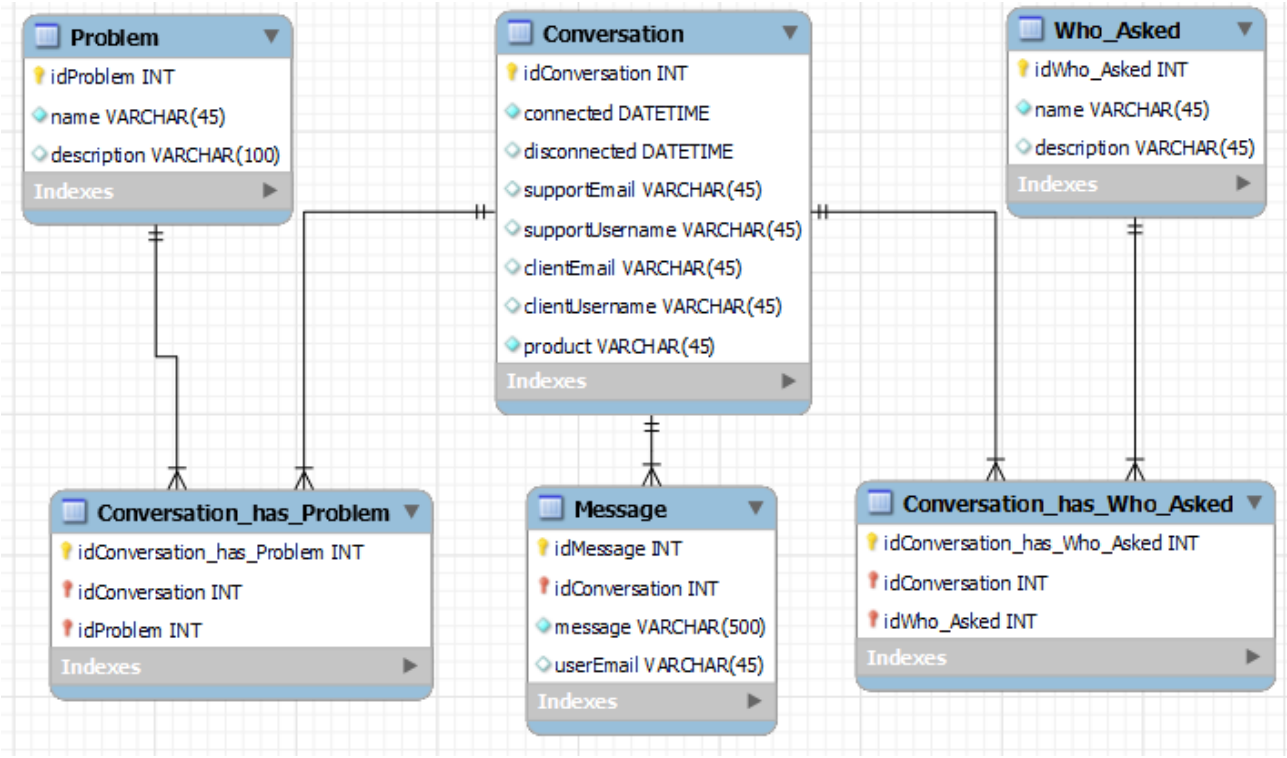
4.2 Backend

4.2.1 Toteutusympäristö

Backend-kehitykseen sisältyi Node.js:llä tehty palvelinohjelmisto, johon sisältyi web API, Socket.IO-moduuli, jonka välityksellä asiakas- ja tukipuolen frontend-sovellukset kommunikoivat sekä MySQL API viestien ja luokittelutietojen tallentamista ja hakemista varten. Myös MySQL-tietokanta luotiin sovellukselle.

4.2.2 Tietokanta

Tietokanta tehtiin MySQL-kantana. Tietokantaan tuli pystyä tallentamaan jokainen viesti, viestin lähettäjän tiedot (käyttäjänimi ja sähköposti), mihin keskusteluun kukin viesti kuului, keskustelun aloitus- sekä lopetusajat sekä mitä valittuja, vaihtoehtoisia luokittelutietoja kuhunkin keskusteluun kuului. Alla esitetään ER-kuvio luodusta tietokannasta (ks. kuvio 11).



Kuvio 11. MySQL-tietokannan ER-kuvio

Tämä tietokantarakenne mahdollistaa kaikki asiakkaan asettamat vaatimukset sovellukselle. Luokittelemattomat keskustelut voidaan poistaa helposti käyttäen jotain ajastettua tehtävää tai MySQL-eventtiä. Keskustelujen ja viestien hakeminen luokittelutietojen perusteella onnistuu myös helposti ja nopeasti, koska kaikki primary- ja foreign-keyt indeksoidaan MySQL-kannassa automaattisesti.

Who_Asked-taulu sisältää ennalta asetettuja, yrityksen sisäisiä osastoja ja Problem-taulu sisältää ennalta asetettuja ongelmakohteita, joita sovelluksen käytössä saattaa tulla vastaan, kuten esim. yleinen käyttöopastus.

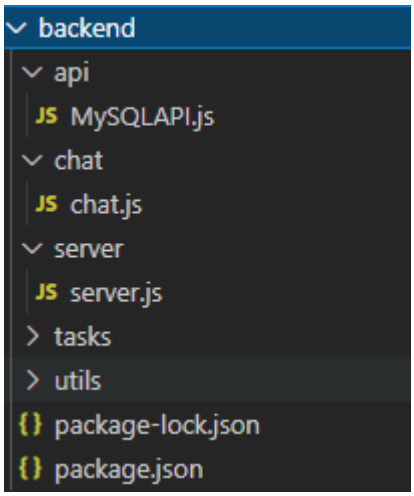
Tietokannalle luotiin myös käyttäjä, jolla on pääsy vain tähän tietokantaan (ks. kuvio 12).

```
CREATE USER 'chat_user'@'localhost' IDENTIFIED WITH mysql_native_password BY 'password';
GRANT ALL PRIVILEGES ON chat_db.* TO 'chat_user'@'localhost';
FLUSH PRIVILEGES;
```

Kuvio 12. MySQL käyttäjän luonti

4.2.3 Node.js-palvelinohjelmisto

Node.js-palvelinohjelmisto eli sovelluksen backend-ohjelmisto koostui kolmesta eri osasta – MySQL API:sta, Socket.IO/Chat API:sta sekä Express.js-palvelinkoodista. Alla esitetään backend-ohjelmiston hakemistorakenne (ks. kuvio 13).



Kuvio 13. Backend-hakemistorakenne

Backendin kehityksessä käytettiin kuutta eri npm-kirjastoa: connect-mongo, connect-roles, express, express-session, mysql ja socket.io. Näistä connect-mongo-, connect-roles- ja express-session-kirjastoja käytettiin Indalgon kirjautumistoimintojen käyttöönottamiseksi sovelluksessa eikä niitä käydä sen tarkemmin läpi tässä opinnäytetyössä. Mainittakoon kuitenkin, että connect-roles-kirjastoa käytettiin tämän sovelluksen web API:ssa tarkistamaan käyttäjän kirjautuminen, kun käyttäjä hakee tukipuolen sovellukseen keskustelut ja luokittelutiedot tietokannasta onnistuneen sisäänkirjautumisen jälkeen.

