Heikki Sillanpää

Yhteystietoverkkosivuston kehittäminen Service Deskin tueksi

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

10.5.2022

Tiivistelmä

Tekijä:	Heikki Sillanpää
Otsikko:	Yhteystietoverkkosivuston kehittäminen Service Deskin tueksi
Sivumäärä:	46 sivua
Aika:	10.5.2022
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Service Desk Manager Leena Raukola Lehtori Simo Silander

---

Opinnäytetyön tavoitteena oli luoda uudistettu yhteystieto-verkkosivu CGI:llä toimivan Service Deskin olemassa olevan sivun tilalle. Yhteystieto-sivun tuli sisältää oman osion Service Deskin työtä tukeville tukitiimeille. Sivuston sisällön muokkaaminen piti olla mahdollista ilman ohjelmointiosaamista.

Työ toteutettiin luomalla ensin taustajärjestelmä ja tietokantayhteys käyttäen seuraavia teknologioita: MongoDB:tä, Mongoosea, Express.js:ää ja Node.js:ää. Taustajärjestelmän tarkoituksena oli tarjota menetelmiä käsitellä tietokannan dataa käyttöliittymässä.

Työn käyttöliittymäpuoli kehitettiin käyttämällä Reactia ja Axiosta. Käyttöliittymän kehityksessä keskityttiin esittämään data selkeästi ja tarjoamaan työkalut datan muokkaamiselle.

Insinöörityön tuloksena saatiin oma sivu tukitiimeille ja yhteystiedoille, joille molemmille on toteutettu vaaditut toiminnot lisätä ja muokata sivujen dataa ilman ohjelmointiosaamista. Tukitiimien sivu on tärkeä Service Deskille, sillä tukitiimien tiedot eivät ole olleet saatavilla yhdessä paikassa. Yhteystietojen sivu oli täynnä dataa ilman jäsentelyä, jonka vuoksi työssä jäsenneltiin yrityksen tiedot omiin säiliöihinsä ja luotiin taulukot yrityksen kontakteille.

Jatkokehityksenä verkkosivusto tarvitsisi tiedonsuodatustoimintoja ja hakutoiminnon. Sivusto voisi tarjota käyttäjille mahdollisuuden muokata sisältöä omien mieltymystensä mukaiseksi.

Avainsanat: Service Desk, React, Node.js, Express.js, MongoDB, Mongoose, Axios, yhteystiedot, tukitiimi, tukiryhmä, palvelutarjooma

Abstract

Author:	Heikki Sillanpää
Title:	Development of Contact Website to Support Service Desk
Number of Pages:	46 pages
Date:	10 May 2022
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Software Engineering
Supervisors:	Leena Raukola, Service Desk Manager
	Simo Silander, Senior Lecturer

The purpose of this graduate study was to develop a newly redesigned contact website for Service Desk operating at CGI. The site needed to contain its own page for support teams providing support for Service Desk. The option to edit the content of the website without any knowledge of programming was mandatory.

The development was started by building the backend of the software and creating a database connection using the following technologies: MongoDB, Mongoose, Express.js and Node.js. The purpose of the backend was to provide methods to manage database data from the client.

The frontend of the software was developed by using React and Axios. The focus of the client-side development was to represent the data in a user-friendly way and to provide tools to make changes to the data.

As a result, support teams and contacts got their own pages on the website and both pages have the function to make changes to the data without any knowledge of programming. The development of the support team page is significant for Service Desk because currently the information is not accessible in one place. The contact page was overflowing with data without providing any support for structuring. As a result, the company's data was structured inside a container and the company's contacts were put in a table.

As a further development, the website would need tools to filter and search data. The website could also allow more user personalization options.

Keywords: Service Desk, React, Node.js, Express.js, MongoDB, Mongoose, Axios, contacts, support team, support group, service provider

Sisällys

Lyhenteet

1 Johdanto 1

2 Service Desk ja ohjeet 2

2.1 Service Desk 2
2.2 Ohjeet 3

3 Contukin arkkitehtuuri 3

3.1 Käytetyt teknologiat 4
3.1.1 React.js 5
3.1.2 Node.js 9
3.1.3 Express.js 13
3.1.4 MongoDB ja Mongoose 15
3.1.5 Axios 16
3.2 Contukin rakenne 17

4 Taustajärjestelmän kehitys 19

4.1 Express-palvelin 21
4.2 Tietokanta 23
4.2.1 Yhteyden muodostaminen 23
4.2.2 Skeemat, mallit ja kokoelmat 24
4.3 Reititys ja CRUD 27

5 Käyttöliittymän kehitys 29

5.1 Reactin reititys 30
5.2 Näkymät 32
5.3 Tukitiimit 32
5.4 Yhteystiedot 38

6 Yhteenveto 41

Lähteet 43

Lyhenteet

CRUD: Create, Read, Update, Delete. Tiedon tallentamisen neljä perusoperaatiota.

DOM: Document Object Model. Dokumenttioliomalli, joka mahdollistaa selaimen elementtien muokkaamisen ohjelmallisesti.

HTTP: Hypertext Transfer Protocol. Protokolla, jota WWW-palvelimet käyttävät tiedonsiirtoon.

JSON: JavaScript Object Notation. Kevyt tiedonsiirtomuoto.

ODM: Object Document Mapper. Oliodokumenttien kartoittaja, jonka tarkoituksena on logiikka datan organisointiin.

ORM: Object-relational mapping. Ohjelmiston oliomallin mukaisen esityksen kuvaaminen relaatiotietokantamallin mukaiseksi esitykseksi ja kääntäen.

React: JavaScript-sovelluskehys.

URL: Uniform Resource Locator. URI:n erikoistapaus, jota käytetään osoittamaan WWW-sivuja.

URI: Uniform Resource Identifier. Merkkijono, jonka avulla kerrotaan tietyn tiedon paikka.

1 Johdanto

Insinöörityön tavoitteena on tehdä CGI:lle Service Deskin ohjeisiin uusi yhteystieto-verkkosivusto, jota on helppo lukea ja tiedot löytyvät nopeasti. Service Desk, josta tässä insinöörityössä puhutaan, on CGI:n tarjoama palvelu, johon asiakkaat voivat soittaa ja Service Deskin agentit pyrkivät auttamaan asiakkaita heidän teknisten ongelmien kanssa. Asiakas voi soittaa esimerkiksi unohtuneesta salasanasta, ohjelman toimimattomuudesta tai asiakas voi myös pyytää ohjeistusta ohjelmiston käytössä.

Tällä hetkellä Service Deskin yhteystietojen verkkosivut on optimoitu Internet Explorerille, sivustolla on runsaasti tekstiä ilman minkäänlaista käyttäjää helpottavaa ulkoasua. Sivusto sisältää eri yritysten kontakteja, yhteystietoja, aukioloaikoja, puhelinnumeroita, sähköpostiosoitteita ja toimipisteiden katuosoitteita. Sivustolle tulisi lisätä lista, josta Service Deskin työntekijät voisivat löytää tukitiimien tunnisteet tikettijärjestelmää varten.

Työn tekijällä ei ole pääsyä alkuperäisen yhteystietosivuston lähdekoodiin, eikä oikeaa dataa voida esitellä insinöörityössä salassapitosopimuksen vuoksi. Tarkoituksena on kehittää verkkosivusto teknologioilla, joiden kanssa ei ole runsaasti kokemusta, jotta voidaan samalla kehittää omaa osaamista uusien teknologioiden kanssa.

Tuloksena pitäisi syntyä verkkosivusto, jossa on oma osio tukitiimeille ja toinen osio yhteystiedoille. Tavoitteena on myös, että sivustolla olisi kaksi näkymää: käyttäjän- ja järjestelmänvalvojan näkymä. Järjestelmänvalvojan näkymän tulisi sisältää ominaisuuden, että sisältöä voisi lisätä/muokata/poistaa ilman ohjelmointiosaamista. Sekä käyttäjän että järjestelmänvalvojan näkymien tulisi olla helposti luettavaa ja tiedon etsiminen tehokasta, jonka vuoksi tietokantayhteyden muodostaminen, taustajärjestelmän lisääminen ja käyttöliittymän eheytys ovat tämän insinöörityön tavoitteita. Työn tilaajana on CGI Suomi Oy.

Tässä insinöörityössä kerrotaan ensin lyhyesti Service Deskistä, jonka jälkeen perehdytään ohjelmiston teknologioihin ja tutustutaan, kuinka ohjelmaa on kehitetty. Lopuksi käydään vielä läpi, mitä saatiin lopputulokseksi ja kuinka kehitystyö jatkuu insinöörityön jälkeen.

2 Service Desk ja ohjeet

Service Desk palvelee yrityksen asiakkaita tai omaa henkilökuntaa esimerkiksi tietokoneisiin tai ohjelmistoihin liittyvissä kysymyksissä ja pulmissa. Tässä insinöörityössä käsitellään Service Deskiä, joka palvelee yrityksen asiakkaita.

2.1 Service Desk

Service Deskissä ratkotaan yrityksen asiakkaiden kysymyksiä ja pulmia pääsääntöisesti puhelimitse, mutta työpyyntöjä tulee myös sähköisesti tikettijärjestelmän kautta. Tikettijärjestelmässä on karkeasti kahdenlaisia työpyyntöjä: palvelupyyntöjä tai häiriöitä.

Palvelupyynnöissä yleisesti kysytään neuvoa tai pyydetään käyttövaltuuksia johonkin järjestelmään. Palvelupyyntöihin kuuluvat myös erilaiset tilaukset ja asennukset, esimerkiksi sovelluksen asennuspyyntö tai uuden laitteen tilaaminen.

Häiriöihin taas yleisesti liittyy erilaiset järjestelmiin, laitteisiin tai ohjelmiin liittyvät viat ja ongelmat. Tämänlaisia voivat olla esimerkiksi fyysinen vika tietokoneessa tai kirjautumisen epäonnistuminen järjestelmään, vaikka kyseisellä asiakkaalla olisi tarvittavat käyttöoikeudet kirjautumista varten.

Puhelun aikana pyritään ratkaisemaan asiakkaan pyyntö tai ongelma ja tarkistetaan ohjeista nykyiset toimintatavat kyseiseen tapaukseen. Asiakas voi esimerkiksi pyytää asentamaan sovellusta. Tällöin tarkistetaan ohjeista, löytyykö

meiltä asennustiedostoa, saadaanko asentaa ilman erillistä lupaa ja asennetaanko sovellus käsin vai laitetaanko sovellus asentumaan yön ylitse jakeluna.

Tikettijärjestelmään kirjataan jokaisesta puhelusta työpyyntö. Ennen työpyynnön tekemistä tulee Service Deskin agentin tehdä päätös, kuuluuko asiakkaan puhelusta tehdä palvelupyyntö vai häiriö. Kun päätös on tehty, tehdään puhelua vastaava työpyyntö ja kirjataan asia järjestelmään. Mikäli asiakkaan pyyntö ratkesi puhelimessa, voi työpyynnön yleensä heti sulkea. Mikäli asiakkaan pyyntö ei ratkennut, selvitetään, kuinka asiakkaan pyynnön tai pulman voi ratkaista ja tarvittaessa ohjataan työpyyntö oikealle tukitiimille jatkoselvitykseen.

2.2 Ohjeet

Ohjeet ovat Service Deskin työntekijöiden päivittäisessä käytössä. Ohjeissa on tietoa asiakkaan kanssa sovituista toimintatavoista eri ohjelmistojen ja käytäntöjen kanssa. Ohjeista löytyvät myös eri ohjelmistojen ja järjestelmien vastuuhenkilöiden yhteystiedot ja toimintaohjeet vikatilanteissa.

Ohjeista tulisi varmistaa, toimitaanko nykyisten toimintatapojen mukaisesti. Esimerkki asiakkaan kanssa sovitusta käytännöstä voi olla vaikka, että salasanaa ei saa luovuttaa käyttäjälle ilman tunnistamista. Ohjeessa tulisi tällöin lukea myös kriteerit siihen, miten määritellään henkilön tunnistaminen.

