OULUN AMMATTIKORKEAKOULU

Joonas Alaruikka

# LINJA-AUTON SUUNNITTELUTYÖKALUN TOTEUTUS REACT.JS-KIRJASTON AVULLA

# LINJA-AUTON SUUNNITTELUTYÖKALUN TOTEUTUS REACT.JS-KIRJASTON AVULLA

Joonas Alaruikka
Opinnäytetyö
Syksy 2021
Tietojenkäsittelyn tutkinto-ohjelma
Oulun ammattikorkeakoulu

**TIIVISTELMÄ**

Oulun ammattikorkeakoulu
Tietojenkäsittelyn tutkinto-ohjelma

---

Tekijä: Joonas Alaruikka
Opinnäytetyön nimi: Linja-auton suunnittelutyökalun toteutus React.js-kirjaston avulla
Työn ohjaaja: Jouni Juntunen
Työn valmistumislukukausi ja -vuosi: Syksy 2021 Sivumäärä: 40

---

Tämän opinnäytetyön tarkoituksena oli toteuttaa web-sovellus linja-auton suunnittelemista varten ReactJS-tekniikan sekä tarvittavien lisäosien ja laajennusten avulla. Tavoitteena oli kehittää sovelluksen ensimmäinen, yksinkertainen mutta julkaisukelpoinen versio. Työn toimeksiantajana toimi linja-autoja myyvä Fibus Oy. Yrityksellä ilmeni kiinnostusta tämän kaltaiselle sovellukselle, jolla asiakkaat voivat rakentaa linja-autoja haluamillaan ominaisuuksilla ja lisäpalveluilla sekä jättää lopuksi yhteydenottopyynnön myyjälle.

Teoriaosuudessa tutkitaan ReactJS-tekniikkaa, Redux-tilanhallintaa, Firebase-tietokantaa sekä paketinhallintaa. Aineistoina on käytetty kehittäjien virallisia dokumentaatioita ja internetartikkeleita. Tietoperustan jälkeen siirrytään käytännön toteutukseen, jossa toteutetaan itse sovellus edellä mainittuja tekniikoita, lisäosia ja tietokantaa hyödyntäen.

Olennaisimmat tavoitteet saavutettiin, eli sovelluksesta saatiin toteutettua toimiva ensimmäinen versio, jota on helppo kehittää jatkossa yhteistyössä toimeksiantajan kanssa. Suurimmiksi kehityskohteiksi jäivät visuaalisuuden toteuttaminen kuvien avulla sekä loppujen valintavaihtoehtojen lisääminen. Tekijälleen työ opetti paljon kokonaisvaltaisemmasta ohjelmoinnista, jossa otetaan huomioon useampia osa-alueita.

---

Asiasanat: sovelluskehitys, verkkopalvelut, käyttöliittymä

**ABSTRACT**

Oulu University of Applied Sciences
Degree Programme in Business Information Systems

---

Author: Joonas Alaruikka
Title of thesis: Implementing a bus-configurator application with ReactJS
Supervisor: Jouni Juntunen
Term and year when the thesis was submitted: Autumn 2021
Number of pages: 40

---

The purpose of this thesis was to develop a bus-configurator application in collaboration with ReactJS library, Redux-state container, Firebase database and npm package manager. The aim of the thesis was to implement first version of the application that works and is ready for publication. The client of the work was bus dealership company Fibus Oy that was interested in this kind of application that customers could design buses with different features and eventually leave a contact request to the sales agent.

The theoretical part consists of research that introduces techniques used in the implementation part. The practical part of the thesis describes the development and design process of the application.

The end result was a working first version of the application that is easy to develop in the future. With this thesis, the author learned programming in a more diverse way in different areas with new skills and techniques.

---

Keywords: web-development, user interface, front-end, React

# SISÄLLYS

# 1 JOHDANTO

Useilla autonvalmistajilla on nykyisin kotisivujen yhteydessä suunnittelutyökalu tai konfiguraattori, jolla on mahdollista suunnitella auto haluamillaan ominaisuuksilla ja joko tilata sellainen suoraan sovelluksen kautta tai vaihtoehtoisesti jättää yhteydenottopyyntö. Myös raskaan kaluston puolelta ainakin Volvolla löytyy kotisivuiltaan työkalu, jolla voi rakentaa haluamansa linja-autoratkaisun yritykselle.

Tämän opinnäytetyön tarkoituksena on soveltaa edellä mainittua ajatusta ja toteuttaa vastaavanlainen sovellus web-selaimelle siten, että sillä on mahdollista suunnitella käytetty linja-auto ennalta määritellyillä vaihtoehdoilla ja ominaisuuksilla sekä jättää lopuksi yhteydenottopyyntö myyjälle. Toimeksiantajayritys on kasvava ja innovatiivinen toimija alallaan ja työn tavoitteena onkin lisätä yrityksen myyntiä, tukea sen markkinointia sekä laajentaa palveluvalikoimaa nykyaikaisen teknologian avulla.

Työn tilaaja on käytettyjä linja-autoja myyvä Fibus Oy, jonka toiminta-ajatuksena on räätälöidä käytetystä linja-autosta asiakkaan tarpeita vastaava työkalu tarjoamalla autoihin erilaisia muutostöitä ja lisäpalveluita. Tästä syystä yrityksellä onkin vahva kiinnostus sekä tarve tämän kaltaiselle sovellukselle liiketoimintaansa tukemaan ja tuomaan näkyvyyttä sekä tietoisuutta palveluilleen. Sovelluksesta on hyötyä myös asiakkaille, koska sen avulla he voivat ilmaista tarpeitaan selkeämmin ja räätälöidä tarkemmin haluamiaan ratkaisuja linja-autojen hankinnan suhteen. Opinnäytetyön tekijän tavoitteena on syventää sekä laajentaa osaamista web-kehityksen ja ohjelmoinnin parissa ja tätä kautta tukea ammatillista osaamista ja tietotaitoa nykyisissä ja tulevissa työtehtävissä. Tämän kaltainen työ tukee tätä tavoitetta mainiosti, sekä opettaa myös etsimään ja hyödyntämään teoriaa ja tietoa nopeasti kehittyvällä sekä muuttuvalla alalla.

Toteutettava työkalu on sovellus, joka koostuu eri näkymistä. Näkymä vaihtuu aina, kun käyttäjä suorittaa tarvittavan valinnan/valinnat. Näitä näkymiä ovat muun muassa linja-auton yleistiedot (kategoria, merkki, malli, vuosimalli, ovet), tekniikka (moottori, päästöluokka, vetotapa, ajokilometrit), sisäpuoli (paikkaluku, WC, verhoilu) sekä lisäpalvelut (maalaus, teippaukset, päästöluokan muutokset, viihdejärjestelmät). Sovelluksessa näkyy hinta-arvio omana elementtinään ja se muuttuu valintojen perusteella. Viimeinen näkymä on yhteenveto valituista ominaisuuksista ja se sisältää

myös yhteydenottolomakkeen. Työ on tarkoitus rajata siten, että toteutetaan yksinkertainen ensimmäinen, toimiva ja julkaisukelpoinen versio, jota on helppo kehittää edelleen. Visuaalisuus pyritään pitämään ensimmäisessä versiossa yksinkertaisena siten, että käytetään 2D-grafiikkaa kuvituksen ja autojen mallintamisen osalta, koska 3D-elementtien toteuttaminen vaatii erityisosaamista, mikä on liian iso kokonaisuus toteutettavaksi tähän työhön.

Opinnäytetyössä selvitetään, mitä tällaisen sovelluksen toteuttaminen vaatii ja kehittämistehtävänä toteutetaan sen ensimmäinen versio React.js -kirjastoa ja sen lisäosia kuten Redux-tilanhallintaa ja Firebase-tietokantaa hyödyntäen. Tarvittavat asennukset suoritetaan Node Package Manager (npm) -komennoilla. Kehitystyö tapahtuu Microsoft Visual Studio Code-ympäristössä aluksi paikallisesti ja lopuksi sovellus julkaistaan yrityksen kotisivujen yhteyteen. Edellä mainitut työkalut ja tavat on valittu, koska ne ovat toimivia ja hyväksi havaittuja tapoja toteuttaa erilaisia web-sovelluksia ja niitä käytetään laajalti tämän päivän web-kehityksessä ympäri maailmaa.

Edellä mainittujen työkalujen valintaan on päädytty tutkimalla ja vertailemalla eri vaihtoehtoja ja tultu siihen johtopäätökseen, että sovelluksen toteutus onnistuu mahdollisimman hyvin niitä hyödyntäen. Valintaan vaikutti myös se, että kyseiset menetelmät ovat tällä hetkellä erittäin suosittuja web-kehityksen saralla, esimerkiksi ReactJS on tämän hetken suosituin JavaScript-kirjasto. (Elliott 2020).

# 2 WEB-KEHITYKSESSÄ KÄYTETTÄVIÄ TEKNIIKOITA

## 2.1 ReactJS

ReactJS on deklaratiivinen, tehokas ja joustava JavaScript-kirjasto, jolla voidaan rakentaa käyttöliittymiä uudelleenkäytettävistä komponenteista. Se perustuu avoimeen lähdekoodiin ja on komponenttipohjainen front-end kirjasto, eli se vastaa sovelluksen näkymäosiosta. Reactin kehitti Facebookilla työskennellyt ohjelmistosuunnittelija Jordan Walke. Sitä käytettiin aluksi Facebookin uutisvirran toteutuksessa vuonna 2011, yleisesti se julkaistiin toukokuussa 2013. Myöhemmin ReactJS on ollut käytössä muun muassa WhatsAppin ja Instagramin kehityksessä. ReactJS:n lisäksi on olemassa React Native, joka on tarkoitettu mobiilisovellusten kehittämistä varten. (JavaTpoint 2018).

