**FullStack-esimerkki (React, Node.js. MongoDB, Express ja Bootstrap)**

1. **Asennetaan omalle tietokoneelle MongoDB ja Robo 3T**
   1. **MongoDB Community**

Relaatiokannoissa rivit talletetaan tauluihin, MongoDB:ssä vastaavasti dokumentit talletetaan kokoelmiin. Dokumentti voi olla hierarkkinen, jolloin monen taulun asiat voidaan tallettaa yhteen dokumenttiin. Rakennemuoto on sama JSON, joka on vakiintunut ohjelmoinnissa ja rajapinnoissa. Helpottaa ohjelmointia kun ei tarvitse muuntaa olio-ohjelmoinnin rakenteita relaatiomuotoon vaan voi tallettaa samalla rakenteella.

Tietokannasta hakeminen on puolestaan monimutkaisempaa, MongoDB kun ei lähtökohtaisesti sisällä hauissa kätevää SQL-kieltä. Yhden hierarkkisen dokumentin haut ovat nopeita, mutta useita dokumentteja yhdistelevät haut voivat olla hankalia ja hitaita. Relaatiokantaihmisten rakastamia liitoksia ei ole, ne tehdään ohjelmoinnin puolella. Monista NoSQL-tuotteista poiketen SQL-rajapintoja on MongoDB-kantaan tarjolla niukasti. SQL on tärkeä kytkeytymisessä muuhun ympäristöön ja BI-tuotteisiin.

NoSQL-tietokannat voidaan jakaa tiedon organisointitapansa perusteella neljään eri luokkaan:

* avain-arvotietokantoihin (engl. key value databases), esim. Redis
* **dokumenttitietokantoihin** (engl. document databases), esim. **MongoDB**
* saraketietokantoihin (engl. columnar databases)
* verkkotietokantoihin (engl. graph databases)

**MongoDB lataus :** [https://www.mongodb.com/download-center/community (Valitse OS-kohdasta Windows)](https://www.mongodb.com/download-center/community%20(Valitse%20OS-kohdasta%20Windows))

|  |  |
| --- | --- |
| Valitaan .zip-version (Package-kohta). Pura zip-tiedosto omalle koneellesi ja uudelleennimeä kansio **mongodb**-nimellä.  Siirrä purettu kansio polkuun **C:\Users\omatunnus**. (omatunnus on käyttäjätunnuksesi).  Tee samaan paikkaan kansio **mongoData**. Viimeksi luotuun kansioon tallennetaan kaikki tietokannat. |  |

Kansiorakenne pitäisi tietokoneella olla

**C:**

**Users**

**omatunnus**

**mongodb**

**mongoData**

MongoDB käynnistetään Visual Studio Codessa **Terminal🡪 New Terminal-ikkunassa kirjoittamalla**:

**C:/Users/omatunnus/mongodb/bin/mongod.exe --dbpath /Users/omatunnus/mongoData**

MongoDB:n oletus datan tallennuspaikka on **C:\data\db**. Mikäli tätä kansiota ei ole, määrittele polku aina --**dbpath**:lla.

* 1. **Robo 3T**

Robo 3T-ohjelmisto, jonka avulla voidaan tarkastella ja hallita siihen yhdistetyn MongoDB-tietokannan sisältöä.

Lataa Robo 3T osoitteesta: <https://robomongo.org> josta ladataan 

Avaa Robo 3T-ohjelma ja valitse valikosta **File 🡪 Connect**

|  |  |
| --- | --- |
|  | 1. Valitse vasemmasta yläkulmasta **Create** jonka jälkeen avautuu uusi ikkunassa, jossa voit antaa yhteydelle nimi. 2. Tarkista, että portti on localhost:27017 3. Testaa yhteys vasemman alakulman Test-painikkeella 4. Tallenna **Save**-painikkeella 5. Nyt voit yhdistää Robo 3T:n klikkaamalla **Connect**-painiketta (yhteyden oltava päällä, jotta toimii). Seuraavilla kerroilla voit klikata suoraan **Connect**. |
|  | Kun yhteys on onnistunut, näkymä on kuvan mukainen. |
|  | Klikkaamalla hiiren oikean puoleista painiketta yhteyden päällä (tässä **New Connection(4)**, saat näkyville **Server Status-valinnalla** mm. polun, josta mondoDB toimii. |

**HUOM!** Jos jatkossa on ongelma, että Robo3T:ssä ei käynnisty/ei toimi yhteys mongod, niin käy Visual Studiossa Terminalissa käynnistämässä mongo, esim. **(C:/Users/omatunnus/mongodb/bin/mongod.exe --dbpath /Users/omatunnus/mongoData)**