Alla (ks. kuvio 14) esimerkiksi keskustelutietojen hakeminen tietokannasta HTTP-pyynnöllä express-reitistä /api/conversations, jossa tarkistetaan connect-roles-kirjastoa käyttäen, että käyttäjällä on oikeus käyttää sovellusta. Tämä tarkistus tehdään käyttäjän evästeet (cookie) läpikäymällä, käyttämällä connect-roles-kirjaston user.can-metodia express-reitin middlewarena.

```
app.get("/api/conversations", user.can('access application'), async (req, res) => {
    try {
        const result = await MySQLAPI.getAllConversations();
        res.send(JSON.stringify(result));
    }
    catch(e) {
        console.log("/api/conversations ERROR!");
        console.log(e);
        res.status(500).send("Something went wrong.");
    }
});
```

Kuvio 14. /api/conversations-reitti keskustelujen hakemiseen server.js-tiedostossa

Seuraavassa kuviossa (ks. kuvio 15) Socket.IO-kirjasto initialisoidaan liittämällä se luotuun express-palvelimeen samassa palvelinkoodit sisältävässä tiedostossa kuin missä API-reitit on määritelty. Sitten initialisoitu Socket.IO-kirjasto annetaan argumenttina itse kehitetylle Chat-moduulille, jotta moduulissa voidaan luoda tarvittavat tapahtumat, joita halutaan kuunnella.

```
let server = https.createServer(cert_settings, app);
let io = socket(server, {
    cors: {
        origin: clientAppOrigins,
        methods: ["GET", "POST"]
    }
});
Chat.init(io);

server.listen(server_settings.port, () => {
    console.log("[redacted]chat-support backend listening at https://localhost:%s", server_settings.port);
});
```

Kuvio 15. Express-palvelimen ja Socket.IO-kirjaston initialisointi server.js-tiedostossa

Socket.IO-alustuksen yhteydessä määritellään myös CORS (cross origin resource sharing) -parametri, joka tarvitaan, jotta eri Node.js-prosessissa käynnistetty asiakassovellus pystyy ottamaan Socket.IO-yhteyden tähän palvelimeen. clientAppOrigins-muuttujan arvona on tässä asiakassovelluksen IP-osoite ja portti. Sallitut HTTP-pyyntö -metodit osoitteesta ovat GET ja POST.

Lopuksi vielä asetetaan express-palvelin kuuntelevaan tilaan server.listen-komennolla.

Chat-moduulissa luodaan Socket.IO-tapahtumat ja määritellään tarvittavat toiminnot, kun kyseiset tapahtumat vastaanotetaan. Periaatteessa jokainen tapahtumametodi on hyvin samanlainen - ne kuuntelevat niille määriteltyjä tapahtumia, tekevät jotain vastaanotetulla datalla, esim. tallentavat vastaanotetut viestit tietokantaan, ja lähettävät viestin toiselle osapuolelle (tai osapuolille). Chat-moduuliin luodut tapahtumat ovat:

1. Connection – Vastaanottaa uusia asiakasyhteyksiä tuki- sekä asiakaspuolelta.
2. Create – Luo chat-huoneen. Jos yhdistänyt asiakas on tukipuolen käyttäjä ja tukipuolen sovelluksessa ei ole muita sisäänkirjautuneita käyttäjiä, luodaan uusi support-hub niminen chat-huone. Jos huone on jo luotu, uudet tukipuolen käyttäjät lisätään jo luotuun huoneeseen. Jos yhdistänyt asiakas ei ole tukipuolen käyttäjä, luodaan uusi chat-huone asiakkaan käyttäjänimen perusteella, lisätään tietokannan conversation-tauluun uusi rivi ja kaikki olemassa olevat tukipuolen käyttäjät lisätään tähän huoneeseen.
3. Message – Vastaanottaa viestejä tuki- sekä asiakaskäyttäjiltä, tallentaa viestit tietokantaan ja välittää ne kaikille huoneen jäsenille.
4. Typing – Vastaanottaa viestin, kun tuki- tai asiakaskäyttäjä aloittaa tai lopettaa chattiin kirjoittamisen. Tämä viesti välitetään huoneen jäsenille, jotta voidaan näyttää frontend-sovelluksissa viesti, että käyttäjä kirjoittaa parhaillaan jotain.
5. Classify – Vastaanottaa tukipuolen sovelluksesta viestin ja luokittelutiedot, kun tukipuolen käyttäjä luokittelee käydyn keskustelun. Luokittelutiedot tallennetaan tietokantaan ja kaikille tukipuolen käyttäjille välitetään viesti, että keskustelu on luokiteltu, jolloin kaikkien tukipuolen käyttäjien käyttöliittymän näkymä voidaan päivittää näyttämään uusimmat luokittelutiedot.
6. Delete – Vastaanottaa tukipuolen sovelluksesta viestin jonkin luokittelemattoman keskustelun poistamiseksi, poistaa keskustelun ja sen viestit tietokannasta ja välittää kaikille tukipuolen sovelluksen käyttäjille viestin, että keskustelu on poistettu, jolloin käyttöliittymän näkymä voidaan päivittää näyttämään tietokannan tietoja vastaava tila.
7. Client-disconnect – Vastaanottaa viestin asiakaspuolen sovelluksesta, kun asiakas lopettaa chat-keskustelun. Tietokannan conversation-taulun disconnected-kenttään tallennetaan kellonaika, kun chat-keskustelu lopetettiin.
8. Disconnect – Vastaanottaa viestin, kun tukipuolen käyttäjä kirjautuu ulos tai päivittää sivuston, tai kun asiakaskäyttäjä lopettaa chat-keskustelun, kirjautuu ulos tai päivittää sivun. Tämä tieto välitetään asiakkaalle tai tukipuolen käyttäjälle, riippuen siitä kuka poistui, ja frontend-sovelluksessa näytetään viesti, että toinen osapuoli on poistunut sovelluksesta. Jos poistunut käyttäjä oli tukipuolen käyttäjä ja tukipuolella on useampia kirjautuneita käyttäjiä, asiakas ei saa tällöin mitään viestiä, koska viestittely voi edelleen jatkua. Jos poistunut käyttäjä oli asiakaskäyttäjä, tukipuolen frontendiin välitetään viesti asiakkaan poistumisesta kyseiseen chat-huoneeseen, jolloin tukipuolen sovelluksen näkymä voidaan päivittää näyttämään keskustelun luokitteluvalinnat.

Alla (ks. kuvio 16) näytetään esimerkki classify-tapahtumasta. Jos keskustelu luokitellaan onnistuneesti ja support-hub-huoneessa on enemmän kuin 1 käyttäjä, käydään läpi lista käyttäjistä for-

loopissa ja lähetetään classification-done-viesti jokaiselle, paitsi luokittelun tehneelle tukipuolen käyttäjälle.

```
socket.on('classify', (data) => {
    MySQLAPI.classifyConversation(data, (err, res) => {
        if(err !== null) {
            console.log("classifyConversation ERROR!");
            console.log(err);
        }
        else {
            if(VERBOSE) console.log("classifyConversation: " + res);
            let supportSockIds = roomsAndUsers.get("support-hub");
            if (typeof supportSockIds == "object") {
                for(let i = 0; i < supportSockIds.length; i++) {
                    if(supportSockIds[i] != socket.id) {
                        io.to(supportSockIds[i]).emit('classification-done', res);
                    }
                }
            }
        }
    })
});
```

Kuvio 16. Chat-moduulin classify-tapahtuma chat.js-tiedostossa

MySQL API on tehty luokkana nimeltä MySQLAPI ja se sisältää staattisia metodeja MySQL-tietokannan kanssa kommunikoimiseksi. Kahta näistä metodeista kutsutaan web API:sta, HTTP GET -pyyntöjen seurauksena, ja kaikkia muita Chat-moduulista Socket.IO-tapahtumien seurauksena. Esimerkiksi aiemmin näytetty /api/conversations-reitti kutsuu MySQL API:n metodia getAllConversations, joka palauttaa kaikki alle 3kk vanhat keskustelut tietokannasta (ks. kuvio 17).

MySQLAPI käyttää npm-kirjasto mysql:ää.

```
/**
 * getAllConversations
 *
 * Get all conversations and their messages for the past 3 months from the database, or
 * all conversations if old == true.
 *
 * @param {boolean} old Also get conversations older than 3 months
 *
 * @reject Error
 * @resolve Array<Object>
 * @returns Promise
 */
static getAllConversations(old) {
    return new Promise((resolve, reject) => {
        db.getConnection(async (err, chatDb) => {
            if(err) {
                reject(new Error(err));
            }
            else {
                let sql = `
                    SELECT c.idConversation, c.connected, c.disconnected, c.product, m.idMessage, m.message, m.username,
                    w.whoAsked, p.name as problem
                    FROM [redacted]_chat_support.conversation as c
                    INNER JOIN [redacted]_chat_support.message as m USING (idConversation)
                    LEFT JOIN [redacted]_chat_support.conversation_has_who_asked USING (idConversation)
                    LEFT JOIN [redacted]_chat_support.who_asked as w USING (idWho_Asked)
                    LEFT JOIN [redacted]_chat_support.conversation_has_problem USING (idConversation)
                    LEFT JOIN [redacted]_chat_support.problem as p USING (idProblem)
                `;
                if(old === undefined || old === false) {
                    sql += "WHERE c.disconnected >= (NOW() - INTERVAL 3 MONTH) OR c.disconnected IS NULL";
                }

                try {
                    let res = await this.query(chatDb, sql);
                    chatDb.release();
                    resolve(res);
                }
                catch(e) {
                    chatDb.release();
                    reject(new Error(e));
                }
            }
        });
    });
}
```