Nykyisin suurin osa web-sivustoista perustuu MVC (Model-View-Controller) -arkkitehtuuriin. React on tässä arkkitehtuurissa View-osiossa, jolloin muu arkkitehtuuri toteutetaan esimerkiksi Redux- tai Flux -tilanhallintaa käyttäen. (JavaTpoint 2018).

ReactJS-sovellus koostuu useista komponenteista, joista jokainen tuottaa pienen, uudelleenkäytettävän HTML-koodin osan. Komponentteja (katso kuva 1) voidaan käyttää myös toistensa sisällä monimutkaisempien sovellusten rakentamiseksi yksinkertaisemmalla tavalla. React käyttää virtuaalista dokumenttioliomallia eli DOM:ää. Tämä mahdollistaa sovelluksen nopean toiminnan, koska se lataa yksittäisiä DOM-elementtejä kerrallaan, eikä jokaisella kerralla koko DOM:ää. (JavaTpoint 2018).

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

*KUVA 1. Funktiokomponentti (Facebook Inc. 2021.)*

Komponentit antavat mahdollisuuden pilkkoa käyttöliittymää itsenäisiin ja uudelleenkäytettäviin palasiin ajatellen kutakin palasta erikseen. Yksinkertaisin tapa komponentin luomiseksi on JavaScript-

funktion kirjoittaminen. Yllä oleva funktio on kelvollinen React-komponentti, koska se sisältää propsin eli ominaisuuden objektiargumentin datan kanssa ja palauttaa React-elementin. Tällaisia komponentteja kutsutaan funktiokomponenteiksi, sillä ne ovat nimenomaan JavaScript-funktioita. Toinen vaihtoehto on käyttää alla olevassa kuvassa esiintyvää ES6:n eli EcmaScript-standardin mukaista luokkaa komponentin määritykseen. (Facebook Inc. 2021).

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

*KUVA 2. Luokkakomponentti (Facebook Inc. 2021.)*

Render-funktio palauttaa kuvauksen siitä, mitä näytöllä halutaan näkyvän. React käyttää kuvausta ja näyttää tuloksen. Käytännössä render-funktio palauttaa React-elementin, joka on kevyt kuvaus siitä, mitä halutaan tulostaa. Useimmat React-kehittäjät käyttävät erityistä syntaksia nimeltään JSX (katso kuva 3), jonka avulla rakenteet ovat helpompia kirjoittaa. (Facebook Inc. 2021).

```
const element = <h1>Hello, world!</h1>;
```

*KUVA 3. JSX-muuttujan määrittely (Facebook Inc. 2021.)*

JSX ei ole merkkijonoa eikä HTML-kieltä, vaan se on JavaScriptin syntaksilaajennus. Se tuottaa React-elementtejä ja sen käyttämistä suositellaankin käyttöliittymän ulkoasun määrityksessä Reactissa. Se saattaa muistuttaa sivupohjakieltä, mutta perustuu täysin JavaScriptiin. Alla olevassa esimerkkikuvassa määritellään muuttuja nimeltään “nimi” ja käytetään sitä JSX:n sisällä kirjoittamalla se aaltosulkujen sisälle. (Facebook Inc. 2021).

```
const name = 'Josh Perez';
const element = <h1>Hello, {name}</h1>;

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

*KUVA 4. Muuttujan käyttö JSX:n sisällä (Facebook Inc. 2021.)*

React-sovelluksen pienin rakennuspalikka on elementti. Se määrittää, mitä näytöllä halutaan näkyvän. Toisin kuin selaimen DOM-elementeissä, Reactin elementit ovat yksinkertaisia olioita, joita on helppo luoda. Reactin DOM huolehtii, että DOM:n päivittyminen täsmää React-elementtien kanssa. Elementit voivat joskus sekoittua tunnetumman konseptin eli komponenttien kanssa, jotka muodostuvat elementeistä. (Facebook Inc. 2021).

```
<div id="root"></div>
```

*KUVA 5. Div-elementti HTML-tiedostossa (Facebook Inc. 2021.)*

Yllä olevaa div-elementtiä kutsutaan “root” eli juuri-DOM-elementiksi, koska kaikki sen sisällä oleva tieto hallitaan React DOM:n kautta. Pelkästään Reactilla rakennetuissa sovelluksissa on yleensä yksi juuri-DOM-elementti. Olemassa oleviin sovelluksiin, joihin React integroidaan, voidaan erillisiä juuri-DOM-elementtejä käyttää niin paljon kuin halutaan. React-elementin tulostukseen juuri-DOM-elementille välitetään molemmat parametrit ReactDOM.render() -funktiolle alla olevan kuvan mukaisesti. Näin ollen sivulle tulostuu “Hello, world” -teksti. (Facebook Inc. 2021).

```
const element = <h1>Hello, world</h1>;
ReactDOM.render(element, document.getElementById('root'));
```

*KUVA 6. Elementin tulostus (Facebook Inc. 2021.)*

Elementit ovat muuttumattomia, eli kun sellaisen kerran luo, ei sen lapsia tai määreitä voi enää muuttaa. Ainoa keino käyttöliittymän päivittämiseen onkin siis uuden elementin luominen ja sen välittäminen ReactDOM.render -funktiolle. (Facebook Inc. 2021).

Kaikkien React-komponenttien tulee toimia puhtaiden funktioiden lailla, eli ne eivät yritä muuttaa tulostuksiaan sekä palauttavat aina saman tuloksen ominaisuuksiinsa nähden. Käyttöliittymien ollessa dynaamisia ja ajan saatossa muuttuvia, on olemassa "state" eli tilakonsepti. Se mahdollistaa komponenttien tulostuksen muutoksen perustuen käyttäjän toimintoihin, verkon vastauksiin ja kaikkeen muuhun rikkomatta puhtaan funktion sääntöä. (Facebook Inc. 2021).

Tila on saman kaltainen propsin eli ominaisuuden kanssa sillä erotuksella, että se on yksityinen ja täysin komponentin hallittavissa (Facebook Inc. 2021.) Ne molemmat ovat tavallisia JavaScript-olioita ja kantavat tietoa, joka vaikuttaa renderöinnin tulokseen. Merkittävä ero on kuitenkin se, että propsit välittyvät komponentille samoin kuin funktion parametrit, kun taas tilaa hallitaan komponentin sisällä samalla tavalla kuin muuttujia, jotka on määritelty funktion sisälle. (Facebook Inc. 2021).

```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }

  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}

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

*KUVA 7. Paikallisen tilan luominen luokkakomponenttiin (Facebook Inc. 2021.)*

Yllä olevassa kuvassa käsittelijä eli Constructor-luokka määrittää alkuarvon this.state:lle. Propsit välitetään käsittelijälle, koska luokkakomponenttien tulee aina kutsua käsittelijää niiden kanssa. (Facebook Inc. 2021).

Reactin versiossa 16.8 julkaistiin uutena lisäyksenä “Hooks”. Ne mahdollistavat tilan käyttämisen ja muita Reactin ominaisuuksia ilman luokkien kirjoittamista. Perusteluina Hooksien kehittämiselle ovat, että ne ratkaisevat lukuisia irrallisia ongelmia, joita Facebookilla on kohdattu yli viiden vuoden komponenttien kirjoittamisen sekä ylläpitämisen aikana. Ne auttavat muun muassa käyttämään tilallista logiikkaa uudelleen ilman komponenttihierarkian muuttamista. Lisäksi Hooksien avulla komponentit pystytään pilkkomaan pienempiin funktioihin sillä perusteella, mitkä osat liittyvät toisiina, kuten tilauksen määritys tai tietojen nouto. On myös huomattu, että luokat hämmentävät niin ihmisiä kuin koneitakin. Niinpä myös tästä syystä Hooksit antavat käyttää enemmän Reactin ominaisuuksia ilman luokkia. Käsitteellisesti React-komponentit ovat aina olleet lähempänä funktioita ja Hooksit omaksuvat funktiot kuitenkaan uhraamatta Reactin käytännön ydintä. Ne eivät myöskään vaadi monimutkaisten tai reaktiivisten ohjelmointitekniikoiden opettelua. (Facebook Inc. 2021).