**Vinkki!**

Jos haet GitHubista jonkin valmiin projekti-paketin (.zip), ota projekti seuraavalla tavalla käyttöösi:

1. Jos et ole asentanut Node.js:ää vielä koneellesi 🡪 asenna NodeJS ja NPM <https://nodejs.org/en/download/>.
2. Lataa haluamasi lähdekoodi GitHubista, esim. <https://github.com/cornflourblue/react-jwt-authentication-example>
3. Asenna kaikki Visual Studiossa npm-paketit projektisi juuressa käskyllä **npm install**  (eli siellä, missä *package.json* -tiedosto löytyy).
4. Käynnistä sovellus Terminal-ikkunassa projektisi juuressa käskyllä **npm start**
5. Selaimesi avautuu automaattisesti esim. http://localhost:8080 tai http://localhost:3000
6. **Esimerkki sovelluksen rakentaminen (katso oppaan viimeinen sivu, jossa kuvaus rakenteesta)**

Create React App asennetaan ajamalla komentokehotteessa komento ”**npm install –g create-react-app**”. React-sovellus luodaan Create React App -paketin komennolla ”create-react-app <sovelluksen nimi>”. Komennossa ilmoitetaan sovelluksen nimi, jolla sovellus sen hetkiseen kansioon generoidaan. Sovelluksen kansiossa on **package.json**-tiedosto, mihin tallennetaan esimerkiksi sovelluksessa käytettävien pakettien versionumerot, sovelluksen yleisiä tietoja sekä kehittäjän määrittelemiä komentoja. Create React App asettaa valmiiksi sovelluksen kehitystä varten tarvittavat paketit ja komennot, joiden avulla sovelluksen kehityspalvelin voidaan käynnistää sekä luoda lopullinen optimoitu sovellus. (Facebook Inc. 2019c.)

React-sovelluksen kehitys tapahtuu suurimmaksi osaksi Create React App paketin generoiman **src-kansion** sisällä. Sovelluksen käynnistystiedostona toimii kansiossa oleva **index.js**-tiedosto. Kansiossa on myös valmiiksi generoitu **App-komponentti**, jonka sisään yleensä asetetaan kaikki React-sovelluksen komponentit. **App-komponentti on pääkomponentt**i.

* 1. **Asennetaan React**

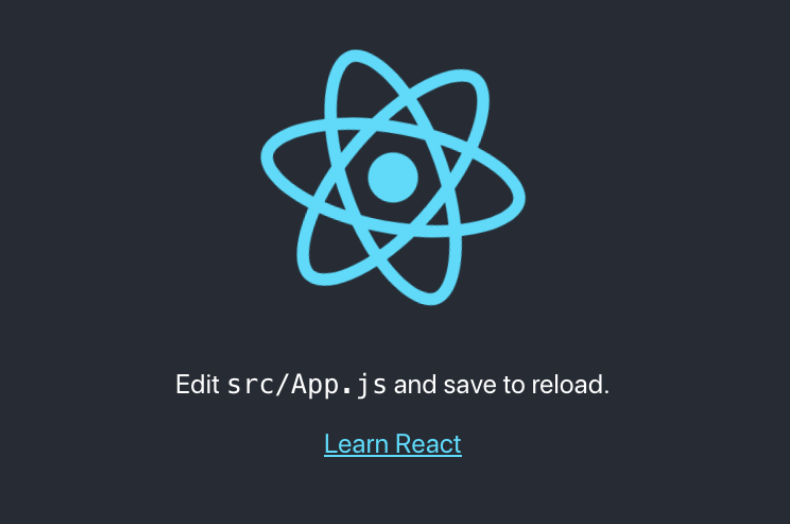
Avaa Visual Studio Code-ohjelma ja valitse valikosta **Terminal 🡪 New Terminal**

npx create-react-app reactcrud

cd reactcrud

npm start

Selaimeen avautuu URL: <http://localhosta:3000/>



* 1. **Asennetaan Bootstrap-framework seuraavalla komennolla**

npm install bootstrap –save

# or

yarn add bootstrap

Liitetään (import) **Bootstrap** CSS framework projektiisi

Muokkaa (copy🡪paste) Visual Studion vasemman puolen ikkunassa projektisi **src**-hakemiston **App.js**-tiedostoa allaolevaksi ja tallenna **File 🡪 Save**

// App.js

import React, { Component } from 'react';

**import 'bootstrap/dist/css/bootstrap.min.css';**

class App extends Component {

render() {

return (

<div className="container">

<h2>Tietokanta Tutorial</h2>

</div>

);

}

}

export default App;