Kuvio 17. MySQL API getAllConversations -metodi MySQLAPI.js-tiedostossa

Yllä oleva metodi kutsuu myös itse tehtyä query-metodia, joka on vain mysql-kirjaston query-metodin wrapperifunktio, joka palauttaa query-metodin promisessa. Wrapperifunktio käyttää myös mysql-kirjaston format-metodia parametrien turvalliseen formatointiin SQL-injektioiden estämiseksi (ks. kuvio 18).

```
/**
 * Helper function that returns the query function in a promise
 *
 * @param {object} db database instance as returned by getConnection
 * @param {string} sql
 * @param {array} params
 */
static query(db, sql, params) {
    let q = "";
    if(params != undefined){
        q = mysql.format(sql, params);
    }
    else {
        q = sql
    };
    return new Promise((resolve, reject) => {
        db.query(q, (err, res) => {
            if(err) {
                reject(new Error(err));
            }
            else {
                resolve(res);
            }
        })
    });
}
```

Kuvio 18. MySQL API query -metodi MySQLAPI.js-tiedostossa

4.3 Frontend

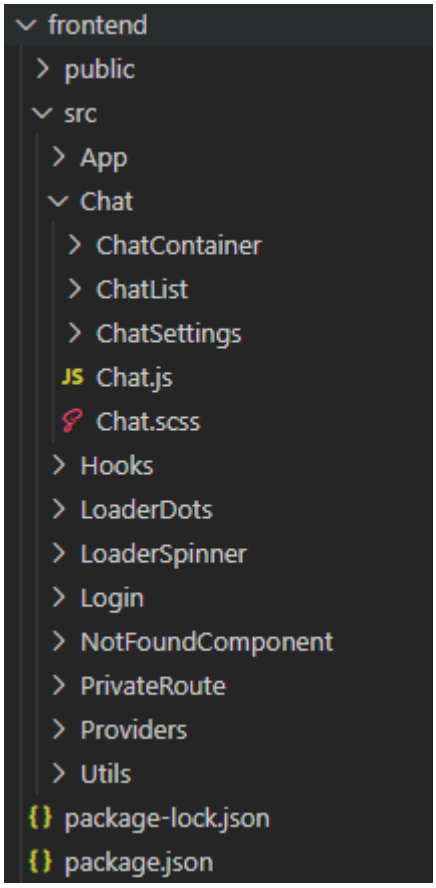
4.3.1 Yleistä

Sovelluksen frontend-puoli rakentui asiakaspuolen moduulista, joka voidaan ottaa käyttöön millä vain JavaScriptillä kehitetyllä verkkosivulla - kunhan verkkosivun palvelimen IP ja portti lisätään backendin CORS-asetuksiin - sekä tukipuolen sovelluksesta, joka on kehitetty Reactilla ja jonka bundlaamisessa käytetään Parcel-ohjelmistoa sovelluskehityksen nopeuttamiseksi ja lopullisen produktioversion minifioinnissa, asetusten säätämisessä ja rakentamisessa. Frontend-sovelluksessa käytetään myös Socket.IO-kirjaston selainversiota itse chat-toimintojen toteuttamiseksi.

4.3.2 Tukipuoli

Tukipuoli on kehitetty React-sovelluskehyksen versiolla 17.0.1 ja se sisältää kirjautumistoiminnot, chat-toiminnot sekä keskustelujen luokittelutoiminnot. Käytetyt npm-kirjastot ovat fontawesome, axios, react, react-dom, react-router-dom ja socket.io-client, sekä development-riippuvuudet parcel-bundler ja sass.

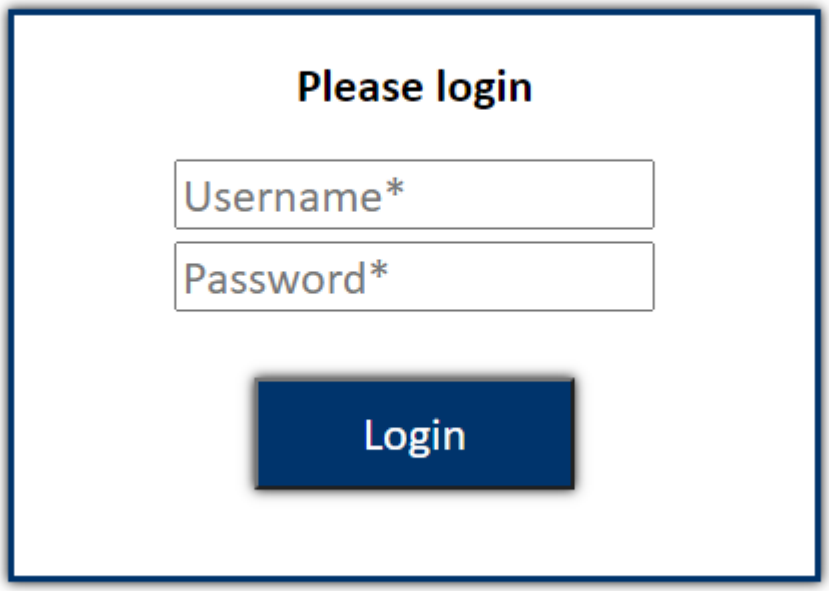
Projekti on jäsennetty niin, että jokainen React-komponentti on omassa, samannimisessä kansiossaan, joka sisältää myös komponentin tyylimäärittelyt omassa scss (Syntatically Awesome Style Sheets) -tiedostossaan (ks. kuvio 19).



Kuvio 19. Frontend-hakemistorakenne

Kun käyttäjä tulee sivustolle, ensimmäiseksi käyttäjää pyydetään kirjautumaan sisään sovellukseen (ks. kuvio 20). Kirjautumistunnuksina toimii asiakasyrityksen Active Directory -tunnukset ja kirjautumisen yhteydessä sovellukseen palautetaan mm. käyttäjän oikea nimi sekä sähköpostiosoite.

Käyttäjän nimi lähetetään viestien mukana asiakkaille ja sähköpostiosoitetta sekä nimeä käytetään viestien ja keskustelujen tallentamisessa tietokantaan, jotta jokaiseen viestiin ja keskusteluun voidaan liittää käyttäjätiedot.



Kuvio 20. Tukipuolen kirjautumisnäkymä

Kirjautumisen yhteydessä backendin palauttamat arvot - käyttäjän nimi sekä sähköpostiosoite - tallennetaan selaimen localStorageen, josta ne voidaan lukea tarvittaessa viestejä tai luokittelutietoja tallentaessa (ks. kuvio 21).

```
signIn(user, pass, cb) {
    axios({
        method: "post",
        url: "/login",
        data: {username: user, password: pass}
    }).then((res) => {
            iasAuth.isAuthenticated = true;
            console.log(res);
            localStorage.setItem("[redacted]ChatUser", res.data.name);
            localStorage.setItem("[redacted]ChatEmail", res.data.email);
            cb();
        })
        .catch((err) => {
            cb(err);
        });
},
```