Hooksien käyttöä suositellaan jatkossa uusien komponenttien kirjoittamisessa. Olemassa olevien luokkien korvaamista Hookseilla ei kuitenkaan suositella, ellei tarkoituksena ole kirjoittaa niitä uudelleen joka tapauksessa. Kehittäjien tavoitteena on korvata Hookseilla kaikki luokkien käyttö mahdollisimman pian. (Facebook Inc. 2021).

```
import React, { useState } from 'react';

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

*KUVA 8. UseState-hookin käyttö tilakomponentissa (Facebook 2021.)*

Yllä olevassa esimerkkikuvassa luodaan laskuri, jossa arvo lisääntyy nappia painamalla. UseState on Hook, jota kutsutaan funktiokomponentin sisällä ja lisätään sille paikallinen tila. React taltioi tämän tilan tulostusten välillä. UseState palauttaa parin, eli sen hetkisen tilan arvon ja funktion, jolla sitä pystyy päivittämään. Se on samankaltainen kuin luokkakomponentin this.setState, mutta se ei yhdistä uutta ja vanhaa tilaa. Ainoa argumentti, jota useState käyttää, on alkuperäinen tila. Yllä olevassa esimerkissä se on 0, koska laskuri alkaa nollasta. Toisin kuin this.state:ssa, tilan ei tarvitse välttämättä olla olio ja alkuperäisen tilan argumenttia käytetään vain ensimmäisen tulostuksen yhteydessä. (Facebook Inc. 2021).

## 2.2 Redux

Redux on tilojen hallintaan tarkoitettu lisäosa JavaScript-pohjaisissa sovelluksissa. Se sai alkunsa, kun Facebook-alustassa ilmeni ongelmia MVC-rakenteen kanssa ja dataan jouduttiin tekemään mittavaa skaalausta. Tällöin julkaistiin Flux-menetelmä, jolla pystyttiin päivittämään View-komponenttia yksisuuntaisesti sekä käsittelemään käyttäjäpohjaisia toimintoja. Fluxiin sekä Elm-kieleen perustuen Dan Abramov loi Reduxin kesäkuussa 2015 tuoden isoja muutoksia siten, että hän käytti vain yhtä tilaa ja poisti dispatch-menetelmän. Fluxin käyttäessä useampia tiloja, Reduxin myötä tarvitaan niitä vain yksi, joka hallitsee komponenttien tiloja. Tällä tavalla pystytään välittämään mikä tahansa tila mihin tahansa komponenttiin, mikä tekee Reduxista tehokkaan ja ripeän mallin. (BairesDev 2021).

Yleisesti tilanhallinta on tapa helpottaa kommunikaatiota ja datan jakamista komponenttien välillä. Se luo konkreettisen datastruktuurin edustamaan sovelluksen tilaa, josta tietoa voi lukea ja kirjoittaa. Tällä tavalla voidaan nähdä normaalisti näkymättömissä olevia tiloja niiden kanssa työskennellessä. (Ighodaro 2021).

Suurin osa kirjastoista kuten React ja Angular on rakennettu siten, että niiden komponentit hallitsevat tilojansa sisäisesti ilman tarvetta ulkoisille kirjastoille tai työkaluille. Tämä toimii pienissä sovelluksissa, joissa on muutamia komponentteja, mutta sovelluksen kasvaessa tilojen hallinnasta tulee työlästä. (Ighodaro 2021).

Sovelluksessa, jossa tietoja jaetaan komponenttien välillä, voi olla vaikeaa tietää tarkalleen missä tilan tulisi sijaita. Ideaalitilanteessa komponentin datan tulisi olla vain yhdessä komponentissa, jolloin sen jakaminen sisarkomponenttien kanssa olisi vaikeaa. (Ighodaro 2021).

Esimerkiksi Reactissa tilan täytyy sijaita parent-komponentissa, jos tietoja halutaan jakaa sisarusten välillä. Tilan päivittäminen tapahtuu parent-komponetilta tulevalla metodilla ja välitettävinä propseina näille sisarkomponenteille. (Ighodaro 2021).

Alla olevassa kuvassa on esimerkki sisäänkirjautumiskomponentista Reactissa. Kirjautumiskomponentin syöttö vaikuttaa siihen, mitä sen sisarkomponentti ”Status” näyttää. Tässä esimerkissä parent-komponentin dataa ei tarvita, mutta sen täytyy tarjota tila, koska sen lapsikomponentin täytyy jakaa dataa. (Ighodaro 2021).

```
class App extends React.Component {
  constructor(props) {
    super(props);
    // First the Parent creates a state for what will be passed
    this.state = { userStatus: "NOT LOGGED IN"}
    this.setStatus = this.setStatus.bind(this);
  }
  // A method is provided for the child component to update the state of the
  // userStatus
  setStatus(username, password) {
    const newUsers = users;
    newUsers.map(user => {
      if (user.username == username && user.password === password) {
        this.setState({
          userStatus: "LOGGED IN"
        })
      }
    });
  }

  render() {
    return (
      <div>
        // the state is passed to the sibling as a props as is updated whenever
        // the child component changes the input
        <Status status={this.state.userStatus} />
        // this method is passed to the child component as a props which it
        // uses to change the state of the userStatus
        <Login handleSubmit={this.setStatus} />
      </div>
    );
  }
});
```

*KUVA 9. Esimerkki Kirjautumis-komponentista Reactissa (Ighodaro 2021.)*

Tapauksessa, jossa tila pitäisi saada jaettua komponenttipuussa kaukana toisistaan sijaitsevien komponenttien välillä se tulisi välittää komponentista toiseen niin monesti, kunnes se päätyisi oikeaan paikkaan. Käytännössä tila siis täytyy nostaa ylös lähimpään parent-komponenttiin ja seuraavaan, kunnes se saavuttaa ancestorin, joka on yhteinen molemmille tilaa tarvitseville komponenteille ja tämän jälkeen välittää se alas. Tämä tekee tilanhallinnasta haastavaa ja vähemmän ennakoitavaa. Dataa välittyy myös sellaisiin komponentteihin, jotka eivät tarvitse sitä. (Ighodaro 2021).

Tilanhallinta vaikeutuu, kun sovelluksesta tulee monimutkaisempi. Tästä syystä tilanhallintaa, kuten Reduxia tarvitaan helpottamaan tilojen ylläpitämistä. (Ighodaro 2021). Reduxin toimintaperiaate on yksinkertainen. Se sisältää keskeisen säilön, johon koko sovelluksen tilat tallentuvat. Näin ollen jokainen komponentti voi lähettää tallennetut tilansa ilman, että propseja tarvitsee lähetellä jatkuvasti alas toisten komponenttien kesken. Toiminta perustuu kolmeen rakennusosaan joita ovat actions eli toiminnat, store eli tila ja reducer eli funktio, joka ottaa sen hetkisen tilan, suorittaa toiminnan ja palauttaa uuden tilan. (Ighodaro 2021).

Actionit (katso kuva 10) ovat yksinkertaisesti tapahtumia. Ne ovat ainoa keino lähettää tietoa sovelluksesta Redux-säilöön. Data voi koostua käyttäjän toimista, API- eli rajapintakutsuista tai vaikkapa lomakkeen lähettämisestä. Actionien lähetys tapahtuu käyttämällä store.dispatch-funktiota. Ne ovat tavallisia JavaScript-olioita ja niillä täytyy olla tyyppiominaisuus ilmaistakseen suoritettavan tapahtuman tyypin. Lisäksi niillä tulee olla payload eli tietosisältö, joka sisältää tiedon siitä, minkä toiminnan tapahtuma suorittaa. (Ighodaro 2021).

```
{
  type: "LOGIN",
  payload: {
    username: "foo",
    password: "bar"
  }
}
```

*KUVA 10. Esimerkki actionista, joka voidaan suorittaa sisäänkirjautumisen yhteydessä (Ighodaro 2021.)*

Tapahtumat luodaan action creatorin eli tapahtumanluojan kautta. Alla olevassa kuvassa luodaan tapahtuma sisäänkirjautumista varten. (Ighodaro 2021).

```
const setLoginStatus = (name, password) => {
  return {
    type: "LOGIN",
    payload: {
      username: "foo",
      password: "bar"
    }
  }
}
```

*KUVA 11. Action creator eli tapahtumanluoja (Ighodaro 2021.)*

Reducerit ovat puhtaita funktioita, jotka ottavat sovelluksen sen hetkisen tilan, suorittavat actionin eli tapahtuman (katso kuva 12) ja palauttavat uuden tilan. Tilat on tallennettu olioiksi, ja ne täsmentävät kuinka sovelluksen tila muuttuu vastauksena säilöön lähetettyyn tapahtumaan. Ne perustuvat JavaScriptin reduce-funktioon, jossa yksittäinen arvo on laskettu useista arvoista callback- eli takaisinkutsufunktion suorittamisen jälkeen. (Ighodaro 2021).

```
const LoginComponent = (state = initialState, action) => {
    switch (action.type) {

      // This reducer handles any action with type "LOGIN"
      case "LOGIN":
          return state.map(user => {
              if (user.username !== action.username) {
                  return user;
              }

              if (user.password == action.password) {
                  return {
                      ...user,
                      login_status: "LOGGED IN"
                  }
              }
          });
default:
          return state;
      }
};
```

*KUVA 12. Reducerin toiminta Reduxissa. (Ighodaro 2021.)*

Kun kyseessä on puhtaat funktiot, välitetyn olion tiedot eivät muutu eikä sovellukseen tule sivuvaikutuksia. Koska kyseessä on sama olio, tuloksen tulisi olla aina sama. (Ighorado 2021).

Store eli säilö (katso kuva 13) ottaa sovelluksen tilan. Yleinen suositus on, että Redux-sovelluksessa pidetään vain yhtä tilaa. Tilaa voidaan pitää tallennettuna, päivittää sitä ja rekisteröidä tai poistaa rekisteröinti helper- eli avustajafunktioiden avulla. (Ighodaro 2021).