* 1. **Konfikuroidaan React reititys (routing)**

Installoidaan seuraavalla komennolla **react- router-dom** -moduuli. Lisätietoja: <https://reacttraining.com/react-router/web/guides/quick-start>

npm install react-router-dom --save

# tai

yarn add react-router-dom

Mene **index.js** tiedostoon ja lisää **BrowserRouter** -objekti

// index.js

import React from 'react';

import ReactDOM from 'react-dom';

import { BrowserRouter } from 'react-router-dom';

import App from './App';

import \* as serviceWorker from './serviceWorker';

// BrowserRouter-komponentin tulee sisältää kaikki sovelluksen

// komponentit, missä hyödynnetään React Router DOM-paketin

// toiminnallisuutta. Täten BrowserRouter-komponentti on

// suoraan ensimmäinen alikomponentti sovelluksen

// juurikomponentille(App-komponentti)ja BrowserRouter-komponentin

// sisälle rakennetaan koko sovelluksen komponenttipuu.

ReactDOM.render(

<BrowserRouter>

<App />

</BrowserRouter>, document.getElementById('root'));

serviceWorker.unregister();

Seuraavaksi luodaan **src**-kansion sisälle **components**-kansio, jonka sisälle luodaan kolme komponenttia.

1. **luo.component.js**
2. **muokkaa.component.js**
3. **listaa.component.js**

// luo.component.js

import React, { Component } from 'react';

export default class Luo extends Component {

render() {

return (

<div>

<p>Tervetuloa luo/tallenna komponenttiin!!</p>

</div>

)

}

}

// muokkaa.component.js

import React, { Component } from 'react';

export default class Muokkaa extends Component {

render() {

return (

<div>

<p>Tervetuloa muokkaus komponenttiin!!</p>

</div>

)

}

}

// listaa.component.js

import React, { Component } from 'react';

export default class Listaa extends Component {

render() {

return (

<div>

<p>Tervetuloa listaa komponenttiin!!</p>

</div>

)

}

}

Kirjoita seuraava koodi **App.js** -tiedostoon

// App.js

import React, { Component } from 'react';

import 'bootstrap/dist/css/bootstrap.min.css';

import { BrowserRouter as Router, Switch, Route, Link } from 'react-router-dom';

import Luo from './components/luo.component';

import Muokkaa from './components/muokkaa.component';

import Listaa from './components/listaa.component';

class App extends Component {

render() {

return (

<Router>

<div className="container">

<nav className="navbar navbar-expand-lg navbar-light bg-light">

<Link to={'/'} className="navbar-brand">Tietokanta esimerkki</Link>

<div className="collapse navbar-collapse" id="navbarSupportedContent">

<ul className="navbar-nav mr-auto">

<li className="nav-item">

<Link to={'/'} className="nav-link">Koti</Link>

</li>

<li className="nav-item">

<Link to={'/create'} className="nav-link">Lomake</Link>

</li>

<li className="nav-item">

<Link to={'/index'} className="nav-link">Lista</Link>

</li>

</ul>

</div>

</nav> <br/>

<h2>Tervetuloa Tietokantaesimerkkiin</h2> <br/>

<Switch>

<Route exact path='/create' component={ Luo } />

<Route path='/edit/:id' component={ Muokkaa } />

<Route path='/index' component={ Listaa } />

</Switch>

</div>

</Router>

);

}

}

export default App;

Selaimessa pitäisi näkyä nyt valikot



* 1. **Luodaan tietojen lisäys lomake (Bootstrap-tyylit)**

Kirjoita alla oleva koodi **luo.component.js** -tiedostoon

// luo.component.js

import React, { Component } from 'react';

export default class Luo extends Component {

render() {

return (

<div style={{marginTop: 10}}>

<h3>Lisää tietoa tietokantaan</h3>

<form>

<div className="form-group">

<label>Henkilön nimi: </label>

<input type="text" className="form-control"/>

</div>

<div className="form-group">

<label>Yritys: </label>

<input type="text" className="form-control"/>

</div>

<div className="form-group">

<label>Numero: </label>

<input type="text" className="form-control"/>

</div>

<div className="form-group">

<input type="submit" value="Tallenna tiedot" className="btn btn-primary"/>

</div>

</form>

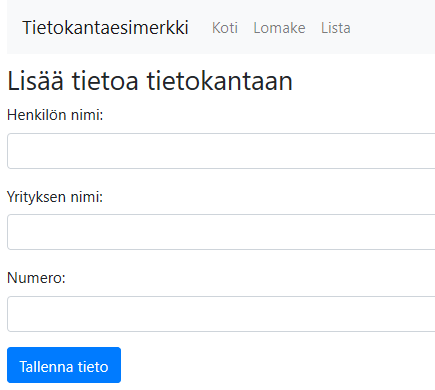
</div>

)

}

}

Selaimessa pitäisi näkyä



* 1. **Lomakkeen lähettäminen**

1. henkilön nimi
2. yritys
3. numero

Meidän on luotava kolmekentää. Myös neljäs toiminto lähettää POST-pyynnön palvelimelle.