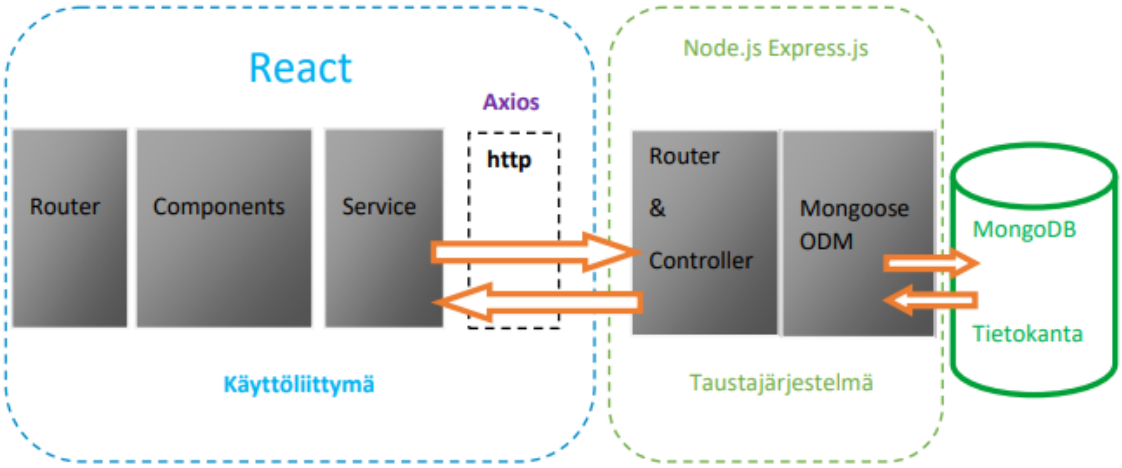
3 Contukin arkkitehtuuri

Annetaan ohjelmistolle nimeksi Contuki. Nimi tulee sanoista ”Contacts ja tukitiimit”. Osion tavoitteena on havainnollistaa Contukin arkkitehtuuria, esitellä käytettävät teknologiat ja esittää lukijalle suunniteltu puurakenne ohjelmiston koodille. Nämä tehdään sen vuoksi, jotta lukijan on helpompi seurata ja ymmärtää tämän ohjelmiston kehittämistä.

3.1 Käytetyt teknologiat

Tässä insinöörityössä päätettiin käyttää Contukin kehittämiseen seuraavia teknologioita: MongoDB:tä, Mongoosea, Node.js:ää, Express.js:ää, Axios ja React.js:ää. Työhön valikoitui teknologiat tekijän intresseistä kasvattaa kokemusta kyseisiin teknologioihin.

Teknologioista MongoDB on dokumenttitietokanta, Mongoose on Node.js:ään perustuva ODM-kirjasto MongoDB:lle [1], Node.js on JavaScriptin verkkopalvelin, Express.js on Node.js:n verkkokehys, Axios on lupaukseen perustuva http-asiakas Node.js:lle ja selaimelle [2], React.js on JavaScriptin sovelluskehys [3].



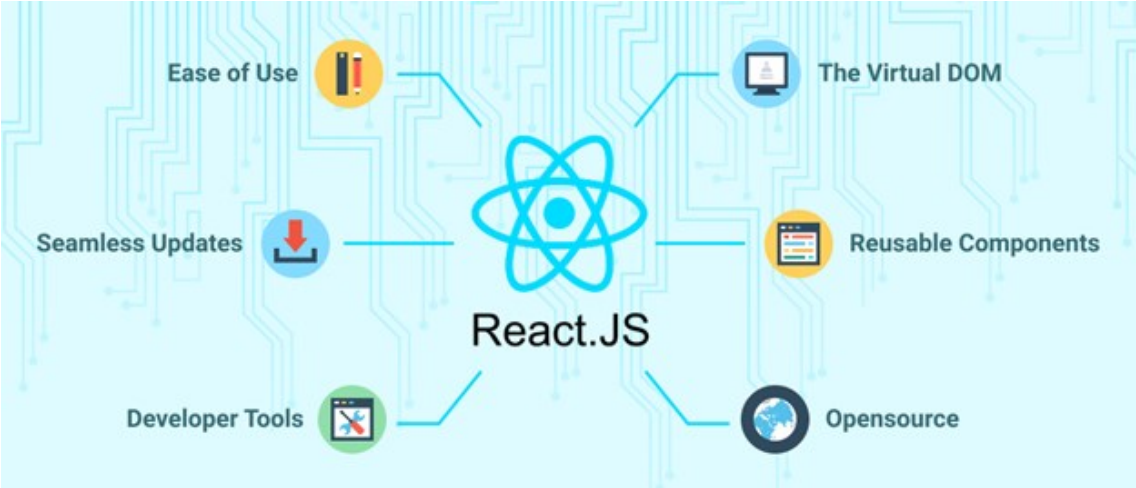
Kuva 1. Insinöörityön arkkitehtuuri.

Tässä projektissa kuvan 1 mukaisesti käytetään MongoDB-tietokannan datan käsittelyyn Mongoose ODM:ää. Mongoose on oliotietojen mallinnuskirjasto ja Mongoose toimii MongoDB-tietokannan käyttöliittymänä [4]. Node.js:n kehyksenä expressin tehtävänä on lisätä eri toiminnallisuuksia http-verbeille (GET, POST, DELETE, jne) ja käsitellä pyyntöjä eri URL-poluille (reitit), joihin pelkällä Node.js:llä eivät riitä ominaisuudet. Näiden Expressin toiminnallisuuksien avulla saadaan lähetettyä ohjelmistosta dataa Mongoosen kautta MongoDB-tietokantaan. [5.] React-asiakas lähettää http-pyyntöjä ja hakee http-vastauksia Axiosin avulla käyttäen komponenttien dataa. Palvelinpuolella Axios käyttää natiivia

Node.js http -moduulia, kun taas käyttöliittymäpuolella se käyttää XMLHttpRequesteja. [6.] React Routeria käytetään yksisivuisen verkkosivun rakentamisessa ja tarkoituksena on tehdä Contukista yksisivuinen verkkosivusto [7].

3.1.1 React.js

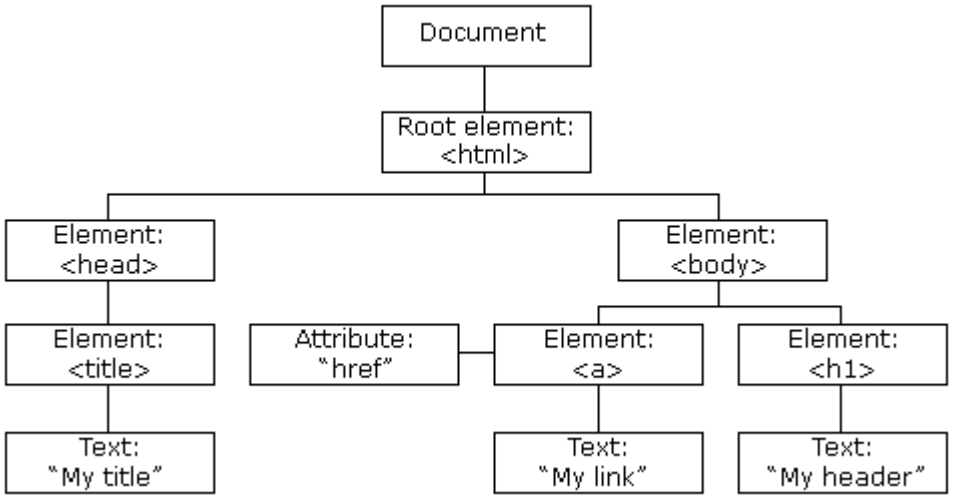
Reactia käytetään Contukin käyttöliittymän kehittämisessä. Käyttöliittymää tarvitaan, jotta verkkosivusta saadaan visuaalisesti tyylikäs ja nykyaikainen. Avataan lukijalle hieman, mikä React on ja mitä Reactin avulla tehdään. React.js on Facebookin kehittämä JavaScript-kirjasto, jota käytetään rakentamaan interaktiivisia verkkosivuja. React mahdollistaa haastavien käyttöliittymien kehittämisen yksinkertaisten komponenttien avulla, mutta React usein tarvitsee muita kirjastoja ratkaisuiden muodostamiseen. Näitä kirjastoja voi olla esimerkiksi React Router reitittämiseen ja isommissa sovelluksissa tilanhallintaan käytettävä kirjasto nimeltä Redux. React ei oleta mitään muista osista ratkaisussa. [8.] Reactin avulla kehitetään yksinkertaisia näkymiä jokaiselle sovelluksen tilalle, ja React päivittää ja esittää oikeat komponentit tilan muuttuessa. Komponentit voidaan kehittää siten, että ne hallinnoivat omaa tilaansa. Komponenttien logiikka on kirjoitettu JavaScriptillä eikä valmiilla pohjilla, jonka vuoksi voidaan pitää sovelluksen tilat DOM:in ulkopuolella. Reactin avulla voi kommunikoida taustajärjestelmän kanssa NodeJS:n avulla ja renderöidä dataa HTML:nä. [9.] Reactilla kehittäminen vaatii taitoja seuraavissa teknologioissa: HTML, CSS ja JavaScript [10].



Kuva 2. React.js:n ominaisuuksia [11].

Kuvassa 2 havainnollistetaan Reactin ominaisuuksia visuaalisesti. Reactin ominaisuuksista lyhyesti: Reactilla on virtuaalinen DOM, JSX, uudelleenkäytettävät komponentit, se on avointa lähdekoodia. Reactilla on Google Chromessa kehittäjän lisäosa. Reactin päivitykset testataan aina ensin Facebookissa, React.js on helppo oppia. Kerrotaan Reactin ominaisuuksista ja perusteista lisää seuraavassa, jotta osataan paremmin ymmärtää Contukin kehittämistä. [12.]

DOM (Document Object Model) tarkoittaa sitä, että selain luo asiakirjaoliomallin (DOM) verkkosivun latautuessa. HTML DOM on standardi HTML-elementtien hankkimiseen, muuttamiseen, lisäämiseen tai poistamiseen. Kuvassa 3 on esitelty DOM. [13.]



Kuva 3. HTML DOM -malli rakennettu oliopuuksi [14].

Reactilla on käytössä virtuaalinen DOM. Mitä etua siitä on? Reactille kerrotaan, missä tilassa halutaan, että käyttöliittymä on. Virtuaalinen DOM luo Reactin avulla tämän tilan selaimen DOM:iin. [15.] Sen sijaan, että manipuloitaisiin selaimen DOM:ia suoraan, React luo virtuaalisen DOM:in muistiin, missä tehdään kaikki tarvittavat manipuloinnit ennen kuin tehdään muutokset selaimen DOM:iin [16]. Virtuaalisen DOM:in päivittäminen on nopeaa ja selaimen DOM:in päivitys on hidasta. Virtuaalisen DOM:in etuna on se, että se vähentää kuormaa, jota tapahtuisi, mikäli kaikki muutokset tehtäisiin suoraan selaimen DOM:iin. [17.] Hyvää lisätietoa virtuaalisesta DOM:sta on osoitteessa https://www.codecademy.com/article/react-virtual-dom

Reactin kehys koostuu komponenteista, mikä mahdollistaa aloittamisen pienistä komponenteista ja vähitellen siirtyä isompiin. Tämä tarkoittaa, että voidaan aloittaa kehittäminen pienistä painikkeista ja laatikoista, jonka jälkeen yhdistetään ne osaksi isompaa kokonaisuutta. Reactissa on tarkoituksena tehdä komponentteja, joita voidaan käyttää uudelleen, eli komponentit voidaan kehittää siten, että niihin ei ole laitettu pysyviä arvoja. [18.]

```
const MapToDb = (formArray) => {
    //maps the array from Form into object for DB
    let temp = {}

    formArray.forEach(item => {
        temp[item.id] = item.value
    })
    console.log('Ready for DB. temp: ', temp)
    return temp
}

export default MapToDb
```

Esimerkkikoodi 1. Uudelleenkäytettävä komponentti Contukista.

Yllä esimerkkikoodissa 1 on esimerkki uudelleenkäytettävästä komponentista. Vakio MapToDb saa ominaisuudekseen lomakkeen sisällön listana, joka on nimetty formArray:ksi. Lomakkeen sisältö muutetaan komponentin avulla olioksi, jonka voi siirtää tietokantaan.

JSX on Reactin käyttämä XML/HTML-tyylinen syntaksi, jotta XML/HTML-tyylistä tekstiä voi olla Javascript/React -koodin seassa. Käytännössä tarkoittaa sitä, että voidaan kirjoittaa HTML-XML-tyylisiä rakenteita samaan tiedostoon, missä kirjoitetaan JavaScript-koodia ja käännetään (esimerkiksi Babel:illa https://babeljs.io/) ilmaisut oikeaksi JavaScript-koodiksi. [19.]