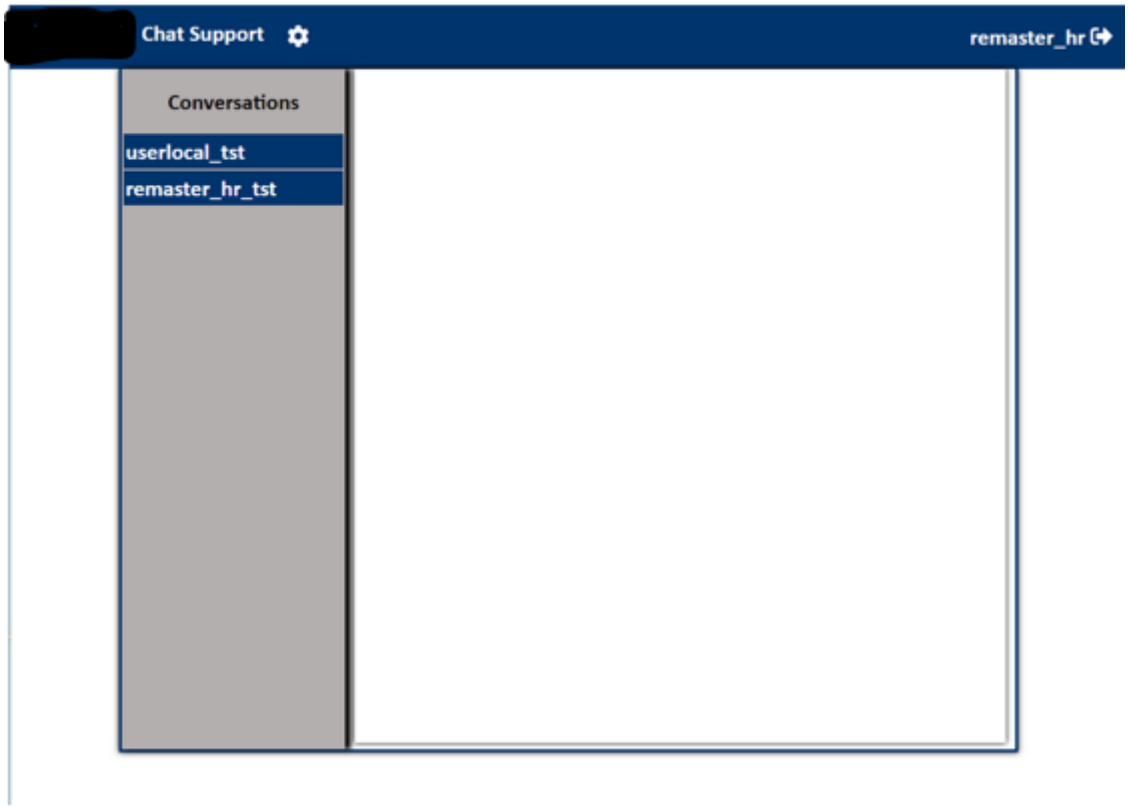
Kuvio 21. Kirjautuminen axios-kirjastoa käyttäen ja localStorageen tallennus useProvideAuth.js-tiedostossa

Kirjautumisen jälkeen käyttäjälle avautuu chat-näkymä, jossa vasemmassa reunassa näkyy kaikki huoneet (käyttäjien nimet), joissa on alle 3kk vanhoja viestejä. Sivuston oikeassa yläreunassa näkyy kirjautuneen käyttäjän nimi, sekä uloskirjautumispainike ja sivuston vasemmassa yläreunassa sovelluksen nimen jälkeen näkyy hammasratas-painike, josta pääsee sovelluksen asetuksiin.

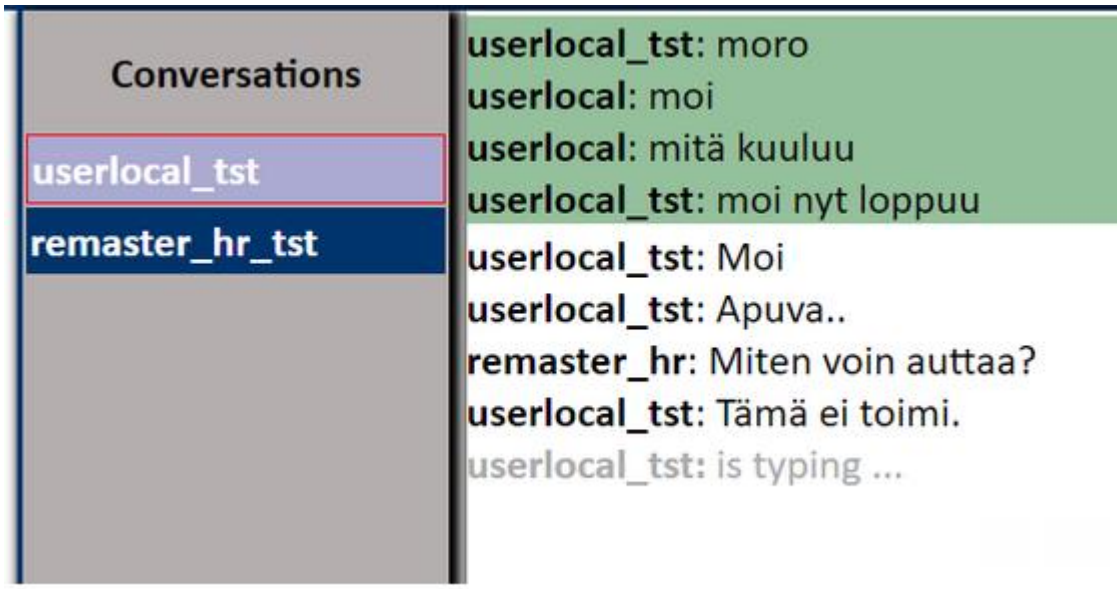
Kun jotain chat-huoneen nimeä painaa, avautuu kyseinen chat-huone sivun keskellä olevalle valkoiselle tyhjälle alueelle ja käyttäjä voi selailla käytyjä keskusteluja, sekä poistaa tai luokitella luokittelemattomia keskusteluja.

Jos jossain huoneessa on lukemattomia viestejä, huoneen nimen perässä on lukemattomien viestien määrä numerona ja huone, jossa on uusin vastaanotettu viesti, nousee listan ylimmäiseksi.

Alla (ks. kuvio 22, 23) olevissa esimerkeissä huoneiden nimet ovat testinimiä, eivätkä ne vastaa sovelluksen todellista tilaa, kun sovellukseen on kirjauduttu oikeilla tunnuksilla, joissa käyttäjätunnukset on liitetty oikeaan henkilöön. Kuten mainittu aiemmin, nimet olisivat oikeassa käytössä käyttäjien oikeita nimiä.



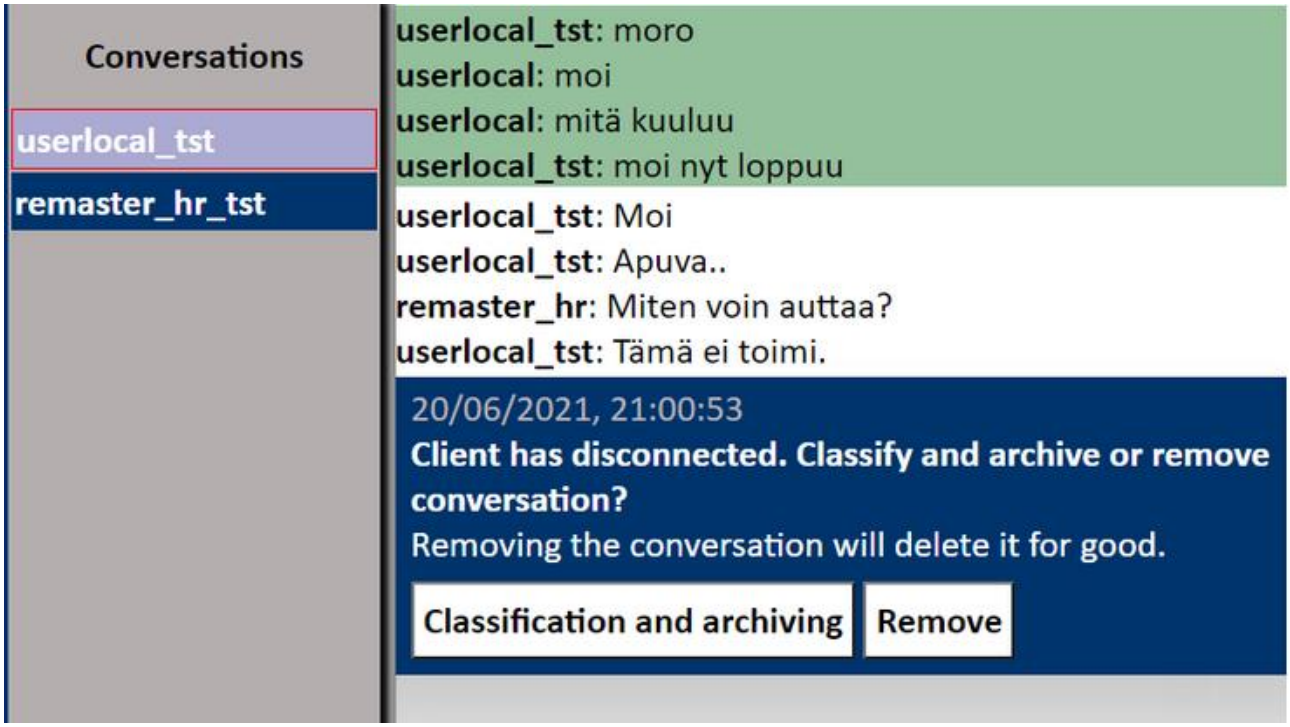
Kuvio 22. Tukipuolen chat-näkymä ilman avattua keskustelua



Kuvio 23. Tukipuolessa avattu huone yhdellä luokitellulla keskustelulla (virheä tausta) ja yhdellä keskeneräisellä keskustelulla. _tst-päätteiset nimet ovat asiakaspuolen käyttäjiä.