Nyt ensin määrittelemme konstruktorin ja asetamme sen alkutilan ja yhdistämme tämän myös eri tapahtumiin.

Määritä sitten eri toiminnot kullekin syöte tekstiarvolle. Joten kun käyttäjä kirjoittaa tekstikenttään, asetamme tilan sen mukaan.

Joten sanotaan, että käyttäjä kirjoittaa henkilön nimen tekstikenttään, muutamme henkilön nimen tila-arvoa. Lopuksi, sama kaikille syöteille ja kun lähetämme lomakkeen, saamme arvot tilasta ja lähetämme **POST**-pyyntöön.

// App.js

import React, { Component } from 'react';

export default class Luo extends Component {

constructor(props) {

super(props);

this.onChangePersonName = this.onChangePersonName.bind(this);

this.onChangeBusinessName = this.onChangeBusinessName.bind(this);

this.onChangeGstNumber = this.onChangeGstNumber.bind(this);

this.onSubmit = this.onSubmit.bind(this);

this.state = {

person\_name: '',

business\_name: '',

business\_gst\_number:''

}

}

onChangePersonName(e) {

this.setState({

person\_name: e.target.value

});

}

onChangeBusinessName(e) {

this.setState({

business\_name: e.target.value

})

}

onChangeGstNumber(e) {

this.setState({

business\_gst\_number: e.target.value

})

}

onSubmit(e) {

e.preventDefault();

console.log(`The values are ${this.state.person\_name}, ${this.state.business\_name}, and ${this.state.business\_gst\_number}`)

this.setState({

person\_name: '',

business\_name: '',

business\_gst\_number: ''

})

}

render() {

return (

<div style={{ marginTop: 10 }}>

<h3>Lisää tiedot</h3>

<form onSubmit={this.onSubmit}>

<div className="form-group">

<label>Henkilön nimi: </label>

<input

type="text"

className="form-control"

value={this.state.person\_name}

onChange={this.onChangePersonName}

/>

</div>

<div className="form-group">

<label>Yritys: </label>

<input type="text"

className="form-control"

value={this.state.business\_name}

onChange={this.onChangeBusinessName}

/>

</div>

<div className="form-group">

<label>Numero: </label>

<input type="text"

className="form-control"

value={this.state.business\_gst\_number}

onChange={this.onChangeGstNumber}

/>

</div>

<div className="form-group">

<input type="submit" value="Tallenna tiedot" className="btn btn-primary"/>

</div>

</form>

</div>

)

}

}

* 1. **Luo backend Node.js:llä**

**Luo ensin projekti juureen kansio api** ja mene terminal-ikkunassa sen sisälle ja asennetaan **package.json** -tiedosto. Lisätietoa tietojen tallentamisesta MongoDB-tietokantaan: <https://fullstack-hy2019.github.io/osa3/tietojen_tallettaminen_mongo_db_tietokantaan>

**npm init -y**

installoi seuraavaksi **node.js**-tiedosto **dependencies**-tietoihin

npm install express body-parser cors mongoose --save

tai

yarn add express body-parser cors mongoose

Asennetaan myös **nodemon**, niin ei tarvitse jatkuvasti käynnitellä koodia, kun teemme muutoksia serverin koodiin.

npm install nodemon --save-dev

Mene seuraavaksi **api**-kansioon ja luo tiedosto **server.js** ja lisää all aoleva koodi sen sisälle.

// server.js

const express = require('express');

const app = express();

const bodyParser = require('body-parser');

const PORT = 4000;

const cors = require('cors');

app.use(cors());

app.use(bodyParser.urlencoded({extended: true}));

app.use(bodyParser.json());

app.listen(PORT, function(){

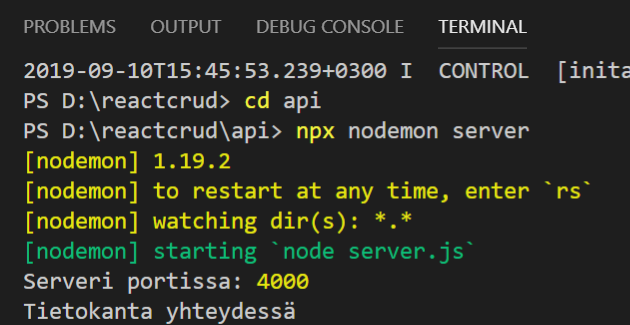
console.log('Serveri portissa:',PORT);