```
const store = createStore(LoginComponent);
```

*KUVA 13. Esimerkki storesta eli säilöstä Kirjautumis-komponentissa (Ighodaro 2021.)*

Tilaan suoritettavat tapahtumat palauttavat aina uuden tilan. Täten tila on helppo ja ennustettava. (Ighodaro 2021).

Reduxin myötä tarvitaan vain yksi yleinen tila säilössä ja kullakin komponentilla on pääsy tilaan. Tämä poistaa tarpeen välittää tilaa jatkuvasti komponentilta toiselle. Se mahdollistaa myös yksittäisen osan valinnan säilöstä tiettyä komponenttia varten ja tekee näin ollen sovelluksesta optimoidumman. (Ighodaro 2021).

Yhteenvetona Reduxia käytettäessä React-sovelluksessa ei tarvitse enää nostaa tiloja ylös. Tämän avulla on helpompaa jäljittää, mikä toiminta aiheuttaa mitäkin muutoksia. Komponenttien ei tarvitse tarjota tiloja tai funktioita lapsikomponenteilleen tietojen jakamiseksi keskenään, koska Redux hoitaa kaiken tämän. Tämä yksinkertaistaa sovellusta ja tekee siitä helpommin hallittavan. (Ighodaro 2021).

## 2.3 Firebase

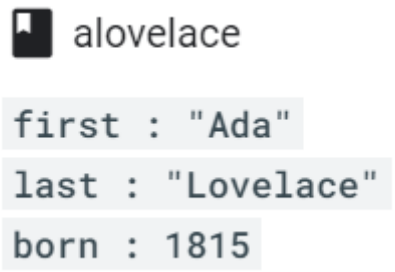
Firebase on Googlen työkaluja sisältävä kehitysympäristö, jolla on mahdollista rakentaa skaalautuvia sovelluksia. Se luokitellaan backend-as-a-service eli BaaS -malliin, joka mahdollistaa sovellusten kehittämisen helpommin siten, ettei backend-osiota tarvitse rakentaa tyhjästä. (Fawcett 2019).

Tietokannan valintaan Firebase tarjoaa kahta erilaista pilvipohjaista, client-käytettävää tietokantaratkaisua, uudempaa Cloud Firestorea sekä alkuperäistä Realtime Databasea, joista molemmat ovat NoSQL-tietokantoja. Eroavaisuuksina Cloud Firestoressa on uusi, intuitiivisempi tietomalli sekä laajemmat, nopeammat kyselyt ja parempi skaalaus. Lisäksi Realtime Database tallentaa datan yhteen suureen JSON-puuhun, kun taas Cloud Firestore käyttää dokumenttimuistia. Tämän ansiosta monimutkaisempi, luokiteltu data on helpompi skaalata käyttäen alikokoelmia dokumenttien yhteydessä. (Google Developers 2021).

NoSQL-tietokannat eroavat relaatiotietokannoista siten, että ne eivät käytä tauluja, sarakkeita, rivejä tai kaavioita tietojen järjestämisessä ja noutamisessa. Relaatiotietokantojen hallintajärjestelmät eivät ole kyenneet täyttämään riittävän hyvin nykyaikaisten sovellusten tarpeita suorituskyvyn, mukautuvuuden ja joustavuuden kannalta. Tästä johtuen monet suuryritykset ovatkin ottaneet käyttöön NoSQL-tietokantoja. Kyseinen tietokanta on erityisen hyödyllinen muun muassa strukturoimattoman datan tallennukseen, joka kasvaa huomattavasti nopeammin kuin strukturoitu data eikä näin ollen sovi relaatiotietokantojen malleihin. Yleisimmin strukturoimaton data sisältää esimerkiksi

käyttäjä- ja istuntotietoja, chat-viestejä ja lokitietoja, aikasarjatietoja sekä suuria olioita kuten videoita tai kuvia. Eri käyttötarpeita varten on luotu useita erilaisia NoSQL-tietomalleja, jotka on jaettu neljään pääluokkaan. Näitä luokkia ovat Key-value- eli avain-arvomuisti, Document- eli dokumenttimuisti, Wide-column- eli laaja sarakemuisti sekä Graph- eli kaaviomuisti. (Riak 2021).

Cloud Firestore on dokumenttipohjainen NoSQL-tietokanta. Siinä jokainen dokumentti (katso kuva 14) sisältää avain-arvoparin ja se on optimoitu tallentamaan laajoja kokoelmia pienistä dokumenteista ja jokainen dokumentti tulee tallentaa kokoelmaan. Dokumentti on kevyt tietue, joka sisältää kentät, jotka yhdistyvät arvoihin. Jokainen dokumentti tunnistetaan nimellä. (Google Developers 2021).




*KUVA 14. Dokumentti käyttäjästä nimeltä "alovelace" (Google Developers 2021.)*

Monimutkaisempia, sisäkkäisiä olioita kutsutaan mapeiksi. Esimerkiksi käyttäjän nimi voidaan rakentaa mapin avulla alla olevan kuvan mukaisesti. (Google Developers 2021).


*Kuva 15. Sisäkkäinen olio mapin avulla rakennettuna (Google Developers 2021.)*

Dokumentit muistuttavat JSON-dataa, joita ne itse asiassa ovatkin. Joitain eroja kuitenkin on, kuten se, että dokumentit tukevat useampia tietotyyppejä yhden megabitin kokoon saakka. Yleisesti niitä voidaan kuitenkin käsitellä kuten kevyitä JSON-tietueita. (Google Developers 2021).

Kokoelmat ovat yksinkertaisuudessaan säilöjä dokumentteja varten. Esimerkiksi alla olevassa kuvassa säilö nimeltä “käyttäjät” sisältää eri käyttäjiä, joista jokainen on esitetty dokumenttina. (Google Developers 2021).


*KUVA 16. Kokoelma käyttäjistä (Google Developers 2021.)*

Cloud Firestore ei vaadi skeemoja eli malleja, joten on vapaasti valittavissa, mitä kenttiä kuhunkin dokumenttiin määritetään ja mitä tietotyyppejä niihin tallennetaan. Kokoelman sisällä olevat dokumentit voivat sisältää erilaisia kenttiä tai näihin kenttiin voi tallentaa eri tietotyyppejä. Suositus kuitenkin on, että samoja kenttiä ja tietotyyppejä käytetään, jotta dokumenttien kyselyt ovat helpommin suoritettavissa. (Google Developers 2021).

Dokumenttien nimet ovat uniikkeja. Ne voidaan määrittää itse esimerkiksi luomalla käyttäjälle ID:n, tai vaihtoehtoisesti Cloud Firestore voi luoda satunnaisen ID:n automaattisesti. (Google Developers 2021).

Firebase Realtime Database on pilvipohjainen tietokanta, jossa data on tallennettu JSON-muodossa ja se synkronoidaan reaaliajassa jokaisen yhdistetyn clientin kanssa. Sen avulla voidaan rakentaa monipuolisia ja yhteistyökykyisiä sovelluksia sallimalla turvallisen pääsyn tietokantaan

suoraan client-puolen koodista. Data säilytetään paikallisesti, jolloin reaaliaikaiset tapahtumat jatkavat toimintaansa myös offline-tilassa. Kun yhteys palautuu, reaaliaikainen tietokanta päivittää tietomuutokset, jotka tapahtuivat offline-tilassa ja yhdistää kaikki ristiriidat automaattisesti. (Google Developers 2021).

Kaikki Firebase Realtime Databasen data tallennetaan siis JSON-olioina. Sitä voidaankin ajatella pilven ylläpitämänä JSON-puuna ja toisin kuin perinteisessä SQL-tietokannassa, se ei sisällä tauluja tai tietueita. Kun tietoja lisätään JSON-puuhun, muodostuu olemassa olevaan JSON-struktuuriin solmu, jolla on liitetty avain. Avaimen voi luoda itse esimerkiksi käyttäjän ID-numeroilla tai kuvaavilla nimillä tai ne voidaan muodostaa käyttämällä push() -funktiota. (Google Developers 2021).

Esimerkiksi chat-sovelluksessa, jossa käyttäjä voi tallentaa alla olevan kuvan mukaisesti profiilin ja yhteystietolistan, tyypillinen käyttäjäprofiili voisi sijaita polulla ”/users/$uid”. Tällöin käyttäjällä ”alovelace” voisi olla alla olevan kuvan mukainen tietokantamerkintä. (Google Developers 2021).