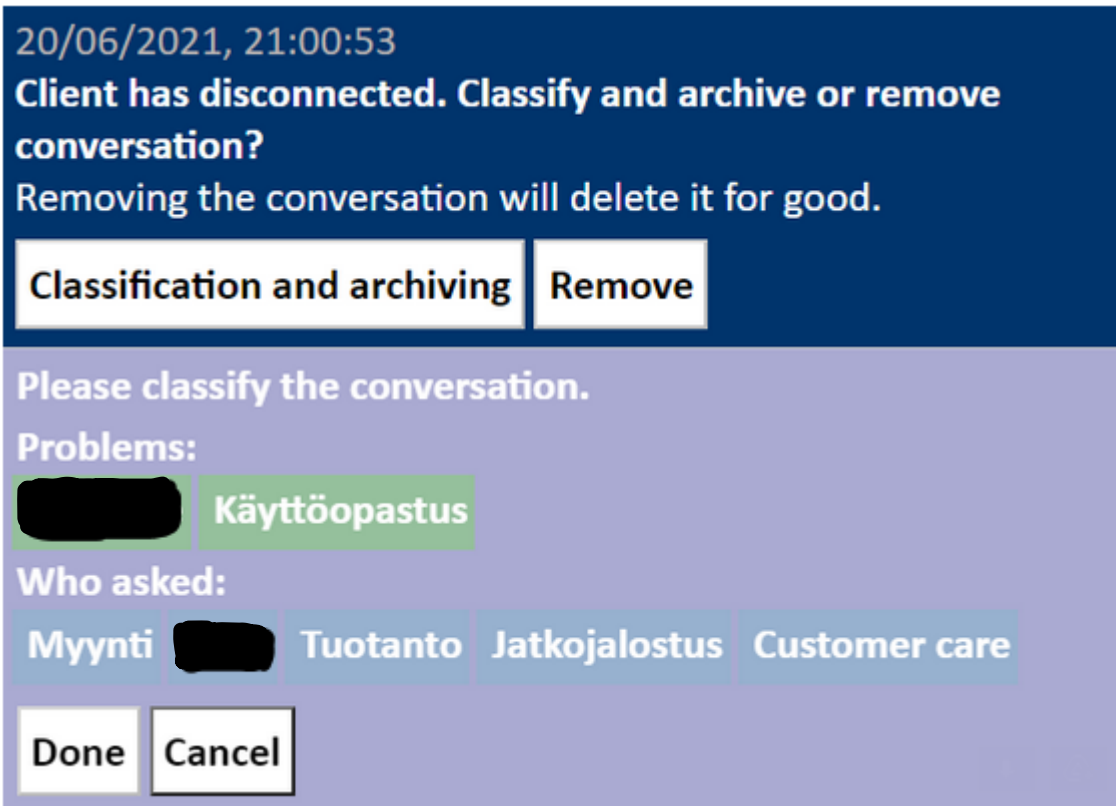
Kuten yllä olevasta kuvasta näkee, yhdessä huoneessa voi olla keskustelemassa myös useita eri tukihenkilöitä. Tässä tapauksessa huoneeseen on vastannut kaksi eri tukihenkilöä; userlocal ja remaster_hr.

Kun asiakaspuolen käyttäjä on poistunut chatistä, voi tukikäyttäjä luokitella käydyn keskustelun tai halutessaan jättää keskustelun luokittelematta ja luokitella keskustelun myöhempänä ajankohtana (ks. kuvio 24). Yhdessä huoneessa voi olla rajattomasti luokittelemattomia keskusteluja viimeisen kolmen kuukauden ajalta.



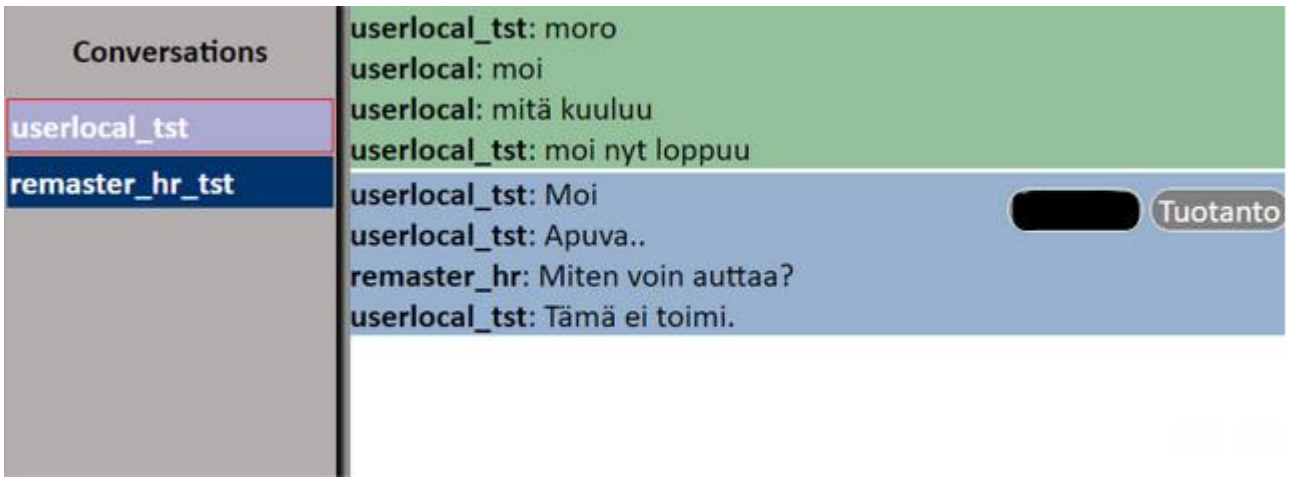
Kuvio 24. Luokittelutoiminto asiakkaan poistumisen jälkeen

Kun tukihenkilö painaa Classification and archiving -painiketta, aukeaa chat-näkymään uusi, luokitteluvaihtoehdot sisältävä näkymä, josta voi valita keskusteluun sopivat luokitteluehdot (ks. kuvio 25).



Kuvio 25. Luokittelutoiminto, avattu valikko luokitteluvaihtoehdoilla

Kun keskustelu on luokiteltu, sen taustaväri muuttuu chat-näkymässä ja kun hiiren vie keskustelun päälle, näkyy valitut luokittelutiedot keskustelun oikeassa yläreunassa (ks. kuvio 26).



Kuvio 26. Luokitellun keskustelun hover-tapahtuma

Keskustelun taustaväri määräytyy valitun Problems-vaihtoehdon arvon mukaan.

Kaikki huoneet ja niissä käydyt keskustelut tallennetaan Reactin rooms-tilamuuttujaan, josta ne voidaan renderöidä sivulle elementeiksi (ks. kuvio 27).

```
this.state = {
    socket: null,
    rooms: [], /* [{
        name: <string>,
        messages: [],
        unread: <int>,
        connected: <Timestamp>,
        disconnected: <null|Timestamp>,
        typing: [<string>], array of typing users
        convId: <int>, newest conversationId in room
        lastMsgAt: <Timestamp>, time latest message was received
        labels: [{
            convId: <int>
            messages: [startIndex, endIndex],
            problems: [],
            whoAsked: [],
            connected: <Timestamp>,
            disconnected: <null|Timestamp>
        }]
    }]*/
    activeRoom: "", // ex: "userlocal",
    loadingConversations: false,
    showSettings: false,
    // settings..
    sounds: false, // message sounds (play sound when message is received)
    whenNotOnTop: false // only play sound if chat support is not active tab/topmost window
}
```

Kuvio 27. Chat-komponentin tilaobjekti Chat.js-tiedostossa

Chat-komponentin tilamuuttujassa on myös tallennettuna itse yhdistetty socketti, activeRoom-arvo, joka pitää sisällään aktiivisen huoneen sekä asetusarvot äänihälytyksille. Tilamuuttujan rooms-objektin labels-taulukko pitää sisällään luokittelutiedot kullekin keskustelulle. Se ei sisällä itse viestejä, vaan pelkästään viestien alku- ja loppuindeksit, joita käyttämällä voidaan jakaa keskusteluhistoria osiin ja renderöidä esim. keskustelujen taustavärit oikein. Itse viestit tallennetaan rooms-objektin messages-taulukkoon.