Reactissa komponentin tila on kuin tietovarasto komponentille, jota käytetään komponentin päivittämisessä, kun käyttäjä esimerkiksi painaa painiketta tai kirjoittaa tekstiä. React.Component on kaikkien luokkapohjaisten React.js -komponenttien perusluokka. Kun luokka perii luokan React.Component, sen konstruktori määrittää luokalle automaattisesti tilan ja laittaa sen arvoksi null. Arvoa voi muuttaa ohittamalla constructor-menetelmän. [20.]

Koukut ovat JavaScript-toimintoja, jotka on lisätty Reactiin versiossa 16.8. ja joiden avulla voidaan käyttää Reactin tilaa ja muita ominaisuuksia ilman luokkia [21]. Koukut ovat toimintoja, joiden avulla voidaan hypätä keskelle Reactin tilaa tai muihin toimintokomponenttien elinkaarten ominaisuuksien väliin. Koukut eivät toimi luokkien sisällä vaan ainoastaan toimintokomponenttien sisällä.

Koukkuja voidaan kutsua vain ylimmällä tasolla, eli niitä ei voi kutsua esimerkiksi silmukoiden sisällä. [22.]

3.1.2 Node.js

Node.js:ää käytetään Contukin taustajärjestelmässä. Se valittiin käytettäväksi teknologiaksi, koska tiedettiin Node.js:n olevan varma valinta verkkosivun kehittämiseen. Kerrotaan seuraavaksi tarkemmin Node.js:tä.

Node.js on Googlen V8-JavaScript-moottoriin perustuva JavaScriptin suoritusympäristö. JavaScript-moottori on riippumaton selaimesta, jossa sitä isännöidään. Tämä ominaisuus teki Node.js:tä suositun. [23.]

Node.js on asynkroninen tapahtumaohjattu ajoympäristö JavaScript-koodin suorittamiseen palvelimella ja se on suunniteltu skaalautuvien verkkosovellusten rakentamiseen. Node.js käyttö on ilmaista ja se toimii usealla eri alustalla (mm. Windows, Linux, Unix, Mac OS X). Node.js:n voi asentaa tietokoneelle osoitteesta: https://nodejs.org/en/download/ [24; 25].

Yleinen tehtävä verkkopalvelimelle on, että sen tulisi avata tiedosto palvelimella ja palauttaa sen sisältö käyttöliittymälle. Node.js hallitsee pyynnön seuraavasti:

1. Lähettää tehtävän tietokoneen tiedostojärjestelmään.

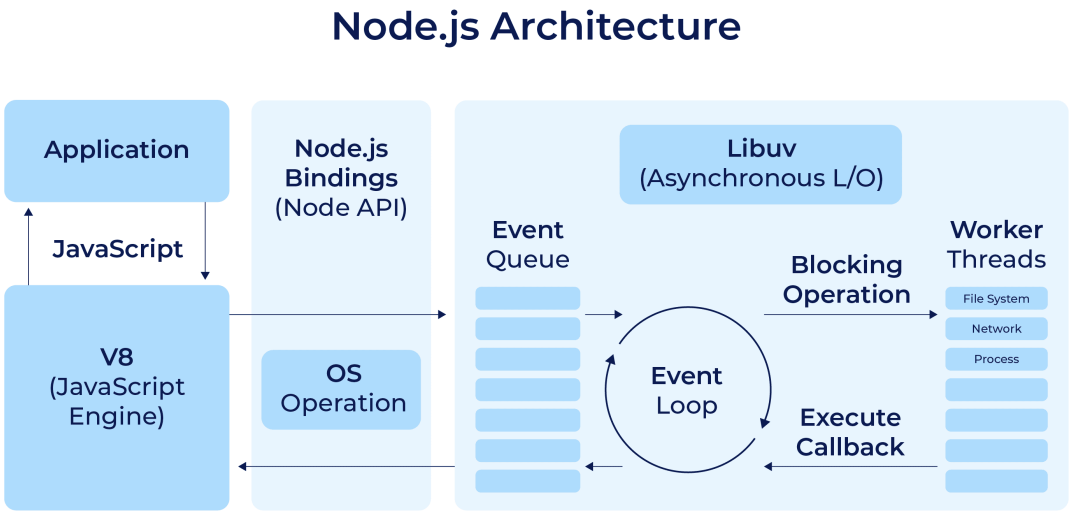
2. On valmis hallinnoimaan seuraavaa pyyntöä.

3. Kun tiedostojärjestelmä on avannut ja lukenut tiedoston, palvelin palauttaa sisällön käyttöliittymään.

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

// luodaan palvelinolio
http.createServer(function (request, response) => {
  response.writeHead(200, { 'Content-Type': 'text/html' }); // header

var url = request.url;
  if(url===’/supteams’){
     response.write(‘<h1>supteams sivu</h1>’);
     response.end();
  }else if(url===’/contact’){
     response.write(‘<h1>contact sivu</h1>’);
  }else{
     response.write(‘<h1>mikä vaan muu sivu</h1>’);
     response.end();
  }
})

const PORT = 3000
app.listen(PORT)
console.log(`Server running on port ${PORT}`)
```

Esimerkkikoodi 2. Yksinkertainen Node.js-palvelin, jossa huomataan, että Node.js turvautuu reitityksessä ehtolauseisiin, minkä vuoksi ehtolauserakenteista tulee nopeasti valtavia ja luettavuus kärsii.

Node.js suorittaa yksisäikeistä, estämätöntä (non-blocking), asynkronista ohjelmointia, joka on hyvin muistitehokasta. Estämättömällä tarkoitetaan, että Node.js ei jää odottamaan esimerkiksi rajanpintaan tehtyjen kyselyiden vastauksia, vaan siirtyy suoraan seuraavaan pyyntöön ja pyrkii aina suorittamaan koodin, mikäli se on mahdollista. Kuvassa 4 havainnollistetaan Node.js:n toiminta visuaalisesti. [25.]



Kuva 4. Node.js:n arkkitehtuuri [26].

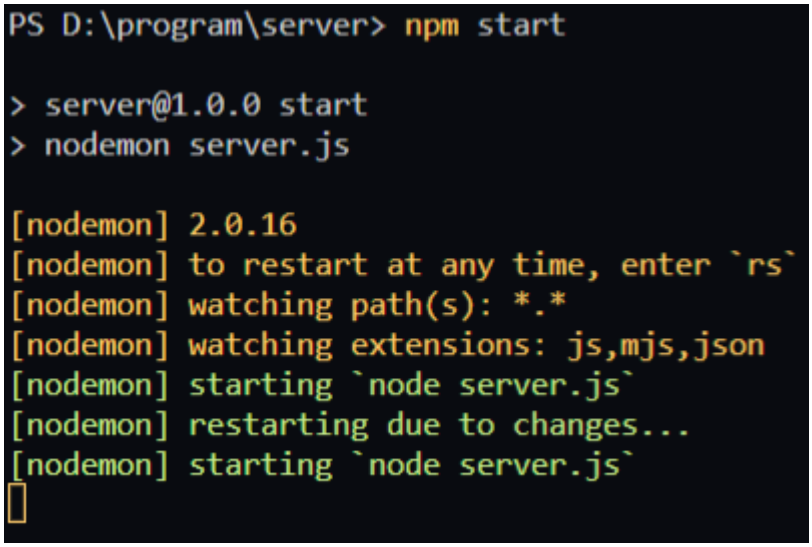
Node.js:n avulla voi rakentaa dynaamista sisältöä. Node.js voi lisätä, avata, lukea, kirjoittaa, poistaa ja sulkea tiedostoja palvelimella. Node.js:n avulla voi kerätä tietoa lomakkeilta sekä lisätä, poistaa ja muokata tietokannan dataa. [27.]

Toinen Node.js:n ominaisuus, joka on edistänyt Node.js:n suosiota on npm (node package manager) eli paketinhallintajärjestelmä. Npm:ää käytetään sekä taustajärjestelmässä että käyttöliittymässä. Npm on standardi riippuvuuksien määrittelyyn ja sen avulla säilytetään kolmannen osapuolen lähdekoodia sekä hallitaan asennettuja moduuleja tietokoneella. Npm:llä voi tehdä monia toimintoja, kunhan projektista löytyy package.json-tiedosto, jonka voi asentaa esimerkkikoodi 3 ensimmäisellä npm-komennolla init. Package.json-tiedosto sisältää metadataa ja perustietoa ohjelmasta, jonka lisäksi se hallinnoi kaikkia ohjelman riippuvuuksia ja paketteja, joita asennetaan npm:n avulla. Npm:n avulla voidaan asentaa kaikki riippuvuudet kerralla, asentaa yksi paketti kerrallaan, päivittää jokin tietty paketti tai suorittaa tehtäviä, kuten Reactin ajaminen. [28.]

```
§ npm init
§ npm install
§ npm install <paketti>
§ npm update
§ npm update <paketti>
§ npm run <tehtävän nimi>
```

Esimerkkikoodi 3. Erilaisia npm-komentoja. <paketti> tilalle löytyy paketteja osoitteesta: https://www.npmjs.com/

Npm:llä voi tallentaa joko paikallisesti tai globaalisti riippuvuuksia tai moduuleja. Contukin kehityksessä tallennettiin paikallisesti riippuvuuksiin esimerkiksi nodemon, jonka tarkoituksena on nopeuttaa Node.js-pohjaisen sovelluksen kehittämistä, sillä se automaattisesti käynnistää Node-ohjelman uudelleen, kun tiedostomuutoksia huomataan hakemistossa. [29.]



Kuva 5. Contukin taustajärjestelmä käynnistetään npm-komennolla start. Kuvassa näkyy, miten nodemon käynnistää palvelimen uudelleen muutoksen havaittuaan.

Kaikki npm:n paketit määritetään kansiossa nimeltä package.json ja sen sisältö kirjotetaan JSON:lla [30].

```
{
    "name": "server",
    "version": "1.0.0",
    "description": "",
    "main": "server.js",
    "scripts": {
        "test": "echo \"Error: no test specified\" && exit 1",
        "start": "nodemon server.js"
    },
    "keywords": [],
    "author": "Heikki",
    "license": "ISC",
    "dependencies": {
        "cors": "^2.8.5",
        "dotenv": "^16.0.0",
        "express": "^4.17.3",
        "mongoose": "^6.3.0"
    }
}
```

Esimerkkikoodi 4. Contukin package.json -tiedosto.

Esimerkkikoodissa 4 on laitettu skripteihin komento ”start”: ”nodemon server.js”, joka tarkoittaa, että komennolla ”npm start” taustajärjestelmä käynnistyy.

3.1.3 Express.js

Contukin kehitykseen valikoitui Express, koska Node.js ja Express toimivat hyvin yhdessä ja haluttiin kehittää osaamista Expressin kanssa. Expressin avulla on huomattavasti helpompaa kehittää taustajärjestelmään toiminnot kuin Node.js:llä. Tämä huomataan vertaamalla esimerkkikoodeja 2 ja 5.

Express on verkkokehys, jonka avulla voidaan jäsennellä verkkosovellus käsittelemään useita http-kutsuja halutussa URL-osoitteessa [31]. Express.js ja muut ohjelmointirajapinnassa tarjottavat kirjastot on tehty helpottamaan Nodella tapahtuvaa web-sovellusten ohjelmointia [32]. Express tarjoaa menetelmiä määrittämään, mitä funktioita kutsutaan tietylle http-verbille (GET, POST, SET, jne) ja URL:ille (polku/reitti). Expressin avulla voidaan käyttää mitä tahansa tietokannan toimintoa, jota Node.js tukee, mutta Express ei itse määritä mitään tietokantaan liittyviä toimintoja. [33.]

Contukin kehittämisessä käytetään Expressiä kahteen sen päätarkoitukseen: http-menetelmiin ja reititykseen. Reitityksestä on tarkemmin luvussa 4.3 ja http-menetelmistä lisää luvussa 4.3. Lisätään Express projektiin npm:n avulla, katso https://www.npmjs.com/package/express.