```
{
  "users": {
    "alovelace": {
      "name": "Ada Lovelace",
      "contacts": { "ghopper": true },
    },
    "ghopper": { ... },
    "eclarke": { ... }
  }
}
```

*KUVA 17. Tietokantamerkintä Realtime Databasessa (Google Developers 2021.)*

Tallennetut tiedot voidaan esittää tiettyinä natiivityyppeinä, jotka vastaavat käytettävissä olevia JSON-tyyppejä, vaikka tietokanta käyttääkin JSON-puuta. Tällöin voidaan kirjoittaa paremmin ylläpidettävää koodia. (Google Developers 2021).

## 2.4 Paketinhallinta

Paketinhallinta on järjestelmä, jolla hallinnoidaan projektien riippuvuuksia eli paketteja. Se tarjoaa menetelmän uusien pakettien asennukseen, määrittää mihin sijaintiin ne voidaan asentaa tiedostojärjestelmässä ja mahdollistaa myös omien pakettien julkaisemisen. (Mozilla 2021).

Paketit ovat kolmannen osapuolen ohjelmistoja, jotka voivat auttaa selvittämään ongelmia ohjelmistokehityksen eri vaiheissa. Yksittäinen web-projekti saattaa sisältää useita paketteja ja näillä riippuvuuksilla voi olla myös aliriippuvuuksia, joita ei asenneta erikseen. (Mozilla 2021).

Yksinkertainen esimerkki hyödyllisestä paketista on koodi, jolla lasketaan suhteelliset päivämäärät ihmisen luettavissa olevana tekstinä. Ratkaisu on mahdollista ohjelmoida itse, mutta melko varmasti joku muu on jo keksinyt sen, joten ajan säästämiseksi on järkevämpää käyttää valmista pakettia. Lisäksi luotettavaksi havaittu paketti on todennäköisesti testattu toimivaksi eri tilanteissa tehden siitä vakaamman ja eri selainten kanssa yhteensopivamman kuin itse kehitetty ratkaisu. (Mozilla 2021).

Projektissa oleva paketti voi olla kokonainen JavaScript-kirjasto kuten React tai Vue tai vastaavasti jokin hyvin pieni apuohjelma. Tällaisia apuohjelmia voivat olla esimerkiksi ihmisen luettavissa oleva päivämääräkirjasto tai komentorivityökalu. (Mozilla 2021).

Ilman moderneja rakennustyökaluja paketit sisällytetään projektiin käyttämällä yksinkertaista script-elementtiä. Tämä ei välttämättä toimi pidemmän päälle, sillä koodin ja pakettien yhteen niputtamiseksi tarvitaan todennäköisesti moderneja työkaluja. Tässä voidaan käyttää bundlea, jota käytetään viittaamaan web-palvelimella tiedostoon, joka sisältää ohjelmiston kaiken JavaScript-koodin. Se on pakattu mahdollisimman pieneksi vähentämään aikaa ohjelmiston latauksen ja käyttäjälle näyttämisen välillä. (Mozilla 2021).

Projektin koon kasvaessa edellä mainittu menetelmä osoittautuu helposti haastavaksi, kun paketteja on paljon ja niitä halutaan esimerkiksi päivittää tai vaihtaa kokonaan toiseen pakettiin. Tällöin on järkevämpää käyttää paketinhallintamenetelmää kuten npm:ää, koska sen avulla koodi lisätään ja poistetaan siistimmin. Lisäksi paketinhallinta käsittelee myös päällekkäisiä paketteja, mikä on tärkeää front-end-kehityksessä. (Mozilla 2021).

Jotta paketinhallinta toimii, se tarvitsee tiedon, mihin paketteja asennetaan. Tämä tapahtuu pakettirekisterin kautta, johon paketti julkaistaan ja josta se voidaan asentaa. (Mozilla 2021).

Yksi suurimmista pakettirekistereistä on Npm eli node Package Manager. Se sisältyy Node.js -kehitysympäristöön ja sen avulla voidaan etsiä ja asentaa paketteja projekteihin ja sovelluksiin. (Belina 2021).

Npm koostuu kolmesta osiosta. Ensimmäinen osio on arkisto, jossa julkaistaan avoimen lähdekoodin Node-projekteja ja jossa kehittäjät voivat jakaa lähdekoodejaan toisten npm-käyttäjien kanssa. Esimerkiksi React, Angular ja jQuery voidaan ladata arkiston kautta. Toinen osio käsittää komentorivikäyttöliittymän, jolla suoritetaan pakettien asennukset (katso kuva 18) ja poistot sekä niiden hallinta komentorivityökalun tai terminaalin välityksellä. Paketit ladataan komentokäyttöliittymän avulla kolmannesta osiosta eli rekisteristä, joka on JavaScript-ohjelmistotietokanta. (Belina 2021).

```
npm install express
```

*KUVA 18. Express-webkehyksen asentaminen npm-komennolla (Belina 2021.)*

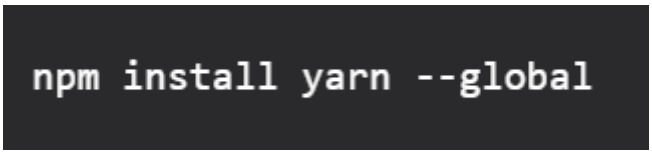
Jokaisella npm-projektilla on juuritiedosto (katso kuva 19) nimeltään package.json. Se sisältää projektien ja pakettien tarkemmat tiedot, kuten versio- ja kehittäjätiedot. Tiedosto tekee pakettien tunnistamisesta, hallinnasta ja asennuksesta yksinkertaisempaa. Tämän vuoksi onkin tärkeää sisällyttää package.json -tiedosto projektiin, ennen kuin se julkaistaan npm-rekisteriin. (Belina 2021).