Kun tukipuolen sovellus ladataan ensimmäistä kertaa kirjautumisen jälkeen, haetaan keskustelut backendistä axios-kirjastoa käyttäen HTTP-pyynnöllä reitistä /api/conversations. Kun vastaus on

saapunut, alustetaan vastauksen objekteista rooms-tilamuuttuja. Alla käydään läpi tämä prosessi osissa useina kuvina (ks. kuvio 28, 29, 30, 31).

```
.then((res) => {
    let data = res.data;
    // Build room objects from data. Username in first object for each
    // conversation will be the room name (because only the client can start conversations)
    // The requested objects should be in the correct order here.
    //
    // First, group by idConversation
    let grouped = groupBy(data, (d) => d.idConversation);
    // Add the roomname to every object to make this easier.
    let withRooms = data.map(d => {
        let g = grouped[d.idConversation];
        return {...d, room: g[0].username}
    });
```

Kuvio 28. Tilaobjekti rooms-alustus Chat.js-tiedostossa, osa 1

```
// group by props:
let groups = ['room', 'idConversation'];
let done = {};
withRooms.forEach(function (a) {
    groups.reduce(function (o, g, i) {
        o[a[g]] = o[a[g]] || (i + 1 === groups.length ? [] : {});
        return o[a[g]];
    }, done).push(a);
});
```

Kuvio 29. Tilaobjekti rooms-alustus Chat.js-tiedostossa, osa 2

```
// get messages per room
let keys = Object.keys(done);
let messagesPerRoom = {}
keys.map(k => {
    messagesPerRoom[k] = Object.values(done[k]).map(conv => {
        return conv.map(m => {return m.username+": "+m.message});
    });
});
```

Kuvio 30. Tilaobjekti rooms-alustus Chat.js-tiedostossa, osa 3

```
// and finally build the rooms state object
let newState = keys.map(k => {
    let vals = Object.values(done[k]);
    let lastEndIndex = 0;
    return {
        name: k,
        messages: [...messagesPerRoom[k]].flat(1),
        unread: 0,
        connected: vals[vals.length-1][vals[vals.length-1].length-1].connected,
        disconnected: vals[vals.length-1][vals[vals.length-1].length-1].disconnected,
        typing: [],
        convId: vals[vals.length-1][0].idConversation,
        lastMsgAt: null,
        labels: vals.map((conv,i) => {
            lastEndIndex = lastEndIndex + (vals[i-1] ? vals[i-1].length : 0);
            return {
                convId: conv[0].idConversation,
                messages: lastEndIndex == 0 ? [lastEndIndex, conv.length-1] : [lastEndIndex, lastEndIndex+conv.length-1],
                problems: [conv[0].problem],
                whoAsked: [conv[0].whoAsked],
                connected: conv[0].connected,
                disconnected: conv[0].disconnected
            }
        })
    }
});
console.log(newState);
this.setState({
    rooms: newState,
    loadingConversations: false
});
```

Kuvio 31. Tilaobjekti rooms-alustus Chat.js-tiedostossa, osa 4

Tukipuolen käyttäjät saavat myös Windows-notifikaatioita aina, kun uusi asiakas yhdistää sovellukseen ja jos tukipuolen sovellus ei ole aktiivinen ikkuna (ks. kuvio 32).

```
const getPermission = () => {
    if(!("Notification" in window)) alert("Your browser does not support desktop notifications!");

    Notification.requestPermission((permission) => {
        if(!('permission' in Notification)) Notification.permission = permission;
    });
}

const createNotification = () => {
    if(Notification.permission === "granted") {
        let notification = new Notification(
            '[redacted]-chat-support", // title
            {   // options
                body: "New client just connected!"
            }
        );
    }
}

export { getPermission, createNotification };
```

Kuvio 32. Windows-notifikaation lupapyyntö- ja luontifunktiot utilFunctions.js-tiedostossa

Yllä esitettyä getPermission-funktiota kutsutaan Chat-komponentin componentDidMount elinkaari -metodissa, kun sivu ladataan ensimmäistä kertaa (ks. kuvio 33).

```
componentDidMount = () => {
    // ask permission to show notifications
    let perm = notification.getPermission();
```

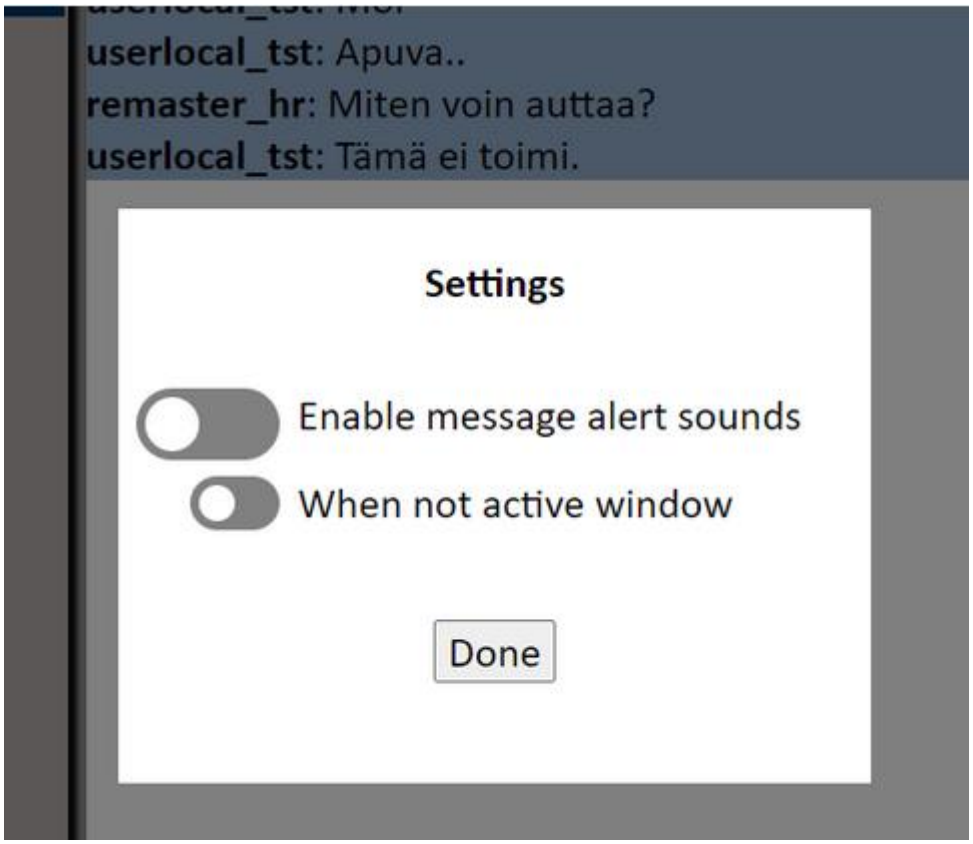
Kuvio 33. Windows-notifikaatioiden getPermission-funktiokutsu Chat.js-tiedostossa

Itse notifikaatio luodaan Chat-komponentin Socket.IO:n new-room-tapahtumassa sen jälkeen, kun uudelle asiakkaalle on luotu room-objekti tilamuuttujaan (ks. kuvio 34).

```
this.setState((prev) => {
    let ts = this.getTimestamp();
    return {
        rooms: [...prev.rooms, {
            name: room,
            messages: [],
            unread: null,
            connected: ts,
            disconnected: null,
            typing: [],
            convId: convId,
            lastMsgAt: new Date().toISOString().slice(0, 19),
            labels: []
        }]}
}, () => {
    //console.log(this.state.rooms);
    // if [redacted]chat-support page is not the active tab, or
    // if browser is not the topmost window, create windows notification
    // for a new client connection.
    if(document.hidden || !document.hasFocus()) notification.createNotification();
});
```

Kuvio 34. Windows-notifikaation luonti createNotification-funktiokutsulla Chat.js-tiedostossa

Sovelluksessa on myös mahdollista ottaa käyttöön sivun yläreunan hammasratasikonia painamalla äänivaroitukset (ks. kuviot 35, 36), kun asiakas lähettää tukisovellukseen viestin.