});

Tallenna tiedosto ja avaa uusi **Terminal**-ikkuna **api**-kansioon ja anna seuraava komento ja kirjaudu **node.js** serveriin

npx nodemon server

Näet **Terminal**-ikkunassa, että **node.js** serveri on käynnissä



* 1. **Asennetaan MongoDB tietokanta**

Lue SQL vai NoSQL: <https://petrikonkka.com/fi/pilvipalvelujen-tietokannat-nosql-mongodb>

Voit lukea lisätietoa: <https://appdividend.com/2018/03/31/nosql-mongodb-tutorial-example/>

Jos olet jo asentanut MongoDB:n, niin käynnistä se Terminal-ikkunassa komennolla

mongoDB

tai

C:\Users\käyttäjätunuksesi\mongodb\bin/mongod.exe --dbpath C:\Users\käyttäjätunnuksesi\mongoData

Menen **api**-kansion sisään ja luo **DB.js**-niminen tiedosto ja kopio alla oleva koodi sinne. Tässä luodaan paikallinen MongoDB-tietokanta. käyttäjätunnusta ja salasanaa ei tässä ole.

// DB.js

module.exports = {

DB: 'mongodb://localhost:27017/reactcrud'

}

Muokkaa nyt **server.js**-tiedoston sisältö alla olevan mukaan

// server.js

const express = require('express');

const app = express();

const bodyParser = require('body-parser');

const PORT = 4000;

const cors = require('cors');

const mongoose = require('mongoose');

const config = require('./DB.js');

mongoose.Promise = global.Promise;

mongoose.connect(config.DB, { useNewUrlParser: true }).then(

() => {console.log('Tietokantaan yhdistetty') },

err => { console.log('Ei yhteyttä tietokantaan'+ err)}

);

app.use(cors());

app.use(bodyParser.urlencoded({extended: true}));

app.use(bodyParser.json());

app.listen(PORT, function(){

console.log('Serveri portissa:',PORT);

});

Tallenna tiedosta ja nyt sinulla on yhteys MongoDB-tietokantaan Node.js:ssä.

* 1. **Luodaan Mongoose scheema**

Luodaan MongoDB-tietokantaan scheema. Luo **api**-kansion sisälle **business.model.js** tiedosto. Tässä määritellään kolme kenttää: (VINKKI: Hae MongoDB-visualisointi työkaluja: esim. <https://mlab.com/lp/mongodb-gui>

* person\_name
* business\_name
* business\_gst\_number

sekä näiden datatyypit.

// business.model.js

const mongoose = require('mongoose');

const Schema = mongoose.Schema;

// Define collection and schema for Business

let Business = new Schema({

person\_name: {

type: String

},

business\_name: {

type: String

},

business\_gst\_number: {

type: Number

}

},{

collection: 'business'

});

module.exports = mongoose.model('Business', Business);

* 1. **Määritetään reitti Node.js Express-sovelluksella**

Luo **api**-kansioon tiedosto **business.route.js**

// business.route.js

const express = require('express');

const businessRoutes = express.Router();

// hakee Business mallin routes modulilista

let Business = require('./business.model');

// Määritellään store route

businessRoutes.route('/add').post(function (req, res) {

  let business = new Business(req.body);

  business.save()

    .then(business => {

      res.status(200).json({'business': 'business lisätty onnistuneesti'});

    })

    .catch(err => {

    res.status(400).send("mahdoton tallentaa tietokantaan");

    });

});

// Määritelty get data(index tai listaus) route

businessRoutes.route('/').get(function (req, res) {

    Business.find(function(err, businesses){

    if(err){

      console.log(err);

    }

    else {

      res.json(businesses);

    }

  });

});

// Määritelty edit route

businessRoutes.route('/edit/:id').get(function (req, res) {

  let id = req.params.id;

  Business.findById(id, function (err, business){

      res.json(business);

  });

});

//  Määritelty update route

businessRoutes.route('/update/:id').post(function (req, res) {

    Business.findById(req.params.id, function(err, business) {

    if (!business)

      res.status(404).send("tietoa ei löydy");

    else {

        business.person\_name = req.body.person\_name;

        business.business\_name = req.body.business\_name;

        business.business\_gst\_number = req.body.business\_gst\_number;

        business.save().then(business => {

          res.json('Päivitys onnistunut');

      })

      .catch(err => {

            res.status(400).send("mahdoton päivittää tietokantaan");

      });

    }

  });

});

// Määritelty delete | remove | destroy route

businessRoutes.route('/delete/:id').get(function (req, res) {

    Business.findByIdAndRemove({\_id: req.params.id}, function(err, business){

        if(err) res.json(err);

        else res.json('Poistaminen onnistunut');

    });

});

module.exports = businessRoutes;

* 1. **Asennetaan Axios-kirjasto ja tehdään POST**

**Lisätietoa:** [**https://appdividend.com/2018/08/30/getting-started-with-axios-tutorial-example/**](https://appdividend.com/2018/08/30/getting-started-with-axios-tutorial-example/)

**2.11 Axios**

**Axios on HTTP-asiakaskirjasto**. Kun selain haluaa tehdä *asynkronisia HTTP-kutsuja*, se käyttää **XmlHttpRequest**-objektia(joka nimestään huolimatta ei ole sidottu XML-muotoiseen dataan). **Axios** huolehtii **XmlHttpRequest**-objektin käsittelystä ja tarjoaa ohjelmoijalle mukavamman rajapinnan HTTP-kutsujen tekemiseen ja vastausten käsittelyyn.

**Axios**-kutsu palauttaa **Promise**-objektin, joka kuvaa asynkronista operaatiota. Sen onnistunut tulos saadaan käsiteltyä objektin **then**-metodissa, jonka parametrina on vastauksen tuottama data. Virhetilanteet saadaan käsiteltyä **catch**-metodissa, jonka perusteella voidaan näyttää käyttöliittymässä virheilmoitus tai muuten reagoida virheeseen

npm install axios --save

# tai

yarn add axios

Nyt voimme lähettää **HTTP POST** kyselyjä lomakkeelta **node.js** serverille.

Lähetämme tiedot objektina, koska olemme käyttäneet taustana olevaa **body-parser**ia keräämään tietoja pyynnöstä ja tallentamaan ne tietokantaan.

Kirjoita seuraavana **luo.component.js** -tiedosto komponentti-kansioon

// luo.component.js - Tällä sivulla luodaan tallennuslomake ja tallennetaan se tietokantaan.

// Tähän tarvitaan axiosia

import React, { Component } from 'react';

import axios from 'axios';

export default class Luo extends Component {

  constructor(props) {

    super(props);

    this.onChangePersonName = this.onChangePersonName.bind(this); //bind-metodi

    this.onChangeBusinessName = this.onChangeBusinessName.bind(this);

    this.onChangeGstNumber = this.onChangeGstNumber.bind(this);

    this.onSubmit = this.onSubmit.bind(this);

    this.state = {

      person\_name: '',

      business\_name: '',

      business\_gst\_number:''

    }

  }

  onChangePersonName(e) {

    this.setState({

      person\_name: e.target.value

    });

  }

  onChangeBusinessName(e) {

    this.setState({

      business\_name: e.target.value

    })

  }

  onChangeGstNumber(e) {

    this.setState({

      business\_gst\_number: e.target.value

    })

  }

// lomakkeen lähettäminen onSubmit:lla palvelimelle

  onSubmit(e) {

    e.preventDefault();

    const obj = {

      person\_name: this.state.person\_name,

      business\_name: this.state.business\_name,

      business\_gst\_number: this.state.business\_gst\_number

    };

    axios.post('http://localhost:4000/business/add', obj)

        .then(res => console.log(res.data));

    this.setState({

      person\_name: '',

      business\_name: '',

      business\_gst\_number: ''

    })

  }

 // Tietoen lisäämislomake

  render() {

    return (

        <div style={{ marginTop: 10 }}>

            <h3>Lisää tietoa tietokantaan</h3>

            <form onSubmit={this.onSubmit}>

                <div className="form-group">

                    <label>Henkilön nimi:  </label>

                    <input

                      type="text"

                      className="form-control"

                      value={this.state.person\_name}

                      onChange={this.onChangePersonName}

                      />

                </div>

                <div className="form-group">

                    <label>Yrityksen nimi: </label>

                    <input type="text"

                      className="form-control"

                      value={this.state.business\_name}

                      onChange={this.onChangeBusinessName}

                      />

                </div>

                <div className="form-group">

                    <label>Numero: </label>

                    <input type="text"

                      className="form-control"

                      value={this.state.business\_gst\_number}

                      onChange={this.onChangeGstNumber}

                      />

                </div>

                <div className="form-group">

                    <input type="submit" value="Tallenna tieto" className="btn btn-primary"/>

                </div>

            </form>

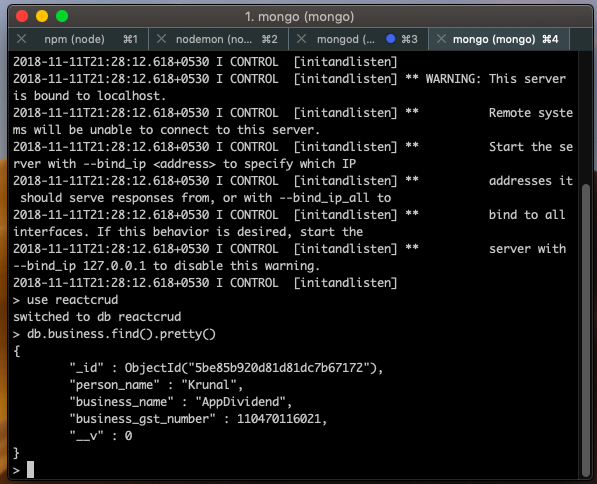
        </div>

    )

  }

}

Käynnistä nyt **mongodb**-tietokanta ja katso meneekö tiedot kantaan.



* 1. **Kuvaruudulle listataan backend data**

Luo tiedosto **listaa.component.js** tiedosto komponentit kansioon ja sen sisälle seuraava koodi.

// listaa.component.js - Saadaan tieto kannasta axiosin avulla selaimeen

// Selaimen ja palvelimen väliseen kommunikaatioon axios-kirjasto

// Reaction vakiofunktiot:

// componentDidUpdate() - suoritetaan renderoinnin jälkeen

// componentDidMount() - suoritaan kerran heti ensimmäisen renderoinnin jälkeen

import React, { Component } from 'react';

import axios from 'axios';

import ListanRivi from './Taulurivit';

export default class Listaa extends Component {

  // this.statessa määritellään komponentin tila

  // Konstruktori asettaa nyt propseina saatavan business-listaan tiedot:

  constructor(props) {

      super(props);

      this.state = {business: []};

    }

    componentDidUpdate() {

      axios.get('http://localhost:4000/business')

      .then(response => {

      this.setState({ business: response.data });

      })

      .catch(function (error) {

      console.log(error);

      })

      }

// Tietojen haku palvelimelta on elinkaari metodi componentDidMount (luokkapainotteinen komponentti),

// joka suoritetaan kertaalleen heti komponentin ensimmäisen

// renderöitymisen jälkeen.

// HTTP-pyynnön takaisinkutsufunktio päivittää komponentin tilaa

// metodilla setState. Metodi toimii siten, että

// se koskee tilassa ainoastaan niihin avaimiin, mitä parametrina

// olevassa oliossa on määritelty.

// Metodin setState kutsuminen aiheuttaa aina luokkakomponentin

// uudelleenrenderöinnin, eli metodin render kutsun.

    componentDidMount(){

      axios.get('http://localhost:4000/business')

        .then(response => {

          this.setState({ business: response.data });

        })

        .catch(function (error) {

          console.log(error);

        })

    }

    tabRow(){

      return this.state.business.map(function(object, i){

          return <ListanRivi obj={object} key={i} />;

      });

    }

    render() {

      return (

        <div>

          <h3 align="center">Henkilölista</h3>

          <table className="table table-striped" style={{ marginTop: 20 }}>

            <thead>

              <tr>

                <th>Henkilö</th>

                <th>Yritys</th>

                <th>Numero</th>

                <th colSpan="2">Toiminnot</th>

              </tr>

            </thead>

            <tbody>

              { this.tabRow() }

            </tbody>

          </table>

        </div>

      );

    }

  }

Tässä läheteään **get**-kyselyjä **node.js** serverille ja noutaa nämä **API**:sta. Tähän on importoitu **TauluRivit.js**-komponetti, joka täytyy olla **konponentit**-kansiossa.

// TauluRivit.js

import React, { Component } from 'react';

import { Link } from 'react-router-dom';

import axios from 'axios';

class ListanRivi extends Component {

  constructor(props) {

        super(props);

        this.delete = this.delete.bind(this);

    }

    delete() {

        axios.get('http://localhost:4000/business/delete/'+this.props.obj.\_id)

            .then(console.log('Poistettu'))

            .catch(err => console.log(err))

    }

  render() {

    return (

        <tr>

          <td>

            {this.props.obj.person\_name}

          </td>

          <td>

            {this.props.obj.business\_name}

          </td>

          <td>

            {this.props.obj.business\_gst\_number}

          </td>

          <td>

            <Link to={"/edit/"+this.props.obj.\_id} className="btn btn-primary">Muokkaa</Link>

          </td>

          <td>

            <button onClick={this.delete} className="btn btn-danger">Poista</button>

          </td>

        </tr>

    );

  }

}

export default ListanRivi;