```
{
  "name": "hostinger-npm",
  "version": "1.0.0",
  "description": "npm guide for beginner",
  "main": "beginner-npm.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "dependencies": {
    "express": "^4.17.1"
  },
  "repository": {
    "type": "git",
    "url": "https://github.com/hostinger/example.git"
  },
  "keywords": [
    "npm",
    "example",
    "basic"
  ],
  "author": "Hostinger International",
  "license": "MIT",
}
```

*KUVA 19. Esimerkki package.json -tiedostosta (Belina 2021.)*

Npm ei ole ainoa saatavilla oleva paketinhallintajärjestelmä. Toinen suosittu ja yleisesti käytetty vastaava järjestelmä on Yarn. Se ratkaisee riippuvuudet käyttäen erilaista algoritmiä, joka voi tarkoittaa muun muassa nopeampaa käyttäjäkokemusta. (Mozilla 2021).

Yarn tulee lyhenteestä Yet Another Resource Negotiator. Se on Facebookin kehittämä ja perustuu nykyisin avoimeen lähdekoodiin. Yarn kehitettiin korjaamaan npm:än suorituskyky- ja turvallisuusongelmia. Sen asennus tapahtuu (katso kuva 20) npm:än kautta. (GeeksforGeeks 2020).




*KUVA 20. Yarnin asennus npm:än kautta (GeeksforGeeks 2020.)*

# 3 SOVELLUKSEN TOTEUTUS

Tavoitteena oli toteuttaa yksinkertainen mutta toimiva sovelluksen ensimmäinen versio, jota on helppo kehittää edelleen. Eri vaihtoehtojen tutkimisen jälkeen toteutustavaksi valikoitui lomakepohjainen sovellus, jossa on useampia näkymiä. Ensimmäisessä näkymässä on auton perustiedot, jossa valitaan merkki, malli ja pituus. Toisessa näkymässä valitaan paikkaluku ja päästöluokka sekä lisäpalveluita autolle. Kolmas ja viimeinen näkymä sisältää yhteenvedon tehdyistä valinnoista ja siitä on mahdollista lähettää lomakkeen kautta yhteydenottopyyntö myyjälle.

Sovelluksen ohjelmoinnissa käytettiin Visual Studio Code-tekstieditoria, johon luotiin aluksi uusi React-projekti npx-create-react-app -komennolla. Tarvittavien pakettien ja lisäosien asennukset suoritettiin npm-komennoilla sekä otettiin käyttöön myös Git-versionhallinta varmuuskopiointia ja eri versioiden testaamista varten.

## 3.1 Tietokanta ja datarakenne

Sovelluksen data on suunniteltu siten, että se sisältää tietyt, yleisimmät autot ja niiden tarkemmat tiedot. Nämä tiedot haetaan tietokannasta ja näytetään sovelluksen käyttöliittymässä. Tietokantaa käytetään myös yhteydenottopyyntöjen tallentamiseen. Tämä tapahtuu käyttäjän täyttäessä ja lähettäessä yhteydenottolomakkeen.

Tässä työssä käytettävä tietokanta on Firebase Realtime Database. Kyseinen tietokanta osoittautui sopivimmaksi vaihtoehdoksi, koska sovelluksen data on kooltaan suhteellisen pieni ja sen rakenne on yksinkertainen.

Tästä syystä järkevin tapa oli luoda data ensin puhtaasti JSON-muodossa. Datan suunnittelu ja testaaminen helpottui, kun sitä pystyi käyttämään sovelluksessa paikallisesti ennen kuin se ajettiin tietokantaan.

Datan rakenne (katso kuva 21) on jaoteltu siten, että autoille on oma juurisolmu, joka sisältää auton merkin sekä merkkikohtaiset mallit ja pituudet omissa taulukoissaan. Päästöluokka ei ole merkistä tai mallista riippuvainen, joten sille on luotu oma yleinen solmunsa. Yhteydenotoille on myös oma

solmu, johon lähetetyt pyynnöt tallentuvat. Jokaisella yhteydenotolla on uniikki id, jonka Firebase luo, kun yhteydenotto rekisteröityy.

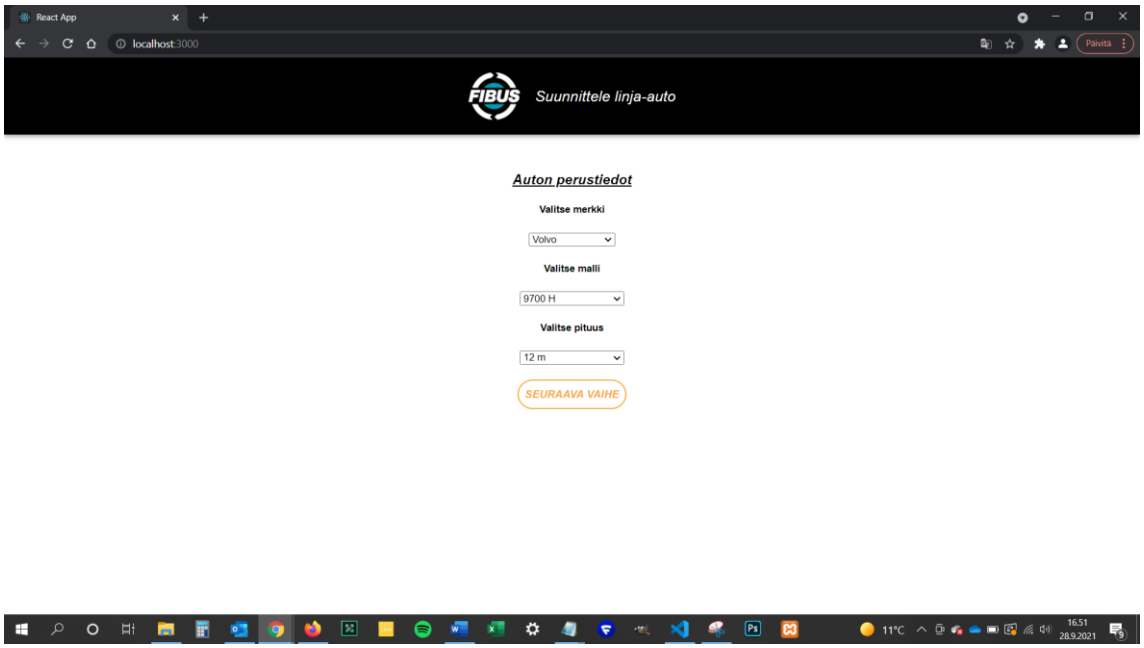

*KUVA 21. Esimerkkidataa JSON-muodossa*

Datan testaamisen ja tarvittavien muutosten jälkeen luotiin Firebaseen (katso kuva 22) tietokanta, johon JSON-data ladattiin. Firebase tarkistaa JSON-tiedoston kelpoisuuden, joten tässä vaiheessa tulee vielä tarkistus, että data on varmasti oikeassa muodossa.


*KUVA 22. Sovelluksen data Firebase Realtime Databasessa*

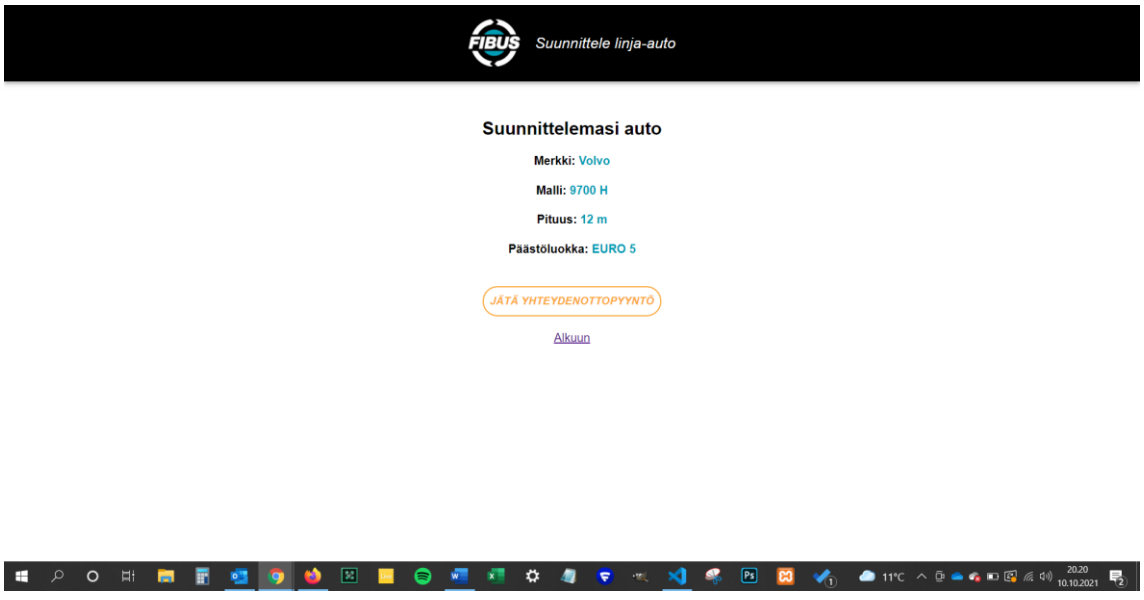
## 3.2 Sovelluksen rakenne

Toteutettu sovellus koostuu alkuvaiheessa kolmesta eri näkymästä. Ensimmäiset näkymät sisältävät (katso kuva 23) auton teknisien tietojen sekä ominaisuuksien valinnan. Valintojen data haetaan Firebasen tietokannasta. Seuraavaan näkymään siirrytään painamalla ”Seuraava vaihe” -painiketta.



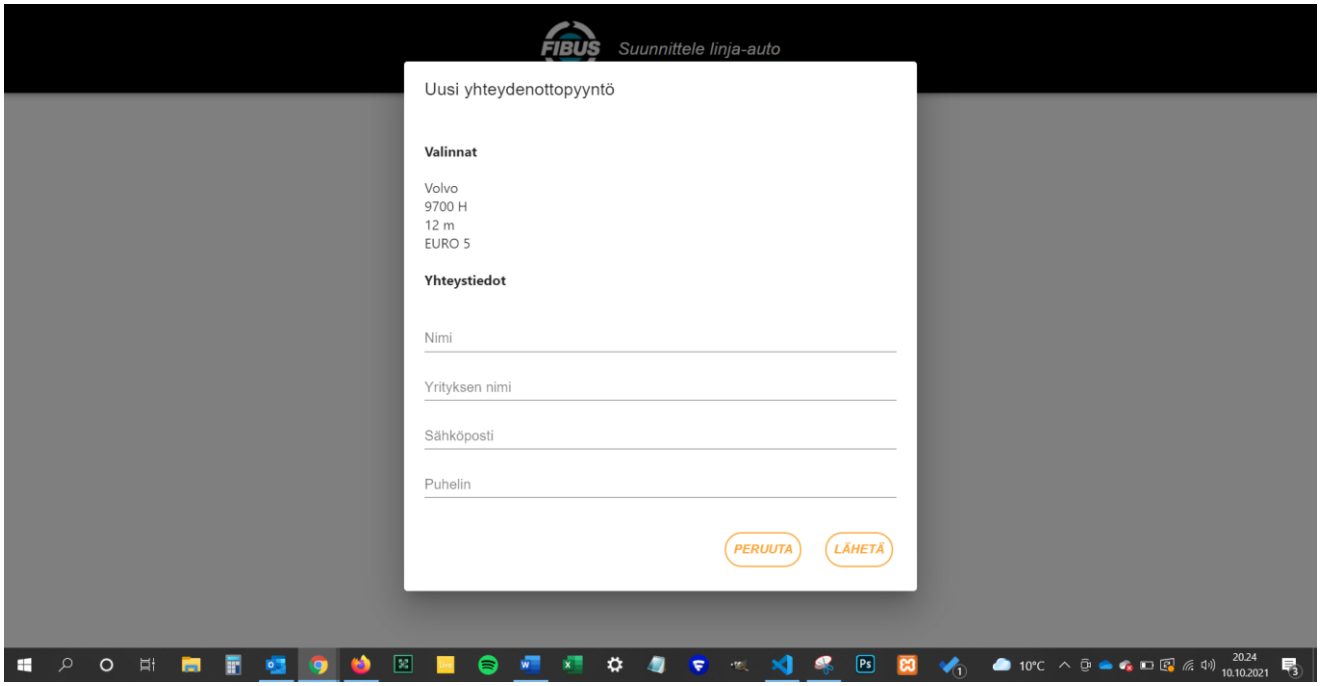

*KUVA 23. Sovelluksen käyttöliittymän perusnäkymä*

Kolmannessa ja viimeisessä näkymässä näytetään yhteenveto (katso kuva 24) tehdyistä valinnoista. Näkymään on toteutettu painike yhteydenottopyyntöä varten sekä linkki, jota painamalla pääsee takaisin alkuun.




*KUVA 24. Sovelluksen viimeinen näkymä*

Yhteydenottopyyntö-painiketta painamalla avautuu (katso kuva 25) lomake, jolla voi jättää myyjälle yhteydenottopyynnön. Lomakkeessa näytetään tehdyt valinnat sekä kentät, johon voi täyttää tarvittavat yhteystiedot. Lähetä-painiketta painaessa yhteydenottopyyntö tallentuu Firebasen tietokantaan.




*KUVA 25. Yhteydenottopyyntö-näkymä*

Käyttöliittymän suunnittelu helpottui huomattavasti toimeksiantajayrityksen brändinmuutosprosessin myötä, jolloin saatiin käyttöön graafinen ohjeistus värimaailmoineen ja typografioineen. Ulkoasu onkin toteutettu kyseisen ohjeistuksen pohjalta Material UI-komponenttien sekä CSS-tyylimääritysten avulla.

### 3.3 Komponenttien toteutus

App-komponentti (katso kuva 26) toimii sovelluksen pääkomponenttina. Autojen valinta- sekä yhteenvetokomponentit tuotiin pääkomponenttiin, ja luotiin reititys niiden välillä käyttämällä React Router -lisäosaa. Header-komponentti eli yläpalkki on luotiin omana komponenttinaan Material-UI -kirjaston avulla ja tuotiin pääkomponenttiin, jolloin se näkyy kaikissa näkymissä. App.css -tiedosto on olemassa tyyli- ja ulkoasumäärityksiä varten. Sen käyttö oli tässä työssä minimaalista ja sitä käytettiinkin lähinnä fonttien muotoiluun muiden ulkoasuelementtien tullessa Material UI -kirjaston kautta.