```
const express = require('express')
const app = express()

app.get('/', (req, res) => {
  res.send('<h1>Etusivu</h1>')
})

app.get('/supteams', (req, res) => {
  res.send('<h1>supteams sivu</h1>')
})

app.get('/contact, (req, res) => {
  res.send('<h1>contact sivu</h1>')
})


const PORT = 3000
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`)
})
```

Esimerkkikoodi 5. Muutettiin esimerkkikoodin 2:n palvelin käyttämään Expressiä. Huomataan kirjoitettavan koodin määrän vähentyvän ja saadaan koodi siistimmäksi ja ymmärrettävämmäksi.

Esimerkkikoodissa 5 tapahtumankäsittelijäfunktiolla on kaksi parametria, request ja response (request funktiosta lisää osoitteessa: http://expressjs.com/en/4x/api.html#req ja response funktiosta lisää osoitteessa: http://expressjs.com/en/4x/api.html#res). Ensimmäinen parametri request sisältää kaikki http-pyynnön tiedot ja toisen parametrin responsen avulla määritellään, kuinka pyyntöön vastataan. Koodissa pyyntöön vastataan response-olion menetelmällä send (lisää tietoa osoitteessa: http://expressjs.com/en/4x/api.html#res.send). Palvelin vastaa http-pyynnön kutsuun lähettämällä vastaukseksi selaimelle merkkijonon <h1>Etusivu</h1>, joka oli send:in parametrina esimerkkikoodissa 5. Parametrin ollessa merkkijono asettaa Express vastauksessa Content-Type-headerin arvoksi text/html, jolloin statuskoodiksi tulee oletusarvoisesti 200. Esimerkkikoodissa 5 toinen reiteistä

määrittelee tapahtumankäsittelijän, joka huolehtii sovelluksen polkuun ”/api/notes” tulevat HTTP GET -pyynnöt. [34.]

3.1.4 MongoDB ja Mongoose

Contukissa päädyttiin käyttämään tietokantana MongoDB:tä ja Mongoosea sen vuoksi, että relaatiotietokannat ovat tutumpia ja haluttiin tässä työssä tutustua enemmän dokumenttitietokantoihin. Tietokantaa tarvitaan Contukissa yhteystietojen ja tukitiimien tietojen tallentamiseen. Contukissa käytetään MongoDB Atlasta (https://www.mongodb.com/atlas/database) ja Mongoosea, jonka voi asentaa projektiin npm-komennolla. MongoDB Atlakseen yhdistämisen ohjeet löytyvät osoitteesta: https://fullstackopen.com/osa3/tietojen_tallettaminen_mongo_db_tietokantaan#mongo-db.

MongoDB on asiakirjasuuntautunut, No-SQL tietokanta, johon talletetaan sovelluksen dataa JSON-muodon kaltaisesti. Käyttöliittymäpuolella voidaan luoda JSON-tiedostoja ja ne lähetetään Express-palvelimelle, jossa ne voidaan prosessoida ja tallentaa suoraan tietokantaan. Githubissa on lyhyt johdanto dokumenttitietokantoihin: https://github.com/fullstack-hy2020/misc/blob/master/dokumenttitietokannat.MD.


```
_id: ObjectId("626e4ff50f1cd47abb6db293")
title: "Otsikko"
support_group: "Tukiryhmä"
service_provider: "Palvelutarjooma"
description: "Kuvaus"
createdAt: 2022-05-01T09:16:37.954+00:00
updatedAt: 2022-05-01T09:25:46.021+00:00
__v: 0
```


Kuva 6. Kuvankaappaus tukitiimin datasta tietokannasta.

Käyttöliittymästä lähetetty data on alla olevan esimerkkikoodin 6 näköistä.

```
{
     title: 'Otsikko',
     support_group: 'Tukiryhmä',
     service_provider: 'Palvelutarjooma',
     description: 'Kuvaus'
}
```

Esimerkkikoodi 6. Tietokantaan lähetettävä data näyttää tukitiimeillä tältä. Miltä data näyttää tietokannassa. Tämän voi katsoa kuvasta 6.

Mongoosen tarkoituksena on antaa kehittäjien tehdä sovellustasolla skeemoja. Skeema viittaa MongoDB:ssä kokoelmaan ja skeema määrittää dokumentin rakenteen kokoelmassa. Skeemalle voidaan luoda säännöt ja rakenteet, joiden avulla päätetään, mitä skeemalta halutaan. Skeemaoliosta luodaan malliolio ja mallin sisältämiä funktioita käytetään MongoDB:n kanssa kommunikoimiseen. [35.]

```
const schema = new mongoose.Schema({ name: 'string', size: 'string'
});
const SupTeam = mongoose.model(SupTeam, schema);
```

Esimerkkikoodi 7. Tässä luodaan ensin uusi Mongoosen skeema, jossa määritetään kaksi kenttää: nimi (tyyppi: merkkijono) ja koko (tyyppi: merkkijono). Skeeman luonnin jälkeen luodaan skeemasta malli [35].

Mongoosen mallit ovat konstruktoreja, jotka on koottu skeeman määritelmistä. Mallin esiintymää kutsutaan dokumentiksi. Mallit ovat vastuussa dokumenttien luomisesta ja lukemisesta taustalla olevasta MongoDB-tietokannasta. [36.]

Lisäksi Mongoose tarjoaa myös erilaisia koukkuja, mallien validointia ja muita ominaisuuksia, joiden tarkoituksena on helpottaa MongoDB:n käyttöä [37].

3.1.5 Axios

Taustajärjestelmän ja käyttöliittymän välinen kommunikaatio tapahtuu Contukissa Axioksen avulla. Axios on lupaukseen perustuva http-asiakas selaimelle ja Node.js:lle. Axioksen avulla lähetetään http-kutsuja Node.js:lle. Reactissa Axios on kirjasto, joka luo ulkoisesti läsnä olevia http-pyyntöjä. Alla olevassa esimerkkikoodissa 8 luodaan Axioksen ilmentymä, jota kutsutaan esimerkkikoodissa 9. [38.]

```
import axios from 'axios'

export default axios.create({
    baseURL: 'http://localhost:5000/api',
    headers: {
        'Content-type': 'application/json',
    },
})
```

Esimerkkikoodi 8. Luodaan Axioksen ilmentymä, jonka tiedostonimi on http-common.js. baseURL on osoite, jossa taustajärjestelmä on käynnissä [39].

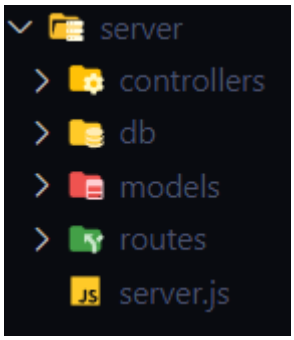
Käyttöliittymäpuolella Contukissa tehdään yhteystietojen (henkilö) http-kutsut seuraavasti. Katso koodiesimerkki 9.

```
import http from '../http-common'

class ContactDataService {
    getAll() {
        return http.get('/contact/findall')
    }
    get(id) {
        return http.get(`/contact/findone/${id}`)
    }
    create(data) {
        return http.post('/contact/create', data)
    }
    update(id, data) {
        return http.put(`/contact/update/${id}`, data)
    }
    delete(id) {
        return http.delete(`/contact/delete/${id}`)
    }
    deleteAll() {
        return http.delete(`/contact/deleteall`)
    }
}

export default new ContactDataService()
```

Esimerkkikoodi 9. Yhteystietojen datapalvelu, joka käyttää Axioksen oliota http-kutsujen lähettämiseen käyttöliittymäpuolella. Tiedostonimi: Contact.Service.js

3.2 Contukin rakenne

Contukissa on tarkoituksena soveltaa MVC-mallin (Model-View-Controller) mukaista arkkitehtuuria.



Kuva 7. Suunniteltu puurakenne taustajärjestelmän koodille.

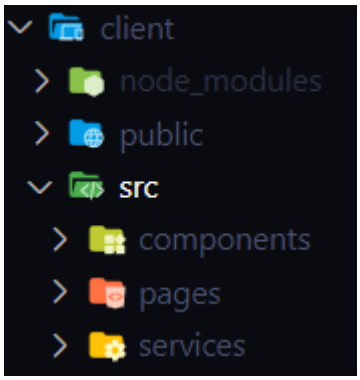
MVC:ssä työnkulku perustuu kolmeen tärkeään ohjelmiston tehtävään ohjaimiin, reitteihin ja malleihin.

Malli (Model): Mallit edustavat datan rakennetta, muotoa ja rajoituksia, joilla sitä tallennetaan. Mallit ovat käytännössä ohjelmiston dataosio.

Näkymä (View): Näkymät ovat sitä, mitä käyttäjä näkee, eli käytännössä käyttöliittymä. Näkymät käyttävät malleja ja esittävät datan siinä muodossa, missä käyttäjä sen haluaa. Käyttäjälle voidaan myös antaa lupa tehdä muutoksia näkymään.

Ohjain (Controller): Ohjaimet muodostavat yhteyden reittien ja näkymien välillä. Ohjaimet ohjaavat käyttäjän pyyntöjä ja antaa asianmukaisen vastauksen, joka syötetään katsojalle. Ohjain hallitsee käytännössä pyyntöjä ja vastauksia. [40.]

Reitit (Routes): Reitit hallitsevat kaikkia reittejä projektissa. Reitit viittaavat siihen, kuinka Contukin päätepisteet (URI) vastaavat asiakkaan pyyntöihin. Reitit määritellään käyttämällä Express-sovellusolion menetelmiä, jotka vastaavat http-menetelmiä, esimerkiksi app.get() käsittelee GET-pyyntöjä ja app.post() käsittelee POST-pyyntöjä [41; 42].



Kuva 8. Suunniteltu puurakenne käyttöliittymän koodille.

Suunnitelmana on, että käyttöliittymäpuolella on src-kansion sisällä kolme kansiota, yksi komponenteille, yksi sivuille ja yksi palveluille.

4 Taustajärjestelmän kehitys

Tässä luvussa esitellään taustajärjestelmän kehittämistä ja toiminnallisuuksien lisäämisiä: Express-palvelimeen yhdistäminen, tietokantayhteys, mallien toteutus tietokantaa varten, ohjelmiston reititys ja CRUD-toimintojen toteutus. Taustajärjestelmän avulla saadaan tallennettua Contukissa halutut tiedot talteen tietokantaan ja tarvittaessa voidaan lisätä, päivittää tai poistaa niitä. Näitä tietoja ovat esimerkiksi lomakkeisiin syötetyt tiedot. Tästä on lisää luvussa 5.2. Lomakkeiden käsittely.

Taustajärjestelmässä kehittäminen alkaa package.json-tiedoston luomisella eli ”npm init” -komennolla. Tiedoston tarkoitus on huolehtia Contukin riippuvuuksista, ajaa komentoja, dokumentoida ohjelmistoa ja alustaa taustajärjestelmä.

Lisätään Contukin taustajärjestelmään ympäristömuuttuja dotenv [43]. Ympäristömuuttujien avulla saadaan muun muassa salasanat piiloon näkyvästä koodista (koodiesimerkki 10).

```
§ npm install dotenv –save
```

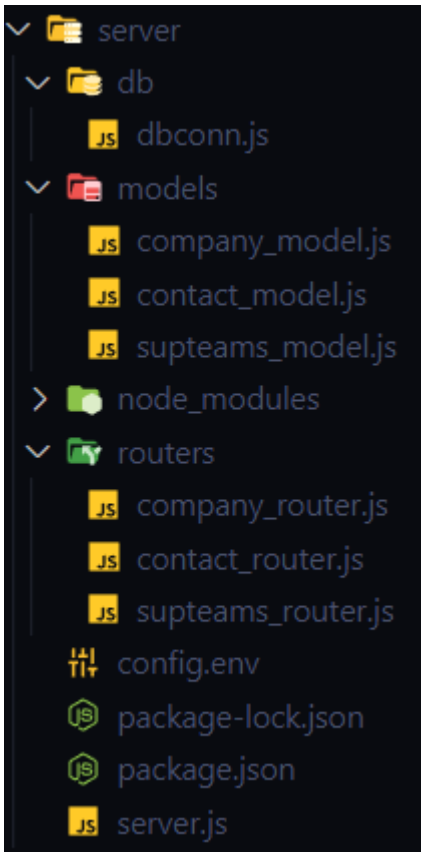
Esimerkkikoodi 10. Npm-komento dotenv asentamiseen. –save tarkoittaa, että tallennetaan dotenv package.json riippuvuuksiin.

Koodiesimerkissä 11 näkyy, kuinka ympäristömuuttuja toimii Express-palvelimen koodissa.