Kuvio 35. Tukisovelluksen ääniasetukset

```
<ChatSettings
    visible={this.state.showSettings}
    sounds={this.state.sounds}
    whenNotOnTop={this.state.whenNotOnTop}
    setSounds={(e) => {
        this.setState({sounds: e.target.checked, whenNotOnTop: false});
    }}
    setWhenNotOnTop={(e) => {
        if(e.target.checked) {
            this.setState({sounds: true, whenNotOnTop: true});
        }
        else {
            this.setState({whenNotOnTop: false});
        }
    }}
    onExit={() => {
        this.setState({showSettings: false});
    }}
/>
```

Kuvio 36. ChatSettings-komponentin renderöinti Chat-komponentin render-funktiossa

ChatSettings-komponentti renderöidään vain, kun hammasrataspainiketta on painettu. Chat-komponentin showSettings-tilamuuttujassa pidetään tietoa, pitääkö ChatSettings-komponentti renderöidä vai ei. ChatSettings-komponentti saa propseina tarvittavat arvot ja funktiot parent-komponentin tilan muuttamiseksi – setSounds ja setWhenNotOnTop. Komponentti (ks. kuvio 37) renderöidään modalina eli se tulee päällimmäiseksi sivustolle ja muu sisältö ”peittyy” läpinäkyvän, harmaan taustan taakse. Modalin voi sulkea klikkaamalla harmaata taustaa tai Done-painiketta.

```
const ChatSettings = ({visible, sounds, whenNotOnTop, setSounds, setWhenNotOnTop, onExit}) => {

    if(!visible) return null;
    return (
        <div className="chatSettings" onClick={(e) => {
            if(e.target == e.currentTarget) {
                onExit();
            }
        }}>
            <div className="settings">
                <h1>Settings</h1>

                <input type="checkbox" id="sounds" checked={!!sounds} onChange={(e) => {
                    setSounds(e);
                }}/>
                <label htmlFor="sounds" className="biggerlabel">Enable message alert sounds</label>

                <div className="secondarySettings">
                    <input type="checkbox" id="background" checked={!!whenNotOnTop} onChange={(e) => {
                        setWhenNotOnTop(e);
                    }}/>
                    <label htmlFor="background" className="smallerlabel">When not active window</label>
                </div>

                <input type="button" className="settingsDoneBtn" value="Done" onClick={onExit}></input>
            </div>
        </div>
    );
}
```

Kuvio 37. ChatSettings-komponentti ChatSettings.js-tiedostossa

4.3.3 Asiakaspuoli

Asiakaspuolen moduuli on kehitetty vanilla JavaScriptiä ja jQueryä käyttäen ja se voidaan ottaa käyttöön missä vain JavaScriptiä käyttävällä verkkosivulla. Asiakaspuolen moduulin riippuvuuksia ovat Socket.IO-client ja jQuery.

Asiakaspuolen moduuli, nimeltään ias_chat, alustetaan käyttämällä moduulin create-funktiota vaadituilla argumenteilla (ks. kuviot 38, 39).

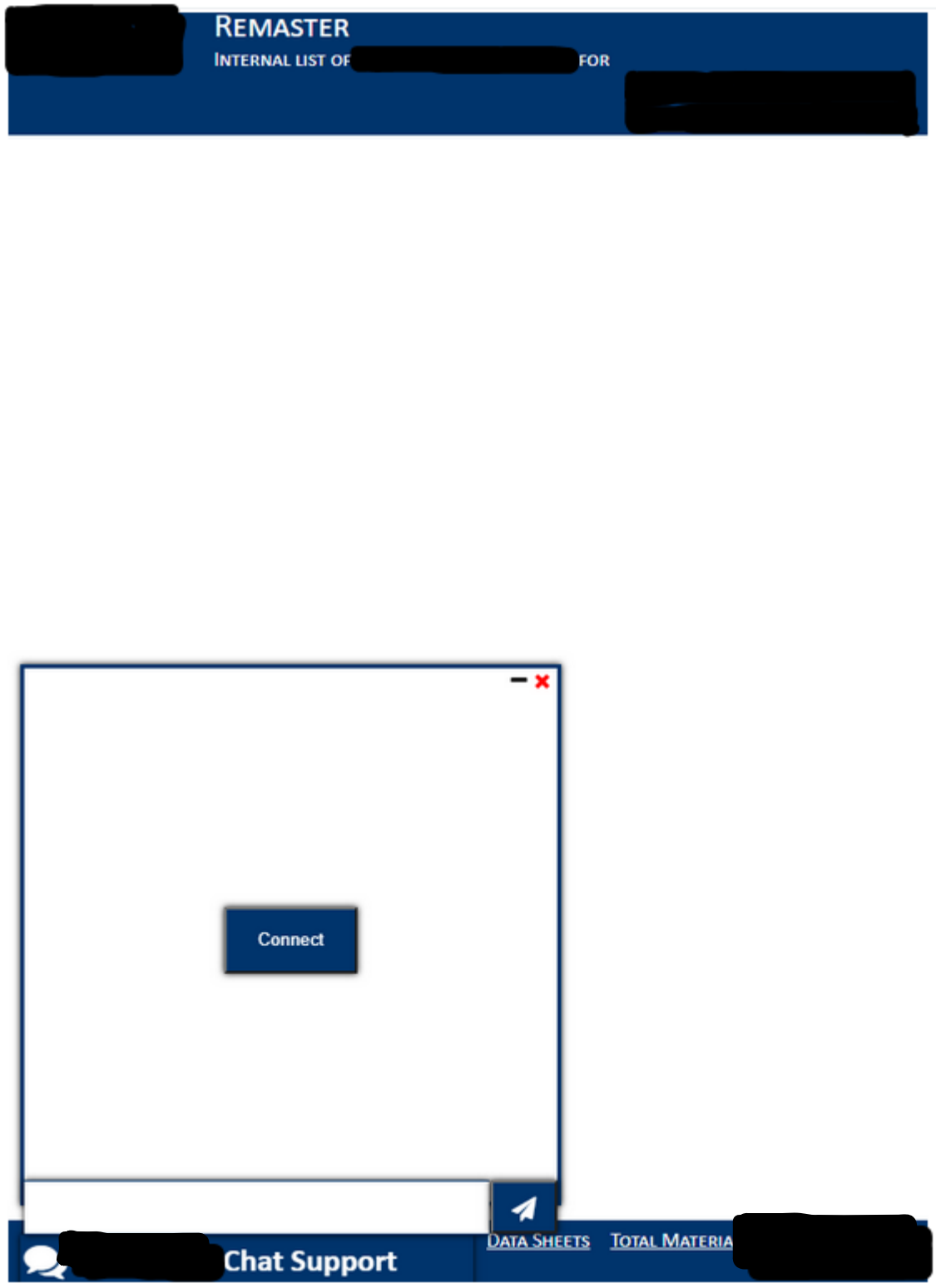
```
ias_chat.create(
    document.getElementsByName("container"),
    "blue",
    "black",
    username,
    email,
    productName
);
```

Kuvio 38. Asiakaspuolen moduulin alustus

```
/**
 * Creates a chatbox element positioned at the bottom left of the page view
 * with event handlers to start a socket connection and send/recv messages.
 *
 * @param {object} container        HTML parent element for chatbox
 * @param {string} backgroundColor
 * @param {string} textColor
 * @param {string} username         $username or something unique for each client
 *                                  ex: userlocal
 * @param {string} email            User email
 * @param {string} productName
 */
ias_chat.create = (container, backgroundColor, textColor, username, email, productName) => {
```

Kuvio 39. ias_chat-moduulin create-funktion parametrit

Kun asiakasmoduuli on alustettu, luo se verkkosivulle vasempaan alareunaan painikkeen, jota painamalla chat-näkymä aukeaa (ks. kuvio 40).

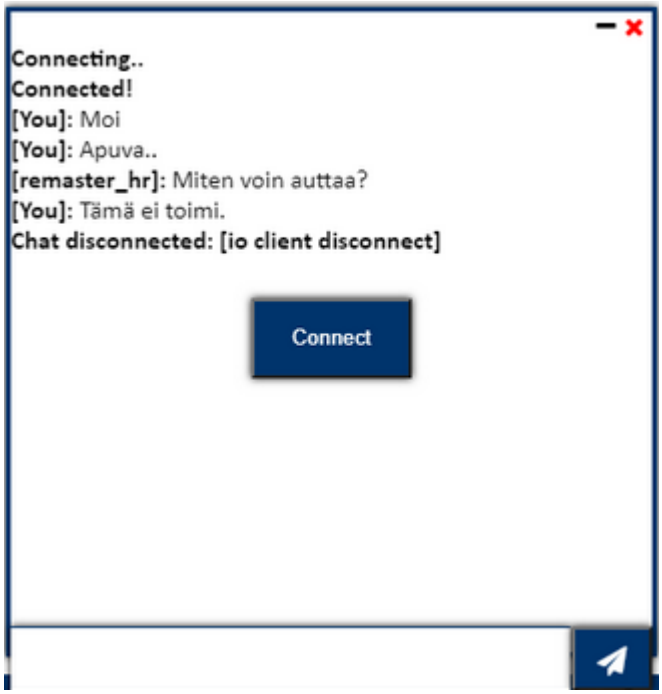


Kuvio 40. Asiakasmoduuli verkkosivulle alustettuna ja chat-näkymä avattuna

Connect-painiketta painamalla käyttäjä voi yhdistää tukipuolen sovellukseen ja aloittaa keskustelun tukihenkilöiden kanssa. Chat-ikkunan oikeassa yläreunassa näkyvät pienet miinus ja x-ikonit

ovat kuin Windows-sovellusten painikkeet, jotka minimoivat ja sulkevat sovelluksen. Chat-yhteys ei katkea, jos ikkuna minimoidaan, mutta x-painiketta painaessa yhteys sen sijaan katkeaa.

Seuraavassa kuvassa (ks. kuvio 41) asiakas on käynyt tukipuolen kanssa lyhyen keskustelun ja on sen jälkeen sulkenut yhteyden, jolloin asiakkaalla on taas mahdollisuus yhdistää tukipuoleen uudelleen niin halutessaan. Yhdistys tapahtuu ias-chat.js-tiedoston connect-tapahtumassa (ks. kuvio 42).



Kuvio 41. Chat-keskustelu käyty ja suljettu

```
socket.on('connect', () => {
    if (user.length == 0) {
        console.log("ERROR: socket.on connect -- no username found!");
        return;
    }
    appendMessage("Connected!");
    // create a room based on username, also send productname and email to save to db.
    socket.emit('create', {user, email, product});
    connected = true;
});
```

Kuvio 42. Tukipuoleen yhdistäminen Socket.IO:n connect-tapahtumassa ias-chat.js-tiedostossa

5 Tulokset

5.1 Lopullinen ohjelmisto

Sovelluksen lopputulos vastaa asiakkaan asettamia vaatimuksia ja toimii ongelmitta. Ainoa itseä kehityksen aikana mietityttämään alkanut asia sovelluksessa on tukipuolen frontendissä suurten datamäärien pitäminen Reactin tilamuuttujissa. Chat-komponentin rooms-tilamuuttuja saattaa kasvaa ajan myötä todella suureksi, koska se pitää sisällään kaikki asiakkaat ja asiakkaiden viimeisen 3kk:n aikana käydyt keskustelut. Kun tilamuuttuja kasvaa suureksi, saattaa esimerkiksi keskustelu-HTML-elementteihin lisätyt onHover-tapahtumat aiheuttaa viivettä sovelluksessa, kun myös niiden määrä kasvaa suureksi. Myös itse viestien renderöinti keskusteluhuoneisiin saattaa kestää hieman pidempään, kun datamäärä kasvaa ja jokin huone avataan. En usko, että aika on kuitenkaan kovin suuri ja kun keskusteluja renderöidään, voidaan käyttäjälle näyttää vaikka jokin latausanimaatio.

Varsinaisia vikoja sovelluksessa en ole havainnut.

5.2 Jatkokehitys

Sovelluksen kehitys jatkuu ainakin virheenkäsittelyn parantamisen ja todennäköisesti muutenkin asiakkaan toiveiden mukaisesti. Kun sovellus saadaan siirrettyä produktioserverille ja siten testikäytöstä todelliseen käyttöön, sovelluksessa havaitaan mahdollisesti puutteita tai muita parannusta vaativia ominaisuuksia lisääntyneen käyttäjämäärän seurauksena.

Rooms-tilamuuttujan kasvavaan kokoon voisi mahdollisesti reagoida ottamalla keskustelujen renderöinnissä käyttöön lazy-loading-ominaisuuden eli keskusteluja ei tarvitsisi ladata sivulle kaikkia kerralla, vaan kun chat-historiaa selataan ylöspäin vanhempiin viesteihin, tarvittavat HTML-elementit luotaisiin ja renderöitäisiin sivulle vasta siinä vaiheessa.

6 Pohdinta

Sovelluksen kehitys onnistui ongelmitta eikä mitään suurempia ongelmia tullut vastaan. Opettelin myös kehityksen aikana käyttämään Reactin uusia ominaisuuksia, kuten tilallisten funktionaalisten komponenttien ja niiden hookkien (useState, useEffect jne.) käyttämistä. Olen käyttänyt Reactia

aika paljon ennenkin, joten muuten kehitystyössä ei tullut oikein mitään uutta vastaan, muuta kuin pari kolmannen osapuolen kirjastoa ja niiden API:en selailu, kuten esim. Socket.IO. Olin päässyt tekemään myös jo aiemmin töitä työssä käytettyjen backend-kirjastojen, kuten connect-roles-kirjaston kanssa, joten backendissäkään ei tullut suurempia ongelmia vastaan. Suurimmat haasteet olivat todennäköisesti frontendin CSS-tyylittely sekä tukipuolen frontend-koodin refaktorointi siinä vaiheessa, kun lisäsin keskustelujen tallentamisen tietokantaan, koska sovellus oli kehitetty alun perin ilman tietokantaa ja viestit tallennettiin vain selaimen localStorageen.

Olen tyytyväinen opinnäytetyön lopputulokseen ja jatkan mielelläni sovelluksen kehittämistä ja parantelua, jos asiakas vain niin toivoo.

Lähteet

An Introduction to JavaScript. N.d. Javascript.info verkkosivusto. Sivua päivitetty 13.6.2021. Viitattu 19.6.2021. https://javascript.info/intro

Components and Props. N.d. Artikkeli reactjs sivustolla. Viitattu 19.6.2021. https://reactjs.org/docs/components-and-props.html

Hot Module Replacement. N.d. Artikkeli parceljs.org sivustolla. Viitattu 19.6.2021. https://parceljs.org/hmr.html

How It Works. N.d. Artikkeli parceljs.org sivustolla. Viitattu 19.6.2021. https://parceljs.org/how_it_works.html

Introduction to Node.js. N.d. Artikkeli nodejs.dev sivustolla. Viitattu 19.6.2021. https://nodejs.dev/learn

JavaScript. 2021. Artikkeli Mozillan sivustolla. Sivua päivitetty 4.5.2021. Viitattu 19.6.2021. https://developer.mozilla.org/en-US/docs/Web/JavaScript

JavaScript language resources. 2021. Artikkeli Mozillan sivustolla. Sivua päivitetty 21.5.2021. Viitattu 11.7.2021. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Language_Resources

Krill P. 2014. React: Making faster, smoother UIs for data-driven Web apps. Artikkeli infoworld sivustolla. Julkaistu 15.5.2014. Viitattu 19.6.2021. https://www.infoworld.com/article/2608181/react--making-faster--smoother-uis-for-data-driven-web-apps.html

Production. N.d. Artikkeli parceljs.org sivustolla. Viitattu 19.6.2021. https://parceljs.org/production.html

Siddique S. 2020. What does it mean by Javascript is single threaded language. Artikkeli Medium sivustolle. Julkaistu 17.6.2020. Viitattu 19.6.2021. https://medium.com/swlh/what-does-it-mean-by-javascript-is-single-threaded-language-f4130645d8a9

State and Lifecycle. N.d. Artikkeli reactjs sivustolla. Viitattu 19.6.2021. https://reactjs.org/docs/state-and-lifecycle.html

Using the State Hook. N.d. Artikkeli reactjs sivustolla. Viitattu 19.6.2021. https://reactjs.org/docs/hooks-state.html

What Socket.IO is. N.d. Artikkeli socket.io sivustolla. Viitattu 19.6.2021. https://socket.io/docs/v4