```
import React from 'react';
import { BrowserRouter as Router, Route } from 'react-router-dom';
import { Step1 } from './Step1';
import { Step3 } from './Step3';
import { Result } from './Result';
import { Header } from './components/Header';
import './App.css';

function App() {

  return (
    <div className="App">
    <Header />
    <Router>
      <Route exact path="/" component={Step1}></Route>
      <Route exact path="/step3" component={Step3}></Route>
      <Route exact path="/result" component={Result}></Route>
    </Router>
    </div>
  )
}

export default App;
```

*KUVA 26. Sovelluksen pääkomponentti App*

Sovelluksen ydintoiminta eli autojen ominaisuuksien valinta toteutettiin siten, että jokaiselle näkymälle luotiin aluksi oma komponentti. Data tuotiin Firebasen tietokannasta (katso kuva 27) useEffect-hookin ja fetch-funktion avulla.

```
  useEffect(() => {
      fetchData();
  }, [])

  const fetchData = () => {
      fetch('https://bussit-1352f-default-rtdb.europe-west1.firebasedatabase.app/cars.json')
  .then(response => response.json())
  .then(data => setBusData(Object.values(data)))
  .catch(err => console.error(err))
}
```

*KUVA 27. Datan tuonti Firebasesta komponenttiin*

Tietokannasta tuleva data kuten merkki, malli ja pituus tulostuvat omiin pudotusvalikoihin. Valitun merkin perusteella näytetään kyseisen merkin mallit ja pituudet. Tämä ominaisuus toteutettiin JavaScriptin find-menetelmällä, kuten alla olevasta kuvasta ilmenee. Menetelmä etsii datasta valitun merkin perusteella löytyvät merkit ja mallit. Valittu arvo on tilamuuttujan arvo.

```
const availableModel = busdata.find((b) => b.name === selectedBrand)
const availableLength = busdata.find((l) => l.name === selectedBrand);
```

*KUVA 28. Auton mallin ja pituuden hakeminen valitun merkin perusteella*

Seuraavaksi malli-taulukko käydään läpi map -funktion (katso kuva 29) avulla, johon on liitetty yllä olevassa kuvassa määritetty availableModel -muuttuja. Muuttujan perässä olevan kysymysmerkin eli ternary operatorin avulla tarkistetaan, ettei tarkistettava arvo ole null. Lopulta tulostuksessa palautuu valitun merkin perusteella saatavilla olevat mallit.

```
<option>Valitse malli</option>
{availableModel?.models.map((e, key) => {
    return (
        <option value={e.name} key={key}>
            {e.name}
        </option>
    )
})}
```

*KUVA 29. Auton mallien näyttäminen valitun merkin perusteella*

Käyttäjän tekemien valintojen tallentaminen sekä tietojen välittäminen toisille komponenteille toteutettiin Reduxin sekä React Hook Form -lisäosan avulla. Tämä toteutustapa osoittautui järkevimmäksi eri vaihtoehtojen vertailujen ja kokeilujen jälkeen.

Reduxin käyttöönotto suoritettiin asentamalla ensin Redux sekä sen konfigurointia helpottava Redux Toolkit -lisäosa. Toolkitin avulla Reduxin käyttöönotto on helpompaa, eikä koodia tarvita yhtä paljon sen valmiiden funktioiden ansiosta.

Ensimmäiseksi luotiin store-komponentti, jossa määritettiin (katso kuva 30) Reduxin store eli säilö. Sovelluksen tilat eli valitut ominaisuudet tallentuvat tähän säilöön.


*KUVA 30. Reduxin storen eli säilön määritys*

Säilöön välittyvät tilat luotiin omana komponenttinaan. Tähän komponenttiin luotiin (katso kuva 31) tilamuuttujat auton ominaisuuksille. Lisäksi luotiin reducerit, joiden sisällä on määritetty actionit eli tapahtumat, jotka suorittavat tilojen muutoksen Reduxissa. Lopuksi reducerit vietiin Store-komponenttiin ja sovelluksen käyttöliittymäkomponentteihin.

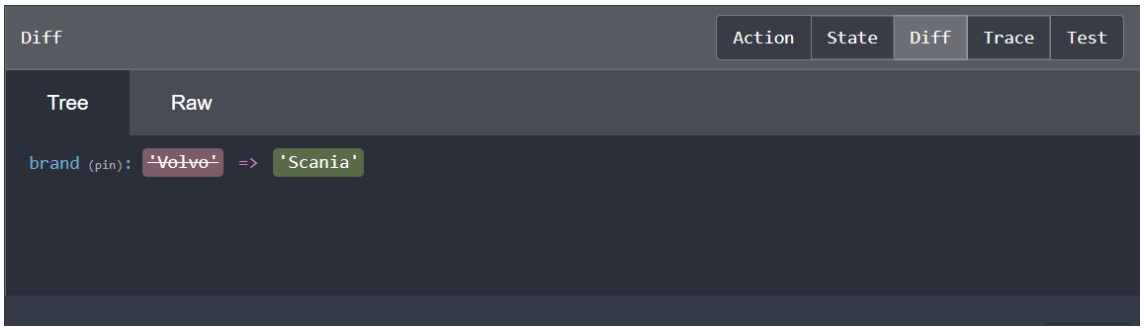

*KUVA 31. Reduxin tilat ja reducerit*

Valintojen tallennus toteutettiin (katso kuva 32) yhdistelemällä React Hook Formin, Reduxin ja React Routerin toimintoja. Toiminnallisuus liitettiin lomakkeen submit-ominaisuuteen siten, että kun käyttäjä on tehnyt valintansa ja painaa Seuraava vaihe -painiketta, suorittaa ohjelma funktion, joka tallentaa tehdyt valinnat Reduxin säilöön ja siirtää käyttäjän seuraavaan vaiheeseen.

```
const dispatch = useDispatch()
const history = useHistory()
const brand = useSelector(state => state.brand)
const model = useSelector(state => state.model)
const length = useSelector(state => state.length)
const { register, handleSubmit} = useForm({ defaultValues: { brand, model, length } })


const onSubmit = (data) => {
    dispatch(chooseBrand(data.brand))
    dispatch(chooseModel(data.model))
    dispatch(chooseLength(data.length))
    console.log(data)
    history.push("./step3")
}

return (

    <form onSubmit={handleSubmit(onSubmit)}>
```

*KUVA 32. Tietojen tallennus Reduxin säilöön*

Reduxin toiminnan testaaminen (katso kuva 33) onnistui helposti selaimeen asennettavan Redux DevTools -laajennuksen avulla. Laajennus toimii selaimen konsolista käsin ja sillä on mahdollista seurata muun muassa tiloja ja niiden muutoksia reaaliajassa.




*KUVA 33. Tilan muutos Redux DevTools -konsolissa*

Sovelluksen viimeistä näkymää varten luotiin Result-komponentti, jossa näytetään käyttäjän tekemät valinnat. Ne saatiin komponentin käyttöön (katso kuva 34) Reduxin välityksellä ja renderöitiin näkyville HTML-elementteinä.

```
export const Result = () => {
    const state = useSelector(state => state)


return (
    <div>

    <h2>Suunnittelemasi auto</h2>
    <h4>Merkki: <span className="result">{state.brand}</span></h4>
    <h4>Malli: <span className="result">{state.model}</span></h4>
    <h4>Pituus: <span className="result">{state.length}</span></h4>
    <h4>Päästöluokka: <span className="result">{state.euro}</span></h4>

    </div>

)
}
```

*KUVA 34. Yhteenveto-komponentti*

Yhteydenottolomaketta varten luotiin oma komponentti, joka vietiin Result-komponentille tietokantaan lähetystä varten. Lomakkeeseen tuotiin ensin tehdyt valinnat Reduxin välityksellä samalla tavalla kuin edellä tehtiin Result-komponentin kohdalla. Tämän lisäksi lisättiin (katso kuva 35) kentät yhteystietoja varten ja luotiin lomakkeen pohja toimintoineen Material-UI -kirjaston avulla.

```
function AddSubmit(props) {

const state = useSelector(state => state)
const [open, setOpen] = useState(false);
const [contact, setContact] = useState({
  brand: state.brand,
  model: state.model,
  length: state.length,
  euro: state.euro,
  name: '',
  company: '',
  email: '',
  phone: ''
});
```

*KUVA 35. Yhteydenottolomakkeen kenttien luonti*

Yhteydenoton lähetys tapahtuu POST-menetelemällä Firebasen tietokantaan. Sitä varten luotiin (katso kuva 36) yhteys tietokantaan fetch-menetelmällä ja luotiin funktio, jolla lähetys tapahtuu.