```
require('dotenv').config({ path: './config.env' })

const express = require('express')
const app = express()

const port = process.env.PORT || 5000

app.listen(port, () => console.log(`Server running on port ${port}`))
```

Esimerkkikoodi 11. Contukissa process.env.PORT tarkistaa, onko ympäristömuuttujaan PORT jo tallennettu portin arvo. Mikäli ei ole, niin käytetään porttia 5000.

Muutettiin taustajärjestelmän puurakennetta Contukissa ja päätettiin poistaa controller-kansio kokonaan ja yhdistetään kontrollerien ja reitittimien toiminnallisuus routers-kansion alle.



Kuva 9. Taustajärjestelmän puurakenne Contukissa.

4.1 Express-palvelin

Express-palvelin on rakennettu Contukissa server.js-tiedoston sisälle (kuva 9). Expressin päätarkoitus Contukissa on reititys ja http-pyynnöt.

```
// Tuonnit
require('dotenv').config({ path: './config.env' })

const express = require('express')
const app = express()

const port = process.env.PORT || 5000

// väliohjelmistot

// Tietokantaan yhdistäminen

// Reitittimet

app.listen(port, () => console.log(`Server running on port ${port}`))
```

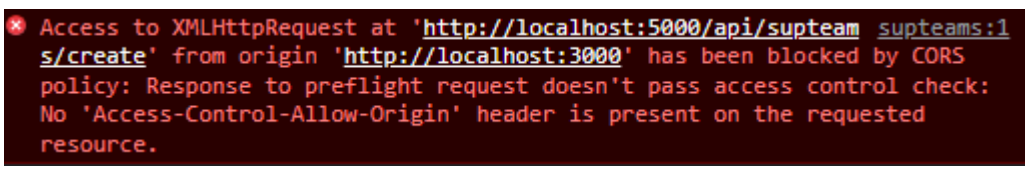
Esimerkkikoodi 12. Suunniteltu rakenne Contukin palvelintiedosto server.js:n kehittämiselle.

Määritetään Express käyttämään väliohjelmistoja cors (cross-origin resource sharing), express.json([asetukset]) ja express.urlencoded([asetukset]).

```
app.use(cors())
app.use(express.json())
app.use(express.urlencoded({ extended: true }))
```

Esimerkkikoodi 13. Contukissa olevat väliohjelmistot. Lisätään esimerkkikoodi 12 kohtaan ”Väliohjelmistot”.

Cors on Node.js-paketti, joka tarjoaa Connect/Express-väliohjelmiston, jonka avulla voidaan käyttää Corsia eri asetuksilla [44]. Cors on http-otsikkopohjainen mekanismi, jonka avulla palvelin voi ilmoittaa minkä tahansa alkuperän (verkkotunnuksen, mallin tai portin) kuin omansa, josta selaimen tulee sallia resurssien lataaminen [45]. Esimerkkikoodissa 12 sallitaan kaikki cors-pyynnöt. Mikäli cors ei olisi käytössä, ohjelma antaisi käyttöliittymässä tietokantaan dataa lisätessä virheilmoituksen, joka valittaa eri lähteiden välisestä resurssien jakamisesta. Katso kuva 10 virheilmoituksesta.



Kuva 10. Virheilmoitus selaimen konsolissa, kun lähetetään käyttöliittymästä dataa tietokantaan silloin, kun cors:ia ei ole otettu käyttöön.

Express.json() on väliohjelmafunktio Expressissä, joka jäsentää saapuvat pyynnöt JSON-hyötykuormilla. Tämä väliohjelmafunktio on tehty body-parserin pohjalta. Body-parserista löytyy tietoa: http://expressjs.com/en/resources/middleware/body-parser.html [46]. Express.urlencoded() on myös Expressin väliohjelmisto, joka jäsentää saapuvat pyynnöt urlenkoodatuilla hyötykuormilla ja sekin perustuu body-parseriin.

Tuodaan reitittimet server.js-tiedostoon routers-kansiosta ja otetaan reitittimet käyttöön koodiesimerkissä 14.

```
const supteamRouter = require('./routers/supteams_router')
const companyRouter = require('./routers/company_router')
const contactRouter = require('./routers/contact_router')
…
app.use('/api/company', companyRouter)
app.use('/api/contact', contactRouter)
app.use('/api/supteams', supteamRouter)
```

Esimerkkikoodi 14. Reitittimien tuonnit ja käyttöönotto Contukissa.

4.2 Tietokanta

4.2.1 Yhteyden muodostaminen

Luodaan db-kansion sisälle tiedosto nimeltä dbconn.js, jossa määritetään tietokantayhteys (kuva 9). Tietokantayhteyden muodostamiseen käytetään Mongoosen funktiota: connect(uri, options). Funktio haluaa osoitteen, johon yhdistetään ja asetukset, joita voi olla useampia. [47.] Esimerkkikoodissa 15 ”uri”-osoite on piilotettu ympäristömuuttujan CONN_URI taakse config.env-tiedostoon, koska se sisältää käyttäjätunnuksen ja salasanan. Asetuksia ovat tässä tapauksessa useNewUrlParser ja useUnifiedTopology.

```
const mongoose = require('mongoose')

mongoose
    .connect(process.env.CONN_URI, {
        useNewUrlParser: true,
        useUnifiedTopology: true,
    })
    .catch((err) => {
        console.error('Connection error', err.message)
    })

const db = mongoose.connection

module.exports = db
```

Esimerkkikoodi 15. Tietokantayhteyden luominen dbconn.js-tiedostossa.

Koodiesimerkissä 16 yhdistetään tietokantaan, käsitellään mahdollinen virhe tietokantaan yhdistämisessä ja yhdistämisen onnistuessa tulee terminaaliin ilmoitus: ”Connected to database”.

```
const dbo = require('./db/dbconn')
dbo.on('error', console.error.bind(console, 'MongoDB connection er-
ror:'))
dbo.once('open', function () {
    console.log('Connected to database')
})
```

Esimerkkikoodi 16. Tietokantaan yhdistäminen server.js-tiedostossa.

4.2.2 Skeemat, mallit ja kokoelmat

Tietokantayhteyden muodostamisen jälkeen voidaan määrittää Contukissa skeemoja ja niitä vastaavia modeleita (malleja). Esimerkkikoodissa 17 määritetään tukitiimeille uusi Mongoosen malli.

```
const mongoose = require('mongoose')

const SupTeam = new mongoose.Schema(
    {
        title: String,
        support_group: String,
        service_provider: String,
        description: String,
    },
    { timestamps: true }
)

module.exports = mongoose.model('Supteam', SupTeam)
```

Esimerkkikoodi 17. Moduuli, jossa tukitiimin skeema ja malli luodaan.

Ensin määritetään vakioon SupTeam tukitiimin skeema, joka kertoo Mongoosselle, miten tukitiimioliot tulee tallentaa tietokantaan. Mallin määrittelyssä ensimmäinen parametri: ”SupTeam” määrittää tukitiimejä vastaavat oliot Mongoosessa kokoelmaan nimeltä ”supteams”, koska Mongoose luo automaattisesti skeeman nimen perusteella kokoelman, joka on skeeman nimi monikossa pienillä kirjaimilla. [48.]

Havainnollistetaan Contukissa olevien kokoelmien sisältöä taulukoiden avulla ja pyritään selventämään lukijalle kenttien tarkoitus. Taulukossa 1 esitellään ”Company”-model eli yrityksen mallin perusteella luotavaa ”companies”-kokoelma.

Taulukko 1. Companies-kokoelman sisältö.

Ominaisuus	**Tyyppi**	**Kuvaus**
company_name	merkkijono	Yrityksen nimi
description	merkkijono	Yrityksen kuvaus
phone_number	numero	Yrityksen puhelinnumero
email	merkkijono	Yrityksen sähköpostiosoite
address	merkkijono	Yrityksen käyntiosoite

additional_info	merkkijono	Mahdollisia lisätietoja yrityksestä
timestamps	päivämäärä	Kertoo päivämäärän ja kellonajan, milloin luotu ja milloin päivitetty.

Seuraavaksi esitellään ”Contact” model eli yhteystietojen mallin mukaan luotava ”contacts”-kokoelma taulukossa 2.

Taulukko 2. Contacts-kokoelman sisältö.

Ominaisuus	**Tyyppi**	**Kuvaus**
name	merkkijono	Yhteystiedon nimi
phone_number	numero	Yhteystiedon puhelinnumero
email	merkkijono	Yhteystiedon sähköpostiosoite
industry	merkkijono	Yhteystiedon toimiala
additional_info	merkkijono	Lisätietoja yhteystiedosta
timestamps	päivämäärä	Kertoo päivämäärän ja kellonajan, milloin luotu ja milloin päivitetty.

Viimeisenä kokoelmana on ”supteams”, eli tukitiimit, joka on luotu ”Supteams”-mallin perusteella. Tukitiimien kokoelma löytyy taulukosta 3.

Taulukko 3. Supteams-kokoelman sisältö.

Ominaisuus	**Tyyppi**	**Kuvaus**
title	merkkijono	Tukitiimien otsikko
support_group	merkkijono	Tukiryhmän nimi
service_provider	merkkijono	Palvelutarjooman nimi
description	merkkijono	Kuvaus

timestamps	päivämäärä	Päivämäärä milloin luotu ja päivitetty

Kokoelmien on tarkoitus kuvata Service Deskin ohjeissa olevien yhteystietojen, yritysten ja tukitiimien sisältöä. Kokoelmien avulla saadaan tieto järjestettyä tietokantaan ja voidaan dataa myöhemmin käsitellä käyttöliittymässäkin.

4.3 Reititys ja CRUD

Reititys viittaa siihen, kuinka ohjelmisto vastaa asiakkaan pyyntöihin tietyssä päätepisteessä, joka on URI (tai polku) ja kuinka se vastaa tiettyyn http-pyynnön menetelmään (GET, POST, jne.) Jokaisella reitillä voi olla yksi tai useampi käsittelijäfunktio, jotka ajetaan reitin aktivoituessa. [49.]

```
app.METHOD(PATH, HANDLER)
```

Esimerkkikoodi 18. Reitin määrityksen rakenne [49].

Esimerkkikoodissa 18 app on Expressin ilmentymä, METHOD on http-kutsun metodi, PATH on palvelimella oleva polku ja HANDLER on funktio, joka ajetaan, kun reitti aktivoituu. [49.]

CRUD-toiminnot ovat neljä olennaista datankäsittelytoimintoa, joissa käyttäjä voi luoda/lisätä, lukea/hakea, päivittää ja poistaa dataa. Nämä toiminnot liittyvät käyttöliittymäpuolen pyyntöihin ja toimintoihin, jotka suoritetaan taustajärjestelmässä asiakkaiden pyyntöjen käsittelemiseksi. Taulukossa 4 käydään läpi Contukissa käytettyjä toimintoja ja reittejä, joita tarvitaan käyttöliittymäpuolella lomakkeiden ja tietojen esittämisen kanssa.

Taulukko 4. Yhteystiedoilla käytetyt toiminnot ja reitit. Samat toiminnot ja reitit ovat käytössä myös tukitiimeillä ja yrityksillä.

Tehtävä	**Funktio Mongoosessa**	**http-pyyntö**	**Reitti**
Lisää/Luo	.save()	POST	/create

Hae ID:n perusteella	.findById()	GET	/findone/:id
Hae kaikki	.find()	GET	/findall
Päivitä ID:n perusteella	.findByIdAndUpdate()	PUT	/update/:id
Poista ID:n perusteella	.findByIdAndRemove()	DELETE	/delete/:id
Poista kaikki	.deleteMany()	DELETE	/deleteall

Esimerkkikoodissa 14 huomataan, että jokaiselle reitittimelle on määrätty oma polkunsa: ”/api/contact”, ”/api/company” ja ”api/supteams”. Tämä tarkoittaa sitä, että esimerkiksi yritykseen liittyvät CRUD-toiminnot tehdään seuraavalla tavalla: ”http://localhost:5000/api/company/+reitti”, eli tällöin lisää toiminnon polku olisi ”http://localhost:5000/api/company/create”.

Mikäli reittiin on määritetty reittiparametri eli nimetty url-osoite, kuten taulukossa 1 reitissä ”/findone/:id” käytetään parametria ”:id” sieppaamaan arvot, jotka on määritetty URL-osoitteen sijainnissa. Siepatut arvot täytetään req.params-olioon ja avaimena on reittiparametrin nimi, joka on määritetty polun jälkeen, tässä tapauksessa ”:id” (esimerkkikoodi 19). [50.]

```
const router = express.Router()
const SupTeam = require('./supteams_model')

router.get('/findone/:id', (req, res) => {
    const id = req.params.id

    SupTeam.findById(id)
        .then((data) => {
            if (!data)
                res.status(404).send({
                    message: "Couldn't find SupTeam with id: " + id,
                })
            else res.send(data)
        })
        .catch((err) => {
            res.status(500).send({
                message: 'Error retrieving SupTeam with id: ' + id,
            })
        })
})
```

Esimerkkikoodi 19. supteams_router.js-tiedostosta taulukon 1 tehtävä: ”Hae ID:n perusteella”.

Esimerkkikoodissa 19 on tallennettu Expressin olio Router() vakioon router ja tuotu tukitiimien Mongoosen malli, joka on tallennettu vakioon SupTeam. Määritetään uusi reitti ”/findone/:id/ ja tallennetaan reittiparametri ”:id” ”req.params.id” avulla vakioon id. Etsitään seuraavaksi Mongoosen mallista ”SupTeam” id:n perusteella dataa ja datan löytyessä palautetaan data komennolla ”res.send(data). Mikäli dataa ei kyseisellä id:llä löydy, annetaan virheilmoitus: ”couldn’t find SupTeam with id: +id”. Mikäli tapahtuu jokin muu virhe hakua tehdessä, niin tulee virheilmoitus: ”Error retrieving SupTeam with id: + id”. Tätä hakua käytetään, kun halutaan etsiä tietty tukitiimi ja tukitiimin id on tiedossa.

5 Käyttöliittymän kehitys

Käyttöliittymän kehityksen tarkoituksena on uudistaa olemassa olevaa Service Deskin yhteystietojen verkkosivua ja luoda alusta tukitiimeille, jotta tukitiimit löytyisivät yhdestä paikasta. Tällä hetkellä Service Deskillä osa tukitiimeistä on Excel-tiedostossa ja muut useammassa paikassa, jonka vuoksi prioriteettina on saada tukitiimit-sivu toimintakuntoon mahdollisimman nopeasti. Verrataan osiossa 5.4. hieman vanhaa sivustoa ja kehitettävää sivustoa. Lopuksi käydään

vielä läpi tavoitteena olleet kaksi näkymää, käyttäjän- ja järjestelmänvalvojan näkymät.

5.1 Reactin reititys

Reactin asennuskomento create-react-app ei sisällä sivujen reititystä, jonka vuoksi lisätään Contukiin React Router npm:n avulla. Reactin Routerin ideana on auttaa kehittämään yksisivuinen verkkosivu, jossa voidaan navigoida eri poluille nopeasti. Navigointi on nopeaa, koska yksisivuisella verkkosivulla ei tarvitse ladata verkkosivua siirtyessä päätepisteeltä toiselle.

Reitityksen perusrakenteen luomiseksi tuodaan react-router-domista BrowserRouter Routes ja Route. Perusrakenne muodostuu seuraavasti: ensin ympäröidään sisältö BrowserRouter:lla, jonka jälkeen Routes, joita voi olla ohjelmistossa useita. Viimeisenä syvyytenä on Route, joita voidaan sisentää. [51.]

Esimerkkikoodi 20 on kuvattuna muutama Contukin reitti, joiden avulla pyritään selventämään React Routerin toimintaa. Reitti ”/”, renderöi Layout-komponentin ja sisennetyt Routet perivät ylemmän tason reitin. Kun lisätään ylemmän tason reitti ja alemman tason reitti, niin saadaan komponentin Supteam reitiksi ”/supteams”. Mikäli reittejä olisi useampi sisäkkäin, voitaisiin saada reitiksi esimerkiksi ”/supteams/edit”.

```
import React from 'react'
import { Routes, Route } from 'react-router-dom'
import { Home, Layout ] from `./pages’
import NavBar from './components/Navbar/NavBar'

const App = () => {
    return (
        <div>
            <NavBar />
            <Routes>
                <Route path="/" element={<Layout />}
                    <Route index path="/" element={<Home />}
                    <Route exact path="/supteams" element={<Supteam
                />} />
                    <Route path="/supteams/:id"
                       element={<SupTeamTable />} />
                    <Route
                        path="*"
                        element={
                            <ErrorPage
                                text={'ERROR'}
                                description={'There is no such page'}
                            />
                        }
                    />
                </Route>
            </Routes>
        </div>
    )
}

export default App
```

Esimerkkikoodi 20. Esimerkki erilaisista reiteistä Contukissa. Huomio, että App.js on ympäröity react-router-dom:in komponentilla BrowserRouter index.js tiedostossa, jossa App.js renderöidään.

Yllä olevassa koodiesimerkissä 20 on erilaisia reittejä, joista ensimmäisessä on attribuuttina index, joka tarkoittaa, että se toimii vakioreittinä ylemmän tason reitille, tässä tapauksessa ”/”. Attribuutti exact tarkoittaa, että reitin pitää olla täsmälleen se, mikä on määritetty, eivätkä muut kelpaa, jotta komponentti voidaan renderöidä. Mikäli reitin edessä on pelkästään path, tarkoittaa se, että React Router palauttaa komponentin, vaikka reitti ei olisi täsmälleen sama kuin annettu polku. Esimerkiksi annettu reitti ”path=”/supteams”” palauttaa saman komponentin, vaikka polku olisi ”/supteams/fdsafds”. Kolmannessa reitissä on määritetty ”path=”*””, joka tarkoittaa, että se palauttaa tietyn sivun aina, kun Reactin reititin ei löydä annettua reittiä. Tämä reitti toimii hyvin virhesivustona. Reiteille voi myös antaa parametreja, esimerkiksi ”:id”. Aiheeseen voi perehtyä lisää osoitteessa: https://reactrouter.com/docs/en/v6.

5.2 Näkymät

Insinöörin yksi tavoitteista oli saada kehitettyä Contukiin kaksi näkymää: järjestelmänvalvojan näkymä ja käyttäjän näkymä. Työssä päätettiin aloittaa kehittäminen järjestelmänvalvojan näkymällä, eli tarkoituksena oli, että saadaan sekä tukitiimien, että yhteystietosivujen toiminnallisuudet tehtyä.

Järjestelmänvalvojan näkymässä on mahdollista lisätä, muokata, lukea ja poistaa sekä tukitiimien, että yhteystietojen tietoja. Tiedot tulevat tietokannasta, ja muutokset päivittyvät tietokantaan.

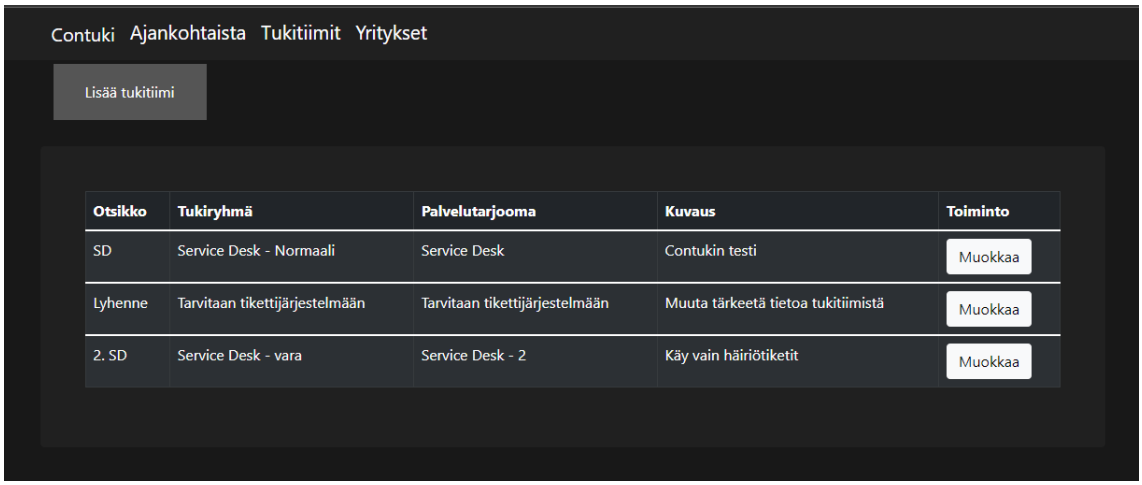
Käyttäjän näkymä jää jatkokehitettäväksi ja siinä on tarkoituksena poistaa mahdollisuus muokata sivuja.

Näkymien jatkokehitettäväksi jää järjestelmänvalvojan tunnistaminen eli kirjautuminen tai tunnistus yrityksen käyttäjätunnuksella ja painike, josta näkymää olisi mahdollista vaihtaa käyttäjän ja järjestelmänvalvojan välillä, mikäli tarvittavat oikeudet löytyvät käyttäjällä.

Esitetään seuraavissa luvuissa 5.3. ja 5.4. Contukia järjestelmänvalvojan näkymän näkökulmasta.

5.3 Tukitiimit

Tukitiimit ovat Service Deskin työtä tukevia tiimejä, joille voidaan ohjata tikettijärjestelmässä työpyyntöjä eli tikettejä, joihin tarvitaan lisäosaamista Service Desk -agenttien ulkopuolelta. Tukitiimeille ohjataan tiketit tukiryhmän ja palvelutarjooman perusteella. Tukiryhmä ja palvelutarjooma ovat merkkijonoja, eivätkä ne sisällä dataa. Service Deskin työntekijälle olennaista on, että tukiryhmä/palvelutarjooma-parit löytyvät yhdestä paikasta ja tiedetään mahdollisista lisätiedoista, onko kyseessä erikoistapaus, että tukitiimille käy ainoastaan häiriötiketit, eikä palvelupyyntöjä voi heille lähettää, tai toisinpäin.



Kuva 11. Contukissa oleva tukitiimien sivu Supteam.js ja esimerkkidataa.

Kuvista 11 ja 12 nähdään, että tukitiimisivu koostuu ”Lisää tukitiimi” -lomakkeesta, taulukosta ja taulukon sisällön muokkauslomakkeesta, joka aukeaa modaalina. Modaaleista löytyy lisää tietoa osoitteesta: https://react-bootstrap.github.io/components/modal/.

Luodaan ensin Supteam.js-tiedosto, johon rakennetaan tukitiimin sivu. Komponentin tarkoitus on näyttää tukitiimisivun sisältö ja huolehtia siitä, että tieto päivittyy, kun muutoksia tehdään. Haetaan tietokannasta tukitiimien dataa useEffect-koukun avulla ja tallennetaan ne useState-koukkuun. Tässä pitää olla tarkkana, koska useEffectin sisällä, kun tallennetaan toiseen koukkuun dataa, saadaan helposti loputon silmukka aikaiseksi. Havainnollisteaan tätä koodiesimerkissä 21.

```
const [supteams, updateSupteams] = useState([])
const [trigger, setTrigger] = useState(Boolean)

useEffect(() => {
    SupTeamsService.getAll()
        .then((res) => {
            console.log('res.data: ', res.data)
            updateSupteams([...res.data])
        })
        .catch((e) => {
            console.log('e: ', e)
        })
}, [trigger])

const addSupteam = (supteam) => {
    updateSupteams([...supteams, supteam])
    setTrigger(!trigger)
}
```

Esimerkkikoodi 21. Supteam.js:n koukkuja sekä addSupteam-funktio.

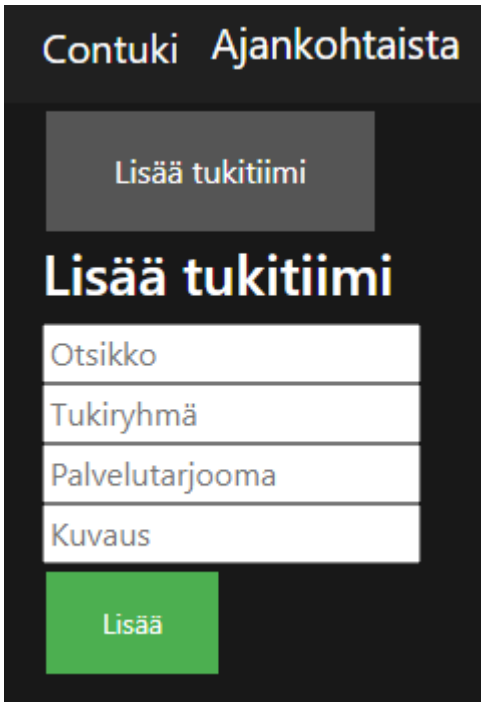
Tehtiin esimerkkikoodin 21 useEffect-koukku ja haettiin tietokannasta dataa funktiolla SupTeamsService.getAll(), jonka jälkeen talletettiin data useState-koukkuun funktiolla updateSupteams([…res.data]). Luotiin myös toinen useState-koukku [trigger, setTrigger], joka saa vain Boolean arvoja, eli 0 tai 1. Tämän koukun tarkoituksena on estää loputtoman silmukan syntyminen. Loputon silmukka syntyisi, jos useEffect-koukun viimeisellä rivillä olevien hakasulkujen sisälle ei laitettaisi useState-koukkua trigger: ”}, [trigger])”, vaan jätettäisiin sisältö tyhjäksi: ”}, [])”.

Taulukko luodaan SupTeamTable.jsx-komponentissa. Komponentin tehtävänä on luoda taulukko, toistaa ja tulostaa map-funktion avulla komponentin ominaisuutena saatu tietokannan data, jonka supteams-ominaisuuden pitää sisältää (esimerkkikoodi 22).

```
{supteams.map((supteam) => (
          <tbody key={supteam._id}>
          <tr>
          <td>{supteam.title}</td>
          <td>{supteam.support_group}</td>
          <td>{supteam.service_provider}</td>
          <td>{supteam.description}</td>
          <td>{<SupTeamEditModal id={supteam._id}
              deleteSupTeam={deleteSupTeam}
              editSupteam={editSupteam} />}</td>
          </tr>
          </tbody>
     ))}
```

Esimerkkikoodi 22. SupTeamTable.jsx-komponentin map-funktio toistaa annetun datan ja tulostetaan ne taulukossa. Esimerkissä näkyy myös, kuinka SupTeamEditModal saa id:n arvon ja ominaisuudet.

Kuvan 12 lomake luodaan komponentissa nimeltä SupTeamForm. Komponentti saa ominaisuutena funktion addSupteam ylemmän tason komponentilta Supteam.js.



Kuva 12. Lisää tukitiimi -painike on toggle-painike, jonka tilaa hallinnoidaan Reactin useState-koukulla.

Luodaan lomakkeet ja alustetaan useState koukun avulla lomakkeiden nimiä vastaavat arvot tyhjiksi, jotta saadaan tallennettua lomakkeiden tiedot näihin

arvoihin ja saadaan lähetettyä dataa tietokantaan (esimerkkikoodi 23). Luodaan kaksi funktiota: handleChange ja handleSubmit. Ensimmäinen funktio handleChange huolehtii, että saadaan lomakkeesta tiedot talteen ja handleSubmit:in tehtävänä on lähettää tieto uuden tukitiimin luonnista tietokantaan, kun käytetään funktiota SupTeamsService.create(), että ylemmän tason komponentille Supteam.js käyttäen funktiota addSupTeam(data).

```
const [supteamInfo, setSupTeamInfo] = useState({
    title: "",
    support_group: "",
    service_provider: "",
    description: "",
});

const handleChange = (event) => {
        setSupTeamInfo({ ...supteamInfo, [event.target.name]:
event.target.value });
    };

const handleSubmit = (event) => {
    event.preventDefault();
    addSupteam(supteamInfo);
    console.log(supteamInfo)
    SupTeamsService.create(supteamInfo).then(res => {
        console.log(res.data)
    }).catch(e => {
        console.log(e)
    })
    setSupTeamInfo({
        title: "",
        support_group: "",
        service_provider: "",
        description: ""
    });
};
```

Esimerkkikoodi 23. useState-koukku, handleChange- ja handleSubmit-funktiot SupTeamForm.jsx-tiedostossa.

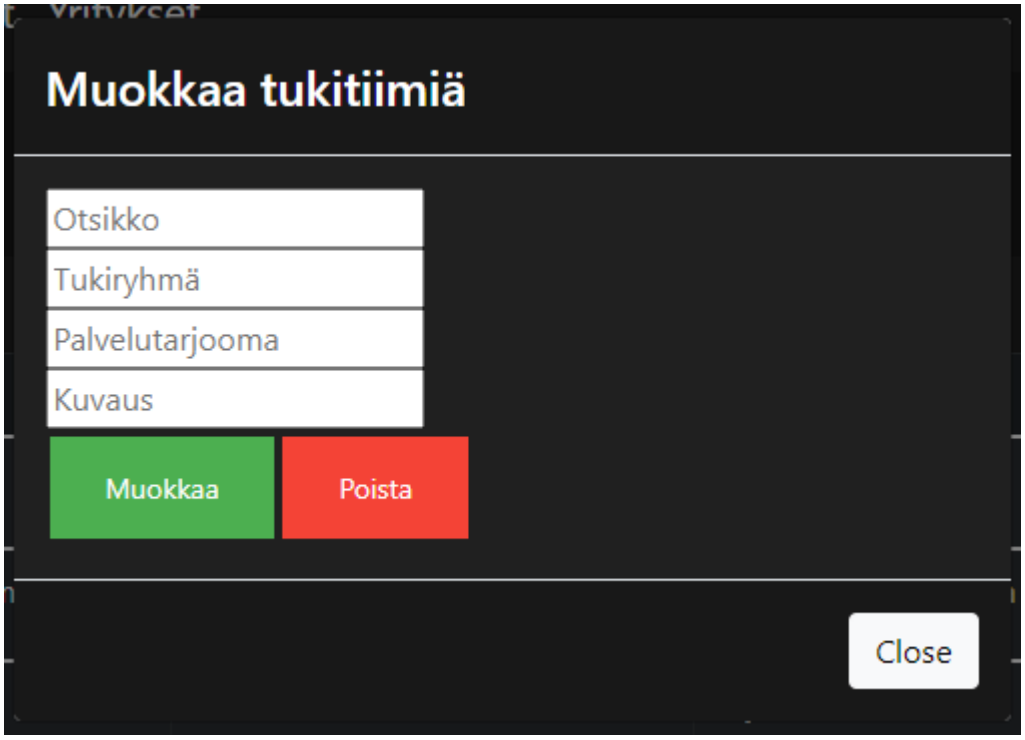
Lomakkeelle annetaan onSubmit arvoksi esimerkkikoodissa 23 handleSubmit-funktio. SupTeamsService:n funktiolla create(data) lähetetään lomakkeen data taustajärjestelmään.

Modaalin hallinta vaatii muutaman koukun lisää, joiden avuilla hallitaan, milloin modaali näkyy ja milloin se suljetaan. Modaali avataan kuvassa 11 näkyvällä Muokkaa-painikkeella ja Modaalin auetessa aukeaa kuvan 13 näköinen ponnahdusikkuna: ”Muokkaa tukitiimiä”.

```
const [show, setShow] = useState(false)
const handleClose = () => setShow(false)
const handleShow = () => setShow(true)
```

Esimerkkikoodi 24. Koukut, joiden avulla hallitaan modaalin näkyvyyttä.

Taulukon muokkaamista varten on tehty modaali kuvassa 12.



Kuva 13. Tukitiimien muokkaamista varten react-bootstrapin avulla tehty modaali SupTeamEditModal.jsx

Kuvassa 12. oleva modaali sisältää mahdollisuuden muokata tai poistaa taulukon sisällä olevaa dataa id:n perusteella. Muokkaamisen voi peruuttaa painamalla ylhäällä olevaa ruksia tai alhaalla olevaa Close-painiketta. Modaali saa Supteam.js:ltä ominaisuutena id, editSupteam ja deleteSupteam. Modaalin tietojen muokkaus tehdään vastaavalla tavalla kuin tukitiimin luonti. Funktio addSupteam vaihtuu editSupteam:ksi ja create(data) muuttuu funktioksi update(id, data).

5.4 Yhteystiedot

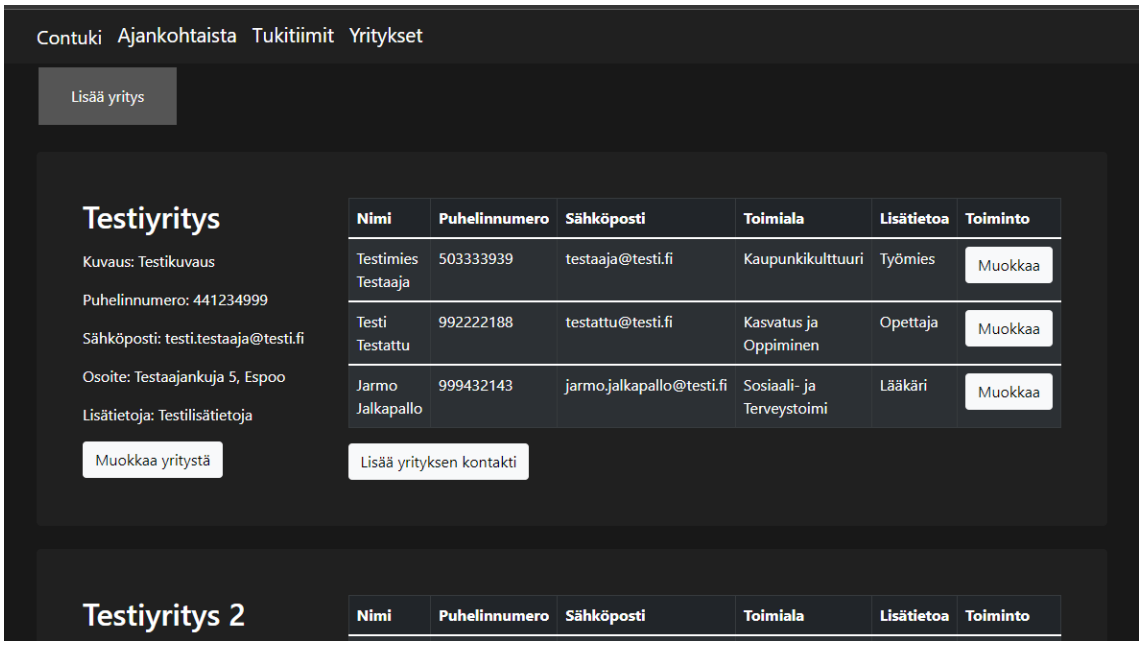
Yhteystietosivun tarkoitus on luoda alusta Service Deskin yritysyhteystiedoille. Service Deskin alkuperäinen yhteystietojen sivun rakenne on yksinkertainen. Sen rakenne on esimerkkikoodin 25 mukainen.

```
<html>
<head>
     sisältöä
</head>
<body>
     <table>
          sisältöä
     </table>
     <center>
          <table>
               sisältöä
          </table>
     </center>
</body>
```

Esimerkkikoodi 25. Service Deskin yhteystietosivuston rakenne, joka saatiin tarkastelemalla sivua kehittäjätyökalulla Internet Explorerilla.

Esimerkkikoodia 25 tarkasteltaessa huomataan, että sivusto koostuu kahdesta taulukosta. Nämä kaksi taulukkoa rakentavat rungon koko sivulle, ja sivuston koko sisältö on isojen taulukoiden solujen sisällä.

Contukissa yhteystietojen sivulla esitetään yritysten tiedot vasemmalla puolella ja yrityksiin kuuluvien yhteyshenkilöiden tiedot oikealla olevassa taulukossa kuvassa 14.



Kuva 14. Yhteystietojen sivu. Yksi yritys, yksi taulukko. Yritykset listataan allekkain.

Yhteystietosivulla Company.js voidaan luoda uusi yritys, jonka avulla luodaan uusi pohja, johon voidaan lisätä myös yrityksen kontakteja. Koodiesimerkissä 26 havainnollistetaan, kuinka yhteystietojen Company.js antaa CompanyList-lapsikomponentille ominaisuuksina yritysten ja yhteystietojen ominaisuudet.

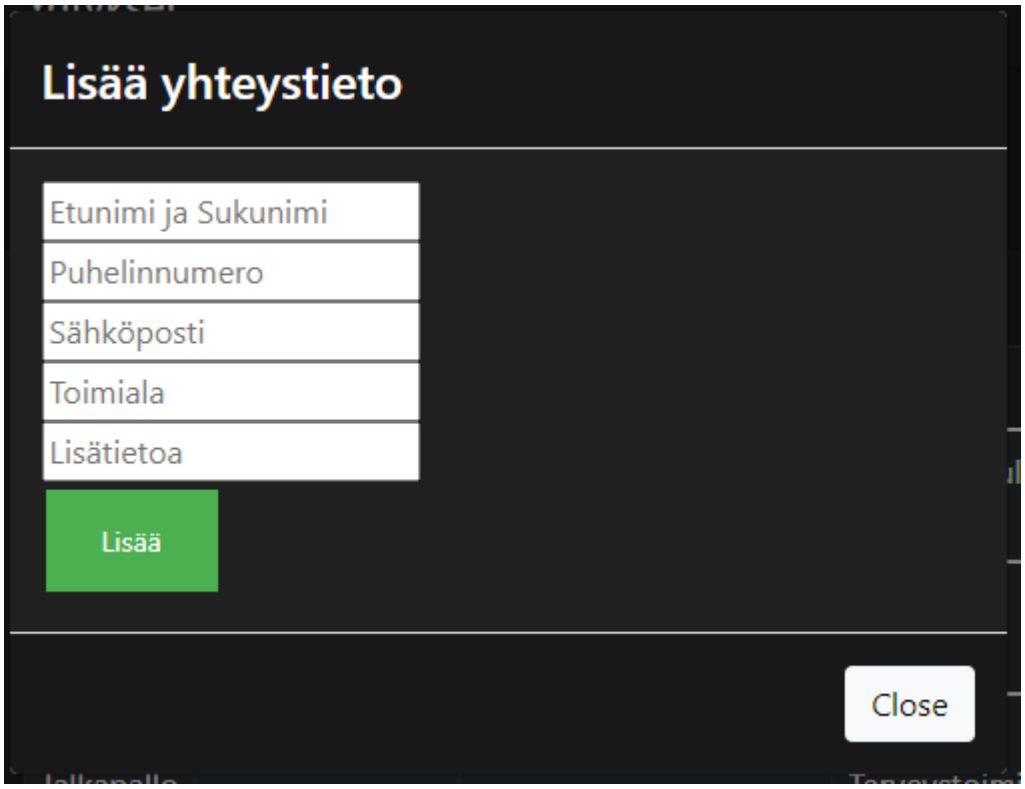
```
<div>
     <Container>
          <button class="toggleEdit" onClick={onClick}>
               Lisää yritys
           </button>
           {showForm ? <CompanyForm addCompany={addCompany} /> : null}
     </Container>
</div>
<div>
     <CompanyList
          companies={companies}
          contacts={contacts}
          addContact={addContact}
     />
</div>
```

Esimerkkikoodi 26. Company.js:ssä toggle-painike ja ominaisuuksien antaminen lapsikomponentille.

Painiketta ”Lisää yritys” painamalla aukeaa lomake kuten tukitiimeissäkin (kuva 15).

Kuva 15. Lisää yritys -lomake

Kuvassa 16 luodaan yritykselle uusi kontakti ja lisätään se taulukkoon kuvassa 14.



Kuva 16. Lisää yhteystieto -lomake

Yhteystietosivun jatkokehittämiseen jää hakutoiminnon ja suodattimen kehittäminen. Mahdollisesti hyödyllinen toiminto voisi olla suosikkiyhteystietojen merkkaaminen ja sivun reunaan tehty sisällysluettelo, jota painamalla voisi siirtyä oikean yrityksen tietoihin.

6 Yhteenveto

Insinöörityössä kehitettiin CGI:n Service Deskille sivustoa nimeltä Contuki. Sivuston tuli sisältää kaksi sivua, yksi tukitiimeille ja yksi yhteystiedoille. Sivustolla tuli myös luoda taustajärjestelmä, tietokantayhteys ja eheyttää käyttöliittymää. Sivustolle oli myös tarkoitus luoda kaksi näkymää, järjestelmänvalvojan näkymä ja käyttäjän näkymä.

Työssä kehitettiin vanhentuneelle verkkosivulle nykyaikaisilla teknologioilla kehitetty sivusto, jolla on taustajärjestelmä, tietokantayhteys ja käyttöliittymä. Taustajärjestelmän avulla voidaan tallentaa sivuston tiedot tietokantaan ja tietoja

voidaan nyt myös muokata ja poistaa. Tehtiin yksisivuinen käyttöliittymä, joka käyttää sivujen polkujen reitittämiseen React Router:ia. Saatiin tavoitteiden mukaisesti tehtyä tukitiimeille ja yhteystiedoille omat sivut.

Näkymistä saatiin vain järjestelmänvalvojan näkymä tehtyä, eikä toiminnallisuutta näkymien vaihtoon ole vielä toteutettu. Pyrittiin toteuttamaan kaikki perustavanlaatuiset toiminnallisuudet. Käyttäjän tunnistamista ei myöskään ole vielä toteutettu. Tunnistaminen tulisi tehdä siten, että CGI:n tunnuksilla tunnistettaisiin, kenellä on valtuudet sivuja muokata.

Mahdollisia jatkokehityskohteita sivustolle ovat hakutoiminto ja suodatin yhteystietojen sivulle ja sisällysluettelon kehittäminen, jotta ison tietomäärän selaaminen olisi helpompaa. Hyvä toiminto olisi myös, jos olisi mahdollista muuttaa, missä muodossa yhteystietojen tiedot esitetään, eli käyttäjällä olisi valta valita esitystapa esimerkiksi taulukkomuodossa tai listana.

Yksi insinöörityön haaste oli uusien teknologioiden opettelu ja tietotaidon kertyessä koodin jatkuva uudistaminen, joka vei aikaa uusilta ominaisuuksilta ja olemassa olevien ominaisuuksien jatkokehittämiseltä. Toisaalta kasvu insinöörityön aikana käytettyjen teknologioiden osaamisessa on merkittävä. Haasteellista oli myöskin jo olemassa olevan yhteystietosivuston ja uuden sivuston vertailun esittäminen ilman saatavilla olevia kuvia.

Lähteet

1 MongoDB & Mongoose: Compatibility and Comparison. Verkkoaineisto. https://www.mongodb.com/developer/article/mongoose-versus-nodejs-driver/ Luettu 28.4.2022.

2 What is Axios. Verkkoaineisto. https://axios-http.com/docs/intro Luettu 28.4.2022.

3 MERN Stack Explained. Verkkoaineisto. https://www.mongodb.com/mern-stack/. Luettu 28.4.2022.

4 Using Mongoose and MongoDB for the LocalLibrary. Verkkoaineisto. https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/mongoose Luettu 4.5.2022.

5 Web Frameworks & Introducing Express. Verkkoaineisto. https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction Luettu 4.5.2022.

6 What is Axios? Verkkoaineisto. https://axios-http.com/docs/intro Luettu 4.5.2022.

7 MERN stack Architecture. Verkkoaineisto. https://www.bezkoder.com/react-node-express-mongodb-mern-stack/ Luettu 28.4.2022.

8 What is React? Verkkoaineisto. https://medium.com/@mahmud.cse7/react-js-fundamentals-99e7eb54f57b Luettu 6.5.2022.

9 React - A JavaScript library for building user interfaces. Verkkoaineisto. https://reactjs.org/ Luettu 28.4.2022.

10 What You Should Already Know. Verkkoaineisto. https://www.w3schools.com/REACT/default.asp Luettu 5.5.2022.

11 What is the React.js framework? Verkkoaineisto. https://medium.com/@shakungrover01/why-we-use-react-js-in-website-development-process-7a4f49053813 Luettu 6.5.2022.

12 React's Tree Reconciliation. Verkkoaineisto. https://medium.com/@mahmud.cse7/react-js-fundamentals-99e7eb54f57b Luettu 6.5.2022.

13 What is the HTML DOM? Verkkoaineisto. https://www.w3schools.com/js/js_htmldom.asp Luettu 6.5.2022.

14 The HTML DOM Tree of Objects. Verkkoaineisto. https://www.w3schools.com/js/js_htmldom.asp Luettu 6.5.2022.

15 Virtual DOM and Internals. Verkkoaineisto. https://reactjs.org/docs/faq-internals.html Luettu 5.5.2022.

16 How Does React Work? Verkkoaineisto. https://www.w3schools.com/REACT/react_intro.asp Luettu 5.5.2022.

17 React: The Virtual DOM. Verkkoaineisto. https://www.codecademy.com/article/react-virtual-dom Luettu 5.5.2022.

18 Reusable components. Verkkoaineisto. https://medium.com/@shakungrover01/why-we-use-react-js-in-website-development-process-7a4f49053813 Luettu 6.5.2022.

19 JSX. Verkkoaineisto. https://medium.com/@mahmud.cse7/react-js-fundamentals-99e7eb54f57b Luettu 6.5.2022.

20 ReactJS State. Verkkoaineisto. https://medium.com/@mahmud.cse7/react-js-fundamentals-99e7eb54f57b Luettu 6.5.2022.

21 Introducing Hooks. Verkkoaineisto. https://reactjs.org/docs/hooks-intro.html Luettu 6.5.2022.

22 Hooks at Glance. Verkkoaineisto. https://reactjs.org/docs/hooks-overview.html Luettu 6.5.2022.

23 The V8 JavaScript Engine. Verkkoaineisto. https://nodejs.dev/learn/the-v8-javascript-engine Luettu 6.5.2022.

24 About Node.js. Verkkoaineisto. https://nodejs.org/en/about/ Luettu 29.4.2022.

25 Node.js Introduction. Verkkoaineisto. https://www.w3schools.com/nodejs/nodejs_intro.asp Luettu 29.4.2022.

26 Node.js Architecture. Verkkoaineisto. https://litslink.com/blog/node-js-architecture-from-a-to-z Luettu 6.5.2022.

27 What Can Node.js Do? Verkkoaineisto. https://www.w3schools.com/nodejs/nodejs_intro.asp Luettu 29.4.2022.

28 An introduction to the npm package manager. Verkkoaineisto. https://nodejs.dev/learn/an-introduction-to-the-npm-package-manager Luettu 6.5.2022.

29 nodemon. Verkkoaineisto. https://www.npmjs.com/package/nodemon Luettu 6.5.2022.

30 Software Package Manager. Verkkoaineisto. https://www.w3schools.com/whatis/whatis_npm.asp Luettu 6.5.2022.

31 What is Express? Verkkoaineisto. https://medium.com/@LindaVivah/the-beginners-guide-understanding-node-js-express-js-fundamentals-e15493462be1 4.5.2022.

32 Express. Verkkoaineisto. https://fullstackopen.com/osa3/node_js_ja_express Luettu 6.5.2022.

33 What does Express code look like? Verkkoaineisto. https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction Luettu 4.5.2022.

34 Web ja Express. Verkkoaineisto. https://fullstackopen.com/osa3/node_js_ja_express Luettu 6.5.2022.

35 Mongoose, Guides, Schemas. Verkkoaineisto. https://mongoosejs.com/docs/guide.html Luettu 4.5.2022.

36 Mongoose, Guides, Models. Verkkoaineisto. https://mongoosejs.com/docs/models.html Luettu 4.5.2022.

37 MongoDB & Mongoose: Compatibility and Comparison. Verkkoaineisto. https://www.mongodb.com/developer/article/mongoose-versus-nodejs-driver/ Luettu 28.4.2022.

38 React axios. Verkkoaineisto. https://www.javatpoint.com/react-axios Luettu 7.5.2022.

39 The Axios Instance. Verkkoaineisto. https://axios-http.com/docs/instance Luettu 5.5.2022.

40 Model-View-Controller(MVC) architecture for Node applications. Verkkoaineisto. https://www.geeksforgeeks.org/model-view-controllermvc-architecture-for-node-applications/ Luettu 1.5.2022.

41 Workflow and Folder Structure for MERN Application. Verkkoaineisto. https://codedec.com/tutorials/workflow-and-folder-structure-for-mern-application/ Luettu 1.5.2022.

42 Routing. Verkkoaineisto. https://expressjs.com/en/guide/routing.html Luettu 4.5.2022.

43 dotenv. Verkkoaineisto. https://www.npmjs.com/package/dotenv Luettu 3.5.2022.

44 cors. Verkkoaineisto. https://expressjs.com/en/resources/middleware/cors.html Luettu 7.5.2022.

45 Cross-Origin Resource Sharing. Verkkoaineisto. https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS Luettu 7.5.2022.

46 express.json([options]). Verkkoaineisto. http://expressjs.com/en/api.html#express.json Luettu 7.5.2022.

47 Mongoose, Guides, Connections, Options. Verkkoaineisto. https://mongoosejs.com/docs/connections.html#options Luettu 3.5.2022.

48 Compiling your first model. Verkkoaineisto. https://mongoosejs.com/docs/models.html Luettu 7.5.2022.

49 Basic Routing. Verkkoaineisto. http://expressjs.com/en/starter/basic-routing.html Luettu 5.5.2022.

50 Route parameters. Verkkoaineisto. https://expressjs.com/en/guide/routing.html Luettu 4.5.2022.

51 React Router. Verkkoaineisto. https://www.w3schools.com/react/react_router.asp Luettu 8.5.2022.