Tämä komponentti noutaa rivitiedot listaan. Tallenna tiedosto ja katso selaimessa URL: <http://localhost:3000/index>**.**



* 1. **Muokkaa ja päivitä toiminnallisuus**

Lisää komponentit-kansioon **TauluRivit.js**-tiedosto

// TauluRivit.js

import { Link } from 'react-router-dom';

<Link to={"/edit/"+this.props.obj.\_id} className="btn btn-primary">Muokkaa</Link>

Lisää **komponentit**-kansioon **muokkaa.components.js**-tiedosto

// muokkaa.component.js

import React, { Component } from 'react';

import axios from 'axios';

export default class Muokkaa extends Component {

  constructor(props) {

    super(props);

    this.onChangePersonName = this.onChangePersonName.bind(this);

    this.onChangeBusinessName = this.onChangeBusinessName.bind(this);

    this.onChangeGstNumber = this.onChangeGstNumber.bind(this);

    this.onSubmit = this.onSubmit.bind(this);

    this.state = {

      person\_name: '',

      business\_name: '',

      business\_gst\_number:''

    }

  }

// selaimen ja palvelimen väliseen kommunikaatioon axios-kirjasto

// Reaction vakiofunktiot:

// componentDidMount() - suoritaan kerran heti ensimmäisen renderoinnin jälkeen

  componentDidMount() {

      axios.get('http://localhost:4000/business/edit/'+this.props.match.params.id)

          .then(response => {

              this.setState({

                person\_name: response.data.person\_name,

                business\_name: response.data.business\_name,

                business\_gst\_number: response.data.business\_gst\_number });

          })

          .catch(function (error) {

              console.log(error);

          })

    }

  onChangePersonName(e) {

    this.setState({

      person\_name: e.target.value

    });

  }

  onChangeBusinessName(e) {

    this.setState({

      business\_name: e.target.value

    })

  }

  onChangeGstNumber(e) {

    this.setState({

      business\_gst\_number: e.target.value

    })

  }

// tietojen päivitys palvelimelle (onSubmit)

  onSubmit(e) {

    e.preventDefault();

    const obj = {

      person\_name: this.state.person\_name,

      business\_name: this.state.business\_name,

      business\_gst\_number: this.state.business\_gst\_number

    };

    axios.post('http://localhost:4000/business/update/'+this.props.match.params.id, obj)

        .then(res => console.log(res.data));

    this.props.history.push('/index');

  }

  render() {

    return (

        <div style={{ marginTop: 10 }}>

            <h3 align="center">Päivitä tietoja</h3>

            <form onSubmit={this.onSubmit}>

                <div className="form-group">

                    <label>Henkilon nmi:  </label>

                    <input

                      type="text"

                      className="form-control"

                      value={this.state.person\_name}

                      onChange={this.onChangePersonName}

                      />

                </div>

                <div className="form-group">

                    <label>Yrityksen nimi: </label>

                    <input type="text"

                      className="form-control"

                      value={this.state.business\_name}

                      onChange={this.onChangeBusinessName}

                      />

                </div>

                <div className="form-group">

                    <label>Numero: </label>

                    <input type="text"

                      className="form-control"

                      value={this.state.business\_gst\_number}

                      onChange={this.onChangeGstNumber}

                      />

                </div>

                <div className="form-group">

                    <input type="submit"

                      value="Päivitä tiedot"

                      className="btn btn-primary"/>

                </div>

            </form>

        </div>

    )

  }

}

1. **Poista dataa**

Tehdään **delete**-funktio **TauluRivit.js**-tiedoston sisälle.

// TauluRivit.js

import React, { Component } from 'react';

import { Link } from 'react-router-dom';

import axios from 'axios';

class ListanRivi extends Component {

  constructor(props) {

        super(props);

        this.delete = this.delete.bind(this);

    }

    delete() {

        axios.get('http://localhost:4000/business/delete/'+this.props.obj.\_id)

            .then(console.log('Poistettu'))

            .catch(err => console.log(err))

    }

  render() {

    return (

        <tr>

          <td>

            {this.props.obj.person\_name}

          </td>

          <td>

            {this.props.obj.business\_name}

          </td>

          <td>

            {this.props.obj.business\_gst\_number}

          </td>

          <td>

            <Link to={"/edit/"+this.props.obj.\_id} className="btn btn-primary">Muokkaa</Link>

          </td>

          <td>

            <button onClick={this.delete} className="btn btn-danger">Poista</button>

          </td>

        </tr>

    );

  }

}

export default ListanRivi;

Tämä materiaali löytyy myös GitHubista: <https://github.com/KrunalLathiya/ReactCRUDExample>