```
const addSubmit = (newSubmit) => {
    fetch('https://bussit-1352f-default-rtdb.europe-west1.firebasedatabase.app/submits.json',
        {
            method: 'POST',
            body: JSON.stringify(newSubmit)
        })
        .then(response => response.json)
        .catch(err => console.error(err))
}
```

*KUVA 36. Yhteydenottopyynnön lähetys tietokantaan*

# 4 POHDINTA

Tämän opinnäytetyön tavoitteena oli toteuttaa responsiivinen web-sovellus ReactJS -käyttöliittymäkirjaston avulla. Lisäksi työssä oli tarkoitus hyödyntää Redux-kirjastoa tilanhallinnan osalta sekä Firebasen tietokantaa datan tuontiin ja vientiin. Edellä mainitut teknologiat olivat tekijälle Reactin peruskurssia lukuun ottamatta täysin vieraita, joten aikaa kului varsinkin alussa perehtymiseen ja uuden opetteluun melko paljon. Teorian ja käytännön yhdistäminen onnistuikin siinä mielessä hyvin, että tietoperustaa kirjoittaessa tuli opittua koko ajan lisää, mikä helpotti osaltaan myöhempää teknistä toteutusta.

Päätavoitteet työssä saavutettiin keskeisimpien tekniikoiden osalta. Sovelluksesta saatiin toteutettua yksinkertainen ensimmäinen versio, jossa kaikki tarvittavat ominaisuudet toimivat ja se on pieniä viilauksia ja loppujen tietojen lisäämistä vaille julkaisukelpoinen. Pienempien tavoitteiden osalta ei kaikkiin aluksi suunniteltuihin tavoitteisiin päästy. Esimerkiksi grafiikkaa kuten autojen kuvia ei saatu lisättyä, koska se osoittautui haastavaksi ominaisuudeksi toteuttaa tämän kaltaiseen lomakepohjaiseen järjestelmään.

Oppimisen kannalta tämä opinnäytetyö antoi tekijälleen paljon. Aikaisempi ohjelmointikokemus oli lähinnä koulun kursseilta opittuja perusteita, joten siinä mielessä lähes kaikki front-end-kehityksen ja JavaScriptin perusteita lukuun ottamatta oli täysin uutta. Ennen opinnäytetyön aloittamista käyty Reactin peruskurssi auttoi jonkin verran, mutta esimerkiksi Reduxin tilanhallinta ja NoSQL-tietokanta olivat kokonaan uusia asioita. Työ opettikin kokonaisvaltaisempaa ja monipuolisempaa ohjelmoinnin ymmärrystä, jossa yhdistellään eri tekniikoita ja osa-alueita. Lisäksi se antoi näkemystä siihen, millaista työ ohjelmoinnin parissa voisi olla, kun tehdään oikeasti käyttöön tulevia sovelluksia. Myös oikeanlaisen tiedonhaun ja sen soveltamisen opetteleminen omaan käyttötarkoitukseen sopivaksi osoittautui tärkeäksi seikaksi.

Suurimmaksi haasteeksi työssä osoittautui selkeästi tekijän puutteellinen osaaminen ohjelmoinnissa, mikä johtui osaltaan siitä, että aiemmat koulun ohjelmointikurssit olivat sujuneet varsin hyvin ja tästä johtuen käsitys omista taidoista oli hieman yläkanttiin arvioitu. Tämä näkyi välillä ihan perusasioissa kuten esimerkiksi siinä, miten saadaan haettua autojen mallit valitun merkin perusteella. Redux tuotti kuitenkin selvästi eniten vaikeuksia, koska sen rooli oli niin suuri koko sovelluk-

sen toiminnan kannalta. Sen opettelussa ilmeni ongelmia varsinkin alussa, kun sen toimintaperiaate ei tahtonut avautua ja käyttöönotto sekä konfigurointi vaati niin paljon koodia ja komponentteja.

Tekniikoiden yhdistämisessä ilmeni myös haasteita, jotta Reactin, Reduxin ja Firebasen sai toimimaan keskenään. Lopulta tekniikat kuitenkin saatiin toimimaan kompromissien myötä siten, että Firestoren sijasta käytettiin Firebasen yksinkertaisempaa Realtime Databasea, jossa data on selkeämmässä JSON-muodossa. Reduxin osalta helpotukseksi osoittautui Toolkit-lisäosa, joka yksinkertaisti sen konfigurointia ja käyttöä sekä React Hook Form, jonka avulla lomakkeen tietojen tallennus ja välitys lopulta onnistui.

Omat haasteensa tuotti myös ajan puute, joka ilmeni lähinnä ajoittaisena työtuntien lisääntymisenä työpaikalla samaan aikaan opinnäytetyötä tehtäessä. Lisäksi toimeksiantaja ei ole ohjelmointialan yritys, joten teknistä apua ja tukea ei aina saanut riittävästi. Tämä asia oli tosin tiedossa jo ennen työn aloittamista, joten se oli tietoinen riski. Muuta tukea ja motivointia sekä haasteiden ymmärtämistä sen sijaan tuli riittävästi ja se auttoi vaikeimmilla hetkillä.

Sovellus saatiin toteutettua sille tasolle, että sitä on helppo kehittää jatkossa. Selkeimmät kehityskohteet ovat kuvien lisääminen autoihin ja eri vaiheisiin sekä yhteydenottopyynnön lähetys suoraan myyjän sähköpostiin nykyisen tietokantamerkinnän sijasta. Tämän pitäisi onnistua hyvin Firebasen kautta Google Workspace -integraationa. Tulevaisuudessa tekijän taitojen kehittyessä ja resurssien mahdollistuessa myös 3D-näkymän integroiminen sovellukseen voisi tulla kysymykseen. Tässä opinnäytetyössä saatiin tehtyä sovellukselle toimiva, yksinkertainen pohja, josta on hyvä jatkaa.

# LÄHTEET


BairesDev 2021. What is Redux and Why It Matters In Web Development. Hakupäivä 27.4.2021 https://www.bairesdev.com/technologies/what-is-redux-web-development/

Belina, Amanda 2021. What Is npm? A Basic Introduction to Node Package Manager for Beginners. Hakupäivä 03.11.2021 https://www.hostinger.com/tutorials/what-is-npm

Elliott, Eric 2020. Top JavaScript Frameworks and Tech Trends for 2021. Hakupäivä 18.4.2021 https://medium.com/javascript-scene/top-javascript-frameworks-and-tech-trends-for-2021-d8cb0f7bda69

Facebook Inc. 2021. What is React? Hakupäivä 26.4.2021 https://reactjs.org/tutorial/tutorial.html#what-is-react

Facebook Inc. 2021. What is React? Hakupäivä 17.5.2021 https://reactjs.org/tutorial/tutorial.html#what-is-react

Facebook Inc. 2021. Introducing JSX. Hakupäivä 17.5.2021 https://reactjs.org/docs/introducing-jsx.html

Facebook Inc. 2021. Rendering Elements. Hakupäivä 17.5.2021 https://reactjs.org/docs/rendering-elements.html

Facebook Inc. 2021. Components and Props. Hakupäivä 18.5.2021 https://reactjs.org/docs/components-and-props.html

Facebook Inc. 2021. State and Lifecycle. Hakupäivä 18.5.2021 https://reactjs.org/docs/state-and-lifecycle.html

Facebook Inc 2021. Component State. Hakupäivä 18.5.2021 https://reactjs.org/docs/faq-state.html

Facebook Inc. 2021. Hooks at a Glance. Hakupäivä 18.5.2021 https://reactjs.org/docs/hooks-overview.html

Facebook Inc. 2021. Hooks FAQ. Hakupäivä 24.5.2021 https://reactjs.org/docs/hooks-faq.html

Fawcett, Amanda 2019. A quick dive into Firebase: jumpstart your full-stack journey. Hakupäivä 02.06.2021 https://www.educative.io/blog/a-quick-dive-into-firebase

GeeksforGeeks. Difference between npm and yarn. Hakupäivä 9.6.2021 https://www.geeksforgeeks.org/difference-between-npm-and-yarn/

Google Developers 2021. Choose a Database: Cloud Firestore or Realtime Database. Hakupäivä 02.06.2021 https://firebase.google.com/docs/database/rtdb-vs-firestore

Google Developers 2021. Cloud Firestore Data Model. Hakupäivä 02.06.2021 https://firebase.google.com/docs/database/web/structure-data

Google Developers 2021. Structure Your Database. Hakupäivä 02.06.2021 https://firebase.google.com/docs/database/web/structure-data

Ighado, Neo 2021. Why use Redux? A tutorial with examples. Hakupäivä 9.5.2021 https://blog.logrocket.com/why-use-redux-reasons-with-clear-examples-d21bffd5835/

Javatpoint 2018. Introduction. Hakupäivä 9.5.2021 https://www.javatpoint.com/firebase-introduction

Javatpoint 2018. React Introduction. Hakupäivä 9.5.2021 https://www.javatpoint.com/react-introduction

Mozilla 2021. Package management basics. Hakupäivä 8.6.2021 https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Understanding_client-side_tools/Package_management#what_exactly_is_a_package_manager

Riak 2021. NoSQL Databases Explained. Hakupäivä 02.06.2021 https://riak.com/resources/nosql-databases/index.html?p=9937.html

Sufiyan, Taha 2021. What is React JS: Introduction To React and Its Features. Hakupäivä 6.5.2021. https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs