Teemu Perttula

Kaavakortiston luominen Seinäjoen kaupungille

Opinnäytetyö
Kevät 2019
SeAMK Tekniikka
Tietotekniikan tutkinto-ohjelma

SeAMK
SEINÄJOEN AMMATTIKORKEAKOULU
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

SEINÄJOEN AMMATTIKORKEAKOULU

Opinnäytetyön tiivistelmä

Koulutusyksikkö: Tekniikan yksikkö

Tutkinto-ohjelma: Tietotekniikka

Suuntautumisvaihtoehto: Ohjelmistotekniikka

Tekijä: Teemu Perttula

Työn nimi: Kaavakortiston luominen Seinäjoen kaupungille

Ohjaaja: Marko Hietamäki

Vuosi: 2019 Sivumäärä: 59 Liitteiden lukumäärä:0

Tämän opinnäytetyön toimeksiantajana toimi Seinäjoen kaupungin kaupunkisuunnittelun ja kaavoituksen -yksikkö. Opinnäytetyön tavoitteena oli muuttaa Seinäjoen kaupungin käytössä oleva Kaavakortisto-ohjelma toimimaan verkkoselaimessa.

Opinnäytetyössä perehdytään kolmeen eri kokonaisuuteen, jotka yhdessä muodostavat toimivan ohjelmiston, joka mahdollistaa tietokannan käytön selaimen kautta. Opinnäytetyössä tutustutaan tietokantojen perusteisiin, SQLite-tietokantaan ja SQLite-tietokannan ominaisuuksiin. Palvelintekniikoista käsitellään REST-ohjelmointirajapinnan luontia Node.js-ympäristöllä. Opinnäytetyössä käydään läpi myös käyttöliittymien suunnitteluun tarkoitetun ReactJS-kirjaston perusteita ja ominaispiirteitä.

Opinnäytetyön tuloksena syntyi selainkäyttöliittymä, jonka avulla pystytään hallitsemaan kaavoitusprojekteihin liittyviä tietoja. Käyttöliittymä hakee, muokkaa ja poistaa tietoja tietokannasta Node.js-ympäristöön luodun REST-ohjelmointirajapinnan kautta.

Avainsanat: kehitystutkimus, ohjelmisto, ReactJS, Node.js, REST, SQLite

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Thesis abstract

Faculty: School of Technology

Degree programme: Information Technology

Specialisation: Programming

Author: Teemu Perttula

Title of thesis: Creating a Building Plan Card File Application for the City of Seinäjoki

Supervisor: Marko Hietamäki

Year: 2019 Number of pages:59

This thesis was assigned by the city planning unit of the City of Seinäjoki. The aim was to replace the current building plan card file application with one that works via a browser.

The browser version of the card file requires three different components to work properly. First this thesis studied the basics of database designing and the features of SQLite. Next, attention was paid to the REST application programming interface (API) in the Node.js environment. Lastly, Facebook's ReactJS library that is mostly used for making user interfaces, was studied.

The result of this thesis was a functioning building plan card file application that works via a browser. The user interface of the card file application gets, modifies, and deletes data from a database using the REST API.

Keywords: development research, software, ReactJS, Node.js, REST, SQLite

SISÄLTÖ

Opinnäytetyön tiivistelmä 1
Thesis abstract 2
SISÄLTÖ 3
Kuvaluettelo 5
Käytetyt termit ja lyhenteet 8
1 JOHDANTO 10
1.1 Työn tausta 10
1.2 Työn tavoite 10
1.3 Työn rakenne 10
1.4 Seinäjoen kaupunki 11
1.4.1 Kaupunkisuunnittelu ja kaavoitus 12
1.4.2 Kaavakortisto 12
2 VANHA KAAVAKORTISTO 13
2.1 Vanha Kaavakortisto 13
2.1.1 Microsoft Access 13
2.1.2 Microsoft Access Runtime 13
2.1.3 Vanhan kaavakortiston käyttöliittymä 14
2.2 Tavoite 15
3 TIETOKANTA 16
3.1 Mikä on tietokanta? 16
3.2 Relaatiotietokanta 16
3.2.1 Taulut 16
3.2.2 Pääavain 16
3.2.3 Viiteavain ja viittaukset 17
3.3 SQLite 20
4 NODE.JS 21
4.1 Web-palvelimet ennen Node.js-ohjelmaa 21
4.2 Node.js 22
4.3 NPM 22
4.4 REST-ohjelmointirajapinta 23

4.4.1 Kommunikointi 23
4.5 Web-palvelimen luonti tietokantayhteydellä 24
4.5.1 Express 25
4.5.2 SQLite3 26
4.5.3 Middleware 27
4.5.4 Resurssien päivitys tietokantaan 29
5 REACTJS 31
5.1 ReactJS yleisesti 31
5.2 Komponentit 32
5.3 Virtual DOM 32
5.4 Properties 34
5.5 State 35
5.6 Lifecycle 35
5.7 ReactJS-sovelluksen luonti 38
5.7.1 Aloitus 38
5.7.2 JSX 39
5.7.3 Komponentit 40
5.7.4 Properties 40
5.7.5 State 42
5.7.6 Axios 43
6 TYÖN ETENEMINEN 46
6.1 Tietokanta 46
6.2 Web-palvelin ja käyttöliittymä 47
6.2.1 Etusivu 48
6.2.2 Kaavakortisto 49
6.3 Kaavoitusohjelma 54
7 POHDINTA 55
LÄHTEET 56

Kuvaluettelo

Kuva 1. Kaupunkiorganisaatio (Seinäjoki [Viitattu 6.2.2019]). 11

Kuva 2. Vanhan kaavakortiston aloitussivu 14

Kuva 3. Tietojenhallintalomake Kaavakortisto-ohjelmassa 15

Kuva 4. Yksi-yhteen-viittaus (perustuu Hock-Chuan 2010) 18

Kuva 5. Yksi-moneen-viittaus (perustuu Hock-Chuan 2010) 19

Kuva 6. Moni-moneen-yhteys (perustuu Hock-Chuan 2010) 19

Kuva 7. Express-sovelluskehyksen asennuskomento. 25

Kuva 8. Express-esimerkkisovellus. 26

Kuva 9. Vastaus oletusreitistä. 26

Kuva 10. SQLite3-moduulin asennuskomento. 26

Kuva 11. SQLite-tietokantahakusovelluksen esimerkki. 27

Kuva 12. Esimerkki tietokantahaun tuloksesta. 27

Kuva 13. Body-parser-asennuskomento. 28

Kuva 14. Body-parser-ohjelmiston käyttöönotto. 28

Kuva 15. CORS-asennuskomento. 28

Kuva 16. CORS-ohjelmiston käyttöönotto. 29

Kuva 17. Esimerkkiohjelma resurssin päivityksestä. 30

Kuva 18. Esimerkki palvelimelle lähettävästä resurssista. 30

Kuva 19. Kaavan nimi päivityksen jälkeen. 30

Kuva 20. Virtual DOM-rakenteen toimintaperiaate (perustuu Hamedani 2018) 33

Kuva 21. Propertiesin toimintamalli (perustuu Chinnathambi 2016)..... 34

Kuva 22. Create-react-app-komento..... 38

Kuva 23. Uuden projektin kansiorakenne..... 38

Kuva 24. Uuden projektin oletusnäkymä..... 39

Kuva 25. JSX-esimerkki..... 39

Kuva 26. Funktionaalikomponentti..... 40

Kuva 27. Luokkakomponentti..... 40

Kuva 28. Properties-esimerkki..... 41

Kuva 29. Ominaisuuksien vaikutus komponentteihin..... 41

Kuva 30. Yksinkertainen tilallinen komponentti..... 42

Kuva 31. Tilallinen komponentti selaimessa..... 43

Kuva 32. Axios-asennuskomento..... 43

Kuva 33. Axios-ohjelmiston käyttöönotto ReactJS-projektissa..... 43

Kuva 34. Yksinkertainen axios-pyyntö..... 44

Kuva 35. Web-palvelimen palauttama objekti..... 45

Kuva 36. Valmistuneen tietokannan kuvaus..... 47

Kuva 37 Käyttöliittymän komponenttirakenne..... 48

Kuva 38. Etusivu-komponentti..... 48

Kuva 39. Yleisdata-komponentti..... 49

Kuva 40. Kaavan lisäys..... 50

Kuva 41. Kaavan haku -ponnahdusikkuna..... 50

Kuva 42. Alanavigointi-komponentti. 50

Kuva 43. Projektihallinta-komponentti. 51

Kuva 44. Toimenpiteet-komponentti. 51

Kuva 45. Korttelivaraukset-komponentti 52

Kuva 46. Raportit-komponentti 53

Kuva 47. Laskutus-komponentti. 53

Kuva 48. Kaavoitusohjelman laadinta -komponentti. 54

Kuva 49. Kaavoitusohjelma-komponentti. 54

Käytetyt termit ja lyhenteet

Asynkroninen	Tässä yhteydessä asynkronisuudella tarkoitetaan ohjelman kykyä suorittaa monia tehtäviä yhtä aikaa.
DOM	Document Object Model, tapa kuvata HTML- tai XML-dokumentteja.
Funktio	Katso metodin selitys.
I/O	Tiedon siirtämistä tietokonelaitteiston komponenttien välillä.
Instanssi	Tässä yhteydessä instanssilla tarkoitetaan ohjelman luokan esiintymää, jonka avulla pystytään käyttämään sille määriteltyjä muuttujia ja metodeja.
JSON	JavaScript Object Notation, avoimen standardin tiedostomuoto, jota käytetään tiedonvälityksessä.
Komentorivikehote	Windows-käyttöjärjestelmän komentotulkki.
Komponentti	Tässä yhteydessä komponentti tarkoittaa käyttöliittymään luotua pientä osaa.
Koodinkäsittelijä	Muuttaa koodin luettavaan muotoon toiselle ohjelmalle.
Metodi	Itsenäinen ohjelman osa, joka kutsuttaessaan suorittaa tietyn toiminnon.
Muuttuja	Tässä yhteydessä muuttujalla tarkoitetaan tietovarastoa, josta voidaan joko hakea tai johon voidaan kirjoittaa tietoa.
Ohjelmointirajapinta	Määritelmä, jonka avulla eri ohjelmat pystyvät tekemään pyyntöjä ja vaihtamaan tietoja keskenään.
Parametri	Ohjelmalle käynnistyksen ja funktioille funktiokutsujen yhteydessä välitettäviä tietoja kutsutaan parametreiksi.

Sovelluskehys	Ohjelmistotuote, joka muodostaa valmiin rungon sen päälle rakennettavalle ohjelmalle.
VBA	Visual Basic for Applications, Microsoftin kehittämä ohjelmointikieli.

1 JOHDANTO

1.1 Työn tausta

Työn taustana on päivittää Seinäjoen kaupungin kaupunkisuunnittelu- ja kaavoitusyksikön käytössä olevan Kaavakortisto-ohjelman käyttö selaimeen. Tällä hetkellä Kaavakortisto-ohjelman tietokantana on Microsoft Access ja sitä käytetään VBA-ohjelmointikielellä tehdyllä käyttöliittymällä, joka vaatii Microsoft Access Runtime -ohjelman asennuksen jokaiseen tietokoneeseen, jolta Kaavakortisto-ohjelmaa halutaan käyttää. Viemällä tietokannan käytön selaimeen pystytään välttämään Runtimen asennus, joka on tuottanut välillä ongelmiakin. Samalla myös teoreettisesti saadaan Kaavakortisto-ohjelman käyttö kaikille tietokoneille, jotka ovat Seinäjoen kaupungin verkossa.

1.2 Työn tavoite

Työn tavoitteena on moderneilla ja avoimen lähdekoodin tekniikoilla luoda toimiva Kaavakortisto-ohjelma, joka toimii täysin selaimen kautta, eikä vaadi käyttäjiltä minkään ohjelman asennusta työasemalle, jotta Kaavakortisto-ohjelmaa voitaisiin käyttää. Seinäjoen kaupungin vaatimuksesta uusi tietokanta tulee tehdä SQLite-tietokannaksi.

1.3 Työn rakenne

Luvussa 2 tutustutaan vanhaan Kaavakortisto-ohjelmaan. Luvussa käydään läpi siihen käytettyjä tekniikoita ja käyttöliittymän ulkonäköä.

Luvussa 3 perehdytään tietokantojen perusteisiin ja SQLite-tietokantaan.

Luvussa 4 tutkitaan Node.js-ympäristöä, REST-ohjelmointirajapintaa ja REST-ohjelmointirajapinnan tekemiseen liittyviä tässä opinnäytetyössä käytettyjä tekniikoita.

Luvussa 5 tutustutaan ReactJS-kirjastoon ja siihen liittyviin ominaisuuksiin. Luvussa käydään läpi myös opinnäytetyössä käytettyjä ReactJS-kirjastoon liittyviä perusteita ja tekniikoita.

Luvussa 6 esitellään opinnäytetyön tuloksena syntynyttä kokonaisuutta ja sen toiminnallisuutta.

Luvussa 7 on pohdintaa opinnäytetyöstä.

1.4 Seinäjoen kaupunki

Tämän opinnäytetyön toimeksiantajana toimii Seinäjoen kaupunki. Seinäjoen kaupunki on laaja organisaatio, joka koostuu monesta toimialasta, kaupunginhallituksesta ja valtuustosta. Työntekijöitä Seinäjoen kaupungilla on noin 4000 (Seinäjoki 2016). Kuvassa 1 on esitettynä kaupunkiorganisaatio ja sen osa-alueiden keskeisempiä tehtäviä.

Kuva 1. Kaupunkiorganisaatio (Seinäjoki [Viitattu 6.2.2019]).

1.4.1 Kaupunkisuunnittelu ja kaavoitus

Tämä opinnäytetyö tehtiin Seinäjoen kaupungin kaupunkisuunnittelu ja kaavoitus -yksikölle. Sen tavoitteena on luoda mahdollisuudet muodostaa toimiva, terveellinen, viihtyisä, kulttuurisesti ja esteettisesti tasokas elinympäristö vahvan suunnittelutyön avulla. Kaavoituksella pyritään eheyttämään kaupungin yhdyskuntarakennetta ja vahvistetaan kaupunkikeskustan muodostumista. (Seinäjoki [Viitattu 5.2.2019].)

1.4.2 Kaavakortisto

Tämän opinnäytetyön pohjana on Kaavakortisto-ohjelma, jota kaupunkisuunnittelu ja kaavoitus -yksikkö käyttää kolmen kaavoitukseen liittyvän aiheen hallitsemiseen. Näitä aiheita ovat kaavoitusohjelma, kaavaprojektien hallinta ja asianhallinta.

Kaavoitusohjelman tarkoituksena on luoda kasvavan kaupungin kaupunkisuunnittelun ja maankäytön kokonaiskuva kaavoituksen tavoitteista ja työkentästä muutaman vuoden ajanjaksolle (Seinäjoki 2017).

Kaavaprojektien hallinnalla tarkoitetaan kaavaan liittyvien tärkeiden tietojen, esimerkiksi kaavaan liittyvien perustietojen, kuten kaavanumero, kaavan nimi ja aineiston vastuuhenkilö, sekä ylläpitäminen ajan tasalla. Lisäksi hallintaan kuuluu kaavaprojektin käsittelyvaiheiden tärkeiden päivämäärien ylläpito. Tärkeä päivämäärä voi olla esimerkiksi kaavan voimaantulo tai koska se on tullut vireille.

Asianhallinnalla tarkoitetaan erilaisten aineistojen hallintaa tiedostotasolla. Käytännössä tämä tarkoittaa Kaavakortisto-ohjelman auttavan tiettyyn kaavaprojektiin liittyvien tiedostojen löytämistä ja avaamista ilman että niitä täytyy etsiä verkkolevyiltä.

2 VANHA KAAVAKORTISTO

2.1 Vanha Kaavakortisto

Vanha Kaavakortisto-ohjelma koostuu kahdesta osasta. Ensimmäinen osa on Microsoft Access -tietokanta, johon kaikki tarvittavat tiedot tallennetaan. Toinen osa on Microsoft Access Runtime -käyttöliittymä, jolla tietokannan tietoja pystytään lisäämään, päivittämään ja tarpeen vaatiessa poistamaan.

2.1.1 Microsoft Access

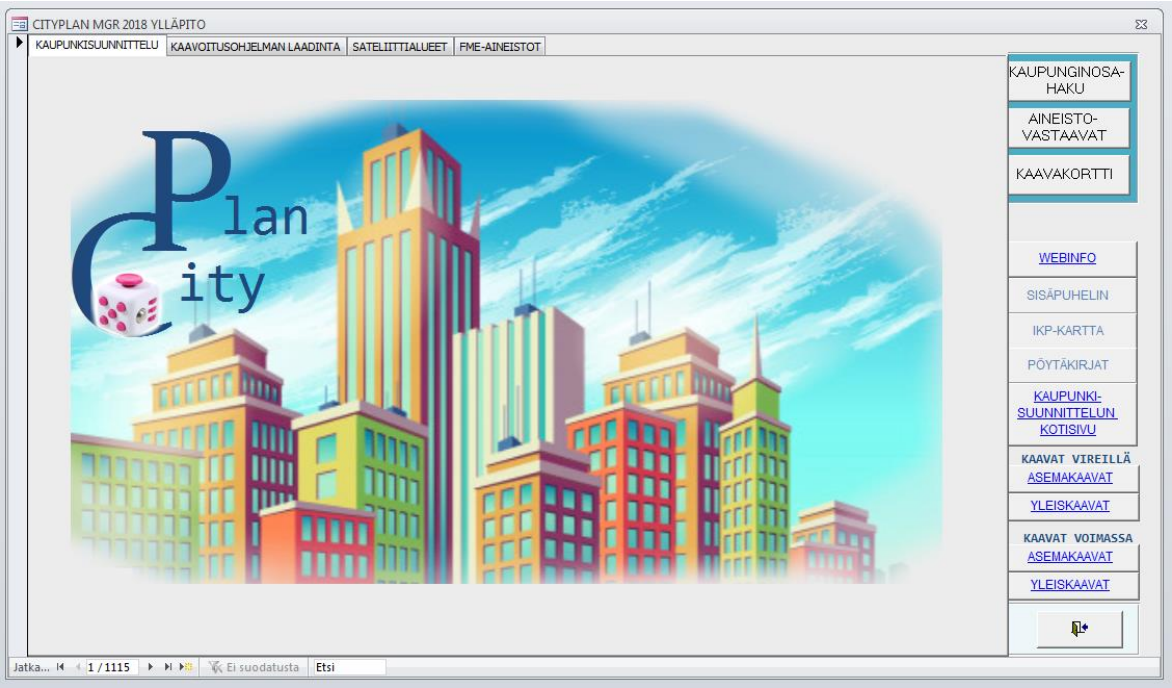
Microsoft Access on Microsoftin kehittämä relaatiotietokantojen hallintajärjestelmä. Microsoft Access on saatavilla joissain Microsoft Office -paketeissa, mutta se on myös erikseen ostettavissa. Microsoft Access -tietokannan suurin etu on sen graafinen käyttöliittymä, jonka avulla ohjelmoinnista ymmärtämätönkin henkilö pystyy luomaan tietokannan ja hakemaan tietokannasta tietoa erilaisiin käyttötarkoituksiin, kuten raportit, tai luomaan erilaisia näkymälomakkeita, joista tietokannan tietoja pystytään tarkastelemaan. Microsoft Access mahdollistaa käytön myös kehittyneemmille käyttäjille tarjoamalla tietokantojen kehittämisen VBA (Visual Basic of Applications) -ohjelmointikielellä. Microsoft Access vaatii toimiakseen maksullisen lisenssin. (Database.Guide 2016.)

2.1.2 Microsoft Access Runtime

Microsoft Access Runtime on kuin Microsoft Access, mutta siitä on otettu pois käytöstä kaikki tietokannan kehitystyökalut. Sillä pystytään siis tarkastelemaan kehittäjän luomia palveluita, kuten näkymälomakkeet ja raportit. Microsoft Access Runtime on ilmainen sovellus. Se mahdollistaa Microsoft Access -tietokannan käytön ilmaiseksi sellaisille käyttäjille, joiden ei tarvitse tehdä varsinaista tietokantakehitystä, vaan jotka tarvitsevat vain tietokannan tarjoamia tietoja tai joutuvat niitä hallitsemaan. (Pineault 2018.)

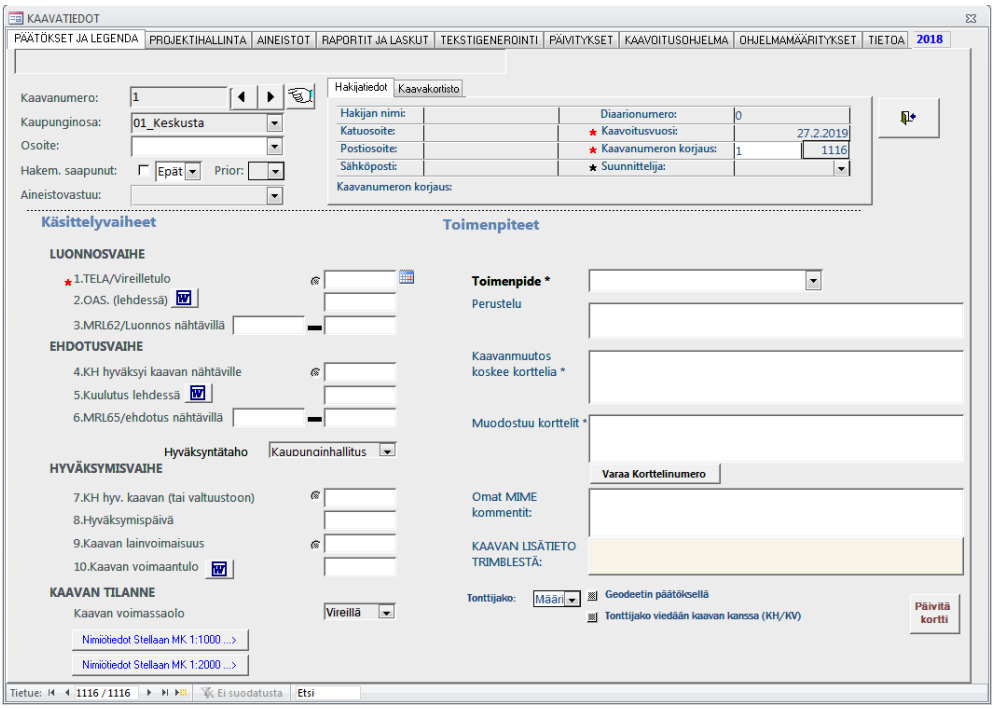
2.1.3 Vanhan kaavakortiston käyttöliittymä

Kuvassa 2 on vanhan Kaavakortisto-ohjelman aloitussivu, josta pystytään navigoimaan ympäri sovellusta.



Kuva 2. Vanhan kaavakortiston aloitussivu

Kuvassa 3 esitetään vanhassa Kaavakortisto-ohjelmassa olevaa tietojenhallintalomaketta. Eri välilehdet eivät ulkonäöllisesti eroa paljon toisistaan, mutta ne sisältävät samantyyppisiä kenttiä, joihin syötetään kaavojen tietoja. Välilehdet on pyritty jakamaan aihealueittain.



Kuva 3. Tietojenhallintalomake Kaavakortisto-ohjelmassa

2.2 Tavoite

Tavoitteena on luoda selainkäyttöliittymä, jonka kautta Seinäjoen kaupungin kaupunkisuunnittelu ja kaavoitus -yksikkö pystyy hallinnoimaan kaavoitusprojekteihin liittyviä tietoja. Toimiva kokonaisuus tulee luoda käyttäen avoimen lähdekoodin tekniikoita. Tietokannaksi vaadittiin SQLite-tietokantaa.

3 TIETOKANTA

3.1 Mikä on tietokanta?

Tietokantojen voidaan ajatella olevan loogisesti yhteenkuuluvien ja tallennettujen tietojen joukko, jonka käyttö tapahtuu tietokantakielellä, esimerkiksi SQL-kielellä. Tietokantojen tietoja voidaan hallita ohjelmistoilla, joita kutsutaan tietokantojen hallintajärjestelmiksi (Database Management System, DBMS). Hyviä esimerkkejä hallintajärjestelmistä ovat Oracle, Microsoft Access, MySQL ja Microsoft SQL Server. (Hovi, Huotari & Lahdenmäki 2005, sivu 4.)

3.2 Relaatiotietokanta

Vuonna 1970 IBM-yrityksessä töissä ollut ohjelmoija E. F. Codd kehitti relaatiotietokannan mallin. Hän ehdotti kirjoituksessaan, ”A Relational Model of Data for Large Shared Data Banks”, että siirryttäisiin pois datan tallentamisesta hierarkia- tai navigointirakenteisiin, ja siirryttäisiin järjestämään tieto tauluihin, jotka koostuvat riveistä ja kolumneista. (TechTarget 2018.)

3.2.1 Taulut

Relaatiotietokannoissa tieto järjestetään tauluihin. Taulut koostuvat riveistä ja kolumneista. Rivit tunnetaan myös tietueina (record, tuple) ja kolumnit kenttinä (field). Tietokannan taulu muistuttaa taulukkoa, mutta taulujen väliset relaatiot mahdollistavat relaatiotietokantoja toimimaan tehokkaasti datan tallentamisessa ja hakemisessa. (Hock-Chuan 2010.)

3.2.2 Pääavain

Jos tietokannan taulussa on edes kaksi identtistä riviä, saattaa tiedonhaussa syntyä epäselvyyksiä, mikä voi johtaa virhetilanteisiin. Tällaisten tilanteiden välttämiseksi

jokaisessa relaatiomallisessa taulussa jokaisen rivin tulee olla yksilöitynä. Tämä tapahtuu pääavaimella (primary key). Jokaisessa taulussa on siis yksi kolumni, jonka tarkoituksena on yksilöidä taulun jokainen rivi. Pääavaimet mahdollistavat myös taulujen välisen yhteyden. (Hock-Chuan 2010.)

Pääavaimen saa itse valita, mutta sen tulee noudattaa kahta ehtoa:

1. Sen arvon täytyy olla uniikki jokaisella rivillä.
2. Sillä on pakko olla jokin arvo, eli se ei voi olla tyhjä. (Hock-Chuan 2010.)

Pääavainta valittaessa tulisi ottaa huomioon:

- Sen pitäisi olla yksinkertainen.
- Sen arvo ei saa koskaan muuttua. Pääavainta voidaan käyttää viitteenä muissa tauluissa. Jos pääavain muuttuu, se tulee muuttaa jokaisessa sen viitteessä, tai viite katoaa.
- Yleisimpiä pääavain-malleja on automaattisesti lisääntyvä juokseva numero, joka korottaa tämän hetken suurinta numeroa yhdellä. Tämä ominaisuus on suurimmassa osassa tietokantojen hallintajärjestelmissä jo valmiina. (Hock-Chuan 2010.)

3.2.3 Viiteavain ja viittaukset

Viiteavain (foreign key) on taulussa pääavaimen tapainen konsepti. Viiteavaimella pystytään luomaan yhteyksiä taulujen välille. Viiteavain viittaa lähes poikkeuksetta toisen taulun pääavaimeen. Viiteavaimet ovat keskeisiä tekijöitä estämään vääränlaisen datan syöttämisen tauluihin. Niillä pystytään myös estämään poisto- tai päivitysoperaatiot, jotka jättäisivät rivejä orvoiksi. (Poolet 1999.)

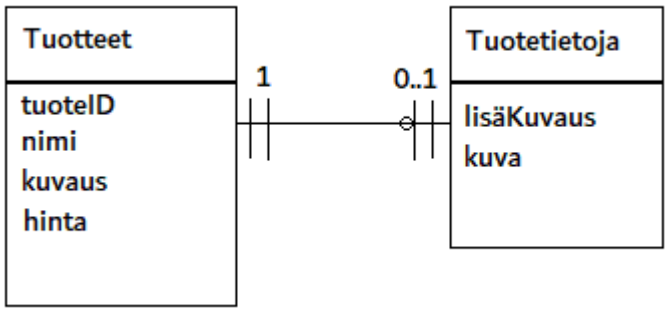
Viittauksia on relaatiotietokannoissa kolmea eri tyyppiä:

- yksi yhteen (one-to-one)
- yksi moneen (one-to-many)

- moni moneen (many-to-many). (Hock-Chuan 2010.)

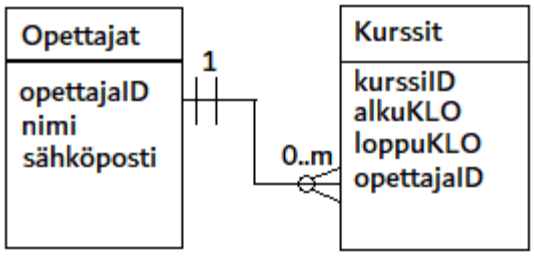
Yksi-yhteen-viittauksissa lapsitaulussa on korkeintaan yksi rivi jokaista isäntätaulun riviä kohden. Hyvä yksi-yhteen-viittauksen käyttökohde on taulun jakaminen osiin. Jotkin tietokannat saattavat rajoittaa taulun kolumnien määrää, jolloin taulun voi jakaa kahteen osaan. Toinen mahdollinen tapa käyttää yksi-yhteen-viittausta on jakaa taulu tiedon arkaluontoisuuden perusteella. Salattavat tiedot sijoitetaan suojattuun tauluun ja yleiset tiedot toiseen ei suojattuun tauluun. (Hock-Chuan 2010.)

Kuvassa 4 esitetään tuotteen tietojen jakamista kahteen eri tauluun.



Kuva 4. Yksi-yhteen-viittaus (perustuu Hock-Chuan 2010)

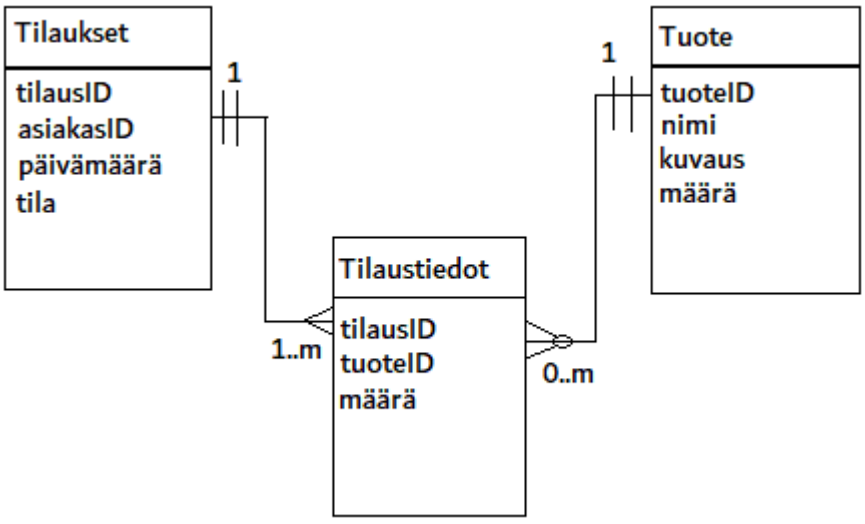
Yksi-moneen-viittauksissa isäntätaululla voi olla lapsitaulussa nollasta äärettömään asti rivejä viitteenä. Koulu-nimisessä tietokannassa voi opettajalla olla pidettävänä monia kursseja. Jokaisella kurssilla on kuitenkin vain yksi opettaja. Kuvassa 5 kuvataan tietokantaa, jossa jokaisella kurssilla on opettaja, johon pystytään viittaamaan opettajan yksilöivällä arvolla. Päinvastoin myös kaikki opettajan pitämät kurssit saadaan selville etsimällä kurssit, joissa opettajan yksilöivät arvot ovat samat. (Hock-Chuan 2010.)



Kuva 5. Yksi-moneen-viittaus (perustuu Hock-Chuan 2010)

Moni-moneen-viittauksissa käytetään kahden taulun sijaan kolmea taulua. Kolmas taulu toimii liittymätauluna kahdelle muulle taululle. Moni-moneen-viittausta voidaan kuvata asiakkaiden ja tilauksien suhteella. Yksi tilaus voi sisältää monta tuotetta. Yksi tuote voi olla monessa tilauksessa. (Hock-Chuan 2010.)

Kuvassa 6 esitetään moni-moneen-yhteyden rakennetta. Tilaustiedot-taulu toimii yhteytenä tilauksille ja tuotteille. Tilaustiedot-taulussa pääavaimena on sekä tilausID että tuoteID, jotka toimivat samalla myös viiteavaimina tilaukset- ja tuotetauluihin. (Hock-Chuan 2010.)



Kuva 6. Moni-moneen-yhteys (perustuu Hock-Chuan 2010)

3.3 SQLite

SQLite on ohjelmistokirjasto, joka tarjoaa kevyen relaatiotietokantojen hallintajärjestelmän. SQLite tunnetaan itsenäisenä ja palvelimettomana (serverless) tietokantana. Lisäksi sen käyttöönotto ei vaadi lainkaan konfigurointia. (SQLite Tutorial [Viitatu 14.3.2019].)

Useimmat tietokantojen hallintajärjestelmät sisältävät asiakas-palvelin-arkkitehtuuria, jossa ohjelmien halutessa yhteyden tietokantaan, täytyy niiden tehdä se palvelimen kautta. SQLite on palveliton, ja tämän vuoksi ohjelmat pystyvät suoraan lukemaan ja kirjoittamaan tietoja SQLite-tietokantatiedostoon. Palvelimettomuuden vuoksi SQLite-tietokantaa ei tarvitse mitenkään asentaa, ennen kuin sitä voi käyttää ja palvelinprosesseja ei tarvitse määrittää, käynnistää tai sulkea. Yksi SQLite-tietokannalle ominainen piirre on dynaamiset tietotyypit tauluille. Tämä tarkoittaa, että tiedon voi tallennettaessa asettaa mihin tahansa tauluun, riippumatta onko sen tietotyyppi esimerkiksi merkkijono tai päivämäärä. (SQLite Tutorial [Viitattu 14.3.2019].)

4 NODE.JS

4.1 Web-palvelimet ennen Node.js-ohjelmaa

Web-sovelluksissa noudatettiin ennen tarkkaa asiakas-palvelin-mallia, jossa asiakas pyytää palvelimelta resursseja, ja palvelin jakaa asiakkaalle sen pyytämät resurssit. Palvelin vastaa siis aina vain silloin, kun asiakas pyytää resursseja ja niiden lähettämisen jälkeen sulkee yhteyden. (Ofoegbu 2018.)

Asiakas-palvelin-mallin tehokkuus piilee yhteyksien sulkemisessa heti, kun pyydetyt resurssit on lähetetty. Tämä vapauttaa palvelimen mahdollisimman nopeasti uusille pyynnöille. Mutta jos palvelimelle lähetetään miljoona pyyntöä? Se suorittaa ne yksi kerrallaan aloittaen seuraavan edellisen päätyttyä, jolloin jonon perällä olevat asiakkaat joutuvat odottamaan omaa vuoroaan. Ennen tähän ongelmaan haettiin ratkaisua säikeistä. Säikeet mahdollistavat ohjelmat suorittamaan monia operaatioita samanaikaisesti. Aina kun palvelin saa uuden pyynnön, se käynnistää uuden säikeen suorittamaan tehtävää. (Ofoegbu 2018.)

Säikeiden toimintaa voidaan kuvata ravintolan toiminnalla. Jos ravintola olisi yksisäikeinen, eli siinä olisi vain yksi tarjoilija, asiakkaiden tulisi aina odottaa edellinen tilaus loppuun, ennen kuin pääsisivät itse tilaamaan. Monisäikeinen ravintola taas sisältäisi monta tarjoilijaa, mikä mahdollistaa monien asiakkaiden palvelemisen yhtä aikaa, minkä seurauksena ravintolan toiminta nopeutuu. (Ofoegbu 2018.)

Asiakas-palvelin-mallin suurin haaste on säikeiden yhtäaikainen määrä. Liian suuri määrä tarjoilijoita verottaa ravintolan tuloja. Kun palvelimeen tulee uusi pyyntö ja säie käynnistetään, se vie osan tietokoneen resursseista käyttöönsä. Suuri määrä säikeitä vie loogisesti suuren määrän resursseja. Tämän mallin suurin etu on suurta prosessointia vaativat ohjelmat. Asiakkaiden palveluun luodut säikeet vievät työtä pois pääsäikeeltä, mikä mahdollistaa ohjelman tehokkaan toiminnan. (Ofoegbu 2018.)

4.2 Node.js

Ryan Dahl esitteli Node.js-projektiaan ensimmäisen kerran yleisölle Euroopan JSConf-tapahtumassa marraskuussa 2009. Dahl oli tyytymätön Apache-palvelimien tapaan käsitellä monia samanaikaisia yhteyksiä ja halusi saada web-kehittäjille vaihtoehtoisen tekniikan käyttöön. (Chandrayan 2017.)

Node.js on avoimen lähdekoodin palvelinpuolen runtime-ympäristö, joka on rakennettu Google Chrome -selaimen V8-JavaScript-moottorin ympärille. Se mahdollistaa JavaScript-ohjelmointikielen käytön tapahtumapohjaisessa ja asynkronisessa I/O-mallissa. (TutorialsTeacher [Viitattu 27.2.2019].)

Node.js prosessoi käyttäjän pyyntöjä eri tavalla kuin perinteiset web-palvelinmallit kuten asiakas-palvelin-malli. Node.js on yksisäikeinen ja kaikki käyttäjien tekemät pyynnöt kulkevat tämän yhden säikeen kautta. Asynkroninen I/O käsittelee kaikki ohjelmaan tulevat pyynnöt. Asynkroninen tehtävien jako mahdollistaa säikeen vapauttamisen uusien pyyntöjen vastaanottamiseen ennen kuin annettu tehtävä on edes valmistunut. Vastaus palvelimelta lähetetään heti, kun asynkroninen I/O on saanut tehtävänsä valmiiksi. (TutorialsTeacher [Viitattu 29.2.2019].)

Node.js on tapahtumiin perustuva teknologia. Node.js-palvelimen käynnistyttyä se suorittaa muuttujiin ja funktioihin liittyvät alustukset, minkä jälkeen se yksinkertaisesti vain odottaa tapahtumia. Tapahtumat käsittelee tapahtumasilmukka (event loop), joka tapahtuman huomatessaan aloittaa siihen liittyvän palautusfunktion (callback function) suorittamisen. (Tutorialspoint [Viitattu 4.3.2019].)

4.3 NPM

NPM tulee sanoista Node Package Manager. NPM on internetissä sijaitseva tietovarasto, johon julkaistaan avoimen lähdekoodin Node.js-projekteja. NPM mahdollistaa myös samannimisen tietovaraston käytön pakettien asennuksessa, versionhallinnassa ja erilaisten riippuvuuksien hallinnassa. (Node.js Foundation 2011.)

NPM-ohjelmisto asentuu Node.js-asennuksen yhteydessä ja sitä käytetään komentorivikehotteelta. NPM mahdollistaa pakettien asennuksen, päivityksen ja poiston. Paketilla tarkoitetaan koodia, joka on julkaistu NPM-tietovarastoon. (Gray 2018.)

Jotta NPM-paketteja pystytään käyttämään Node.js-projekteissa, tulee projektiin luoda package-tiedosto. Tähän tiedostoon kirjataan kaikki NPM-paketit, jotka projektiin on asennettu. Lisäksi package-tiedosto sisältää jokaisesta NPM-paketista asennetun version numeron. (Gray 2018.)

4.4 REST-ohjelmointirajapinta

REST (Representational State Transfer) -arkkitehtuurin tavoitteena on standardisoida tietokoneiden välisiä yhteyksiä verkossa helpottaen niiden keskinäistä kommunikointia. (Codecademy [Viitattu 13.3.2019]).

REST-ohjelmointirajapinta mahdollistaa palvelimen ja asiakkaan olevan itsenäisiä osia, joiden ei tarvitse tietää toisistaan mitään. Tämän vuoksi koodin muuttaminen asiakas osassa ei vaikuta palvelimen toimintaan mitenkään. Tärkeintä on tietää, missä muodossa viestit tulee lähettää, että toinen osapuoli pystyy sen ymmärtämään. Erottamalla asiakaspuolen ongelmat palvelinpuolen ongelmista, pystytään luomaan joustavampia ja skaalautuvampia alustoja. Käyttämällä REST-ohjelmointirajapintaa saadaan useille eri asiakkaille suoritettua täsmälleen samat toiminnot, ja saadaan ne vastaanottamaan täsmälleen samat viestit. (Codecademy [Viitattu 13.3.2019].)

4.4.1 Kommunikointi

REST-ohjelmointirajapinnoissa asiakkaan ja palvelimen välinen kommunikointi tapahtuu asiakkaan lähettäessä pyyntöjä palvelimelle. Asiakas lähettää pyyntöjä palvelimelle, ja palvelin lähettää resursseja takaisin palvelimelle tai mahdollisesti muuttaa jo olemassa olevia resursseja. Pyynnöt koostuvat yleensä seuraavista osista:

- HTTP-avainsana, jolla määritetään, millainen operaatio suoritetaan.

- Tunniste (header), jossa on tietoja asiakkaan tekemästä pyynnöstä.
- Osoitepolku, johon asiakas ottaa yhteyttä saadakseen resursseja käyttöönsä.
- Vapaaehtoinen viesti, johon voidaan sisällyttää dataa. (Codecademy [Viitattu 13.3.2019].)

Neljä yleisintä HTTP-avainsanaa, joita käytetään resurssien hallintaan, ovat:

- GET – Pyydetään resursseja palvelimelta.
- POST – Luodaan uusia resursseja.
- PUT – Päivitetään olemassa olevia resursseja.
- DELETE – Poistetaan olemassa olevia resursseja. (Codecademy [Viitattu 13.3.2019].)

Tunnisteessa asiakas kertoo palvelimelle, millaisessa muodossa se pystyy vastaanottamaan dataa. Tunnistetta voidaan kutsua myös Accept-kentäksi, sillä se varmistaa, että palvelin ei lähetä asiakkaalle dataa, jota se ei pysty ymmärtämään tai käsittelemään. (Codecademy [Viitattu 13.3.2019].)

Pyynnöillä tulee olla osoitepolku, josta haluttu operaatio pystytään suorittamaan. Osoitepolut olisi hyvä pitää mahdollisimman selkeinä ja kuvaavina. Osoitepolkuun on mahdollista syöttää resursseja yksilöiviä tietoja. (Codecademy [Viitattu 13.3.2019].)

4.5 Web-palvelimen luonti tietokantayhteydellä

Tässä osiossa esitellään Express-sovelluskehys, jonka avulla pystytään luomaan middleware-ohjelmistojen avulla toimiva web-palvelin, joka pystyy lähettämään ja vastaanottamaan dataa. Lisäksi tutustutaan SQLite3-moduuliin, jonka avulla saadaan web-palvelimelle toimiva tietokantayhteys.

4.5.1 Express

Express on suosituin sovelluskehys, jota käytetään Node.js-web-sovellusten tekemiseen. Sitä käytetään pohjana monille vastaavanlaisille sovelluskehyksille. Express on varustettu seuraavilla ominaisuuksilla:

- Käsittelijöiden kirjoittaminen eri HTTP-avainsanoille eri osoitepoluissa.
- Yleisten web-sovelluksiin liittyvien asetusten määrittäminen.
- Ulkopuolisten kirjastojen (middleware) hyödyntäminen pyyntöjen datan käsittelyssä. (MDN web docs [Viitattu 19.3.2019].)

Express asennetaan projektiin kirjoittamalla komentorivikehotteeseen kuvassa 7 näkyvä komento:

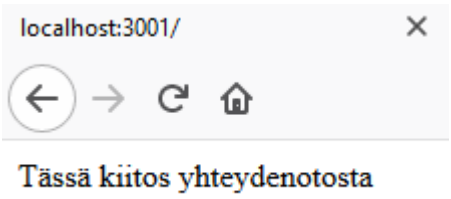
```
C:\>npm install express --save
```

Kuva 7. Express-sovelluskehyksen asennuskomento.

Jotta Express-sovelluskehystä voidaan käyttää projektissa, se täytyy ottaa käyttöön. Tämä tapahtuu sijoittamalla se muuttujaan. Tällä muuttujalla voidaan nyt luoda käsittelijöitä kuuntelemaan yhteydenottoja annettuihin osoitepolkuihin. Express-sovellus vaatii vielä määrittelyn portille, josta se vastaanottaa yhteyksiä. Alla olevassa koodissa (kuva 8) on esimerkki Express-sovelluksesta, joka palauttaa oletusreittiin yhteyttä ottaville asiakkaille Tässä vastaus -tekstin (kuva 9).

```
JS testi.js
1   const express = require('express');
2
3   const app = express();
4
5   app.get('/', function( req, res){
6       res.send('Tässä kiitos yhteydenotosta!');
7   });
8
9   app.listen(3001, function(){
10      console.log('Listening on port 3001');
11  });
12
```

Kuva 8. Express-esimerkkisovellus.



Kuva 9. Vastaus oletusreitistä.

4.5.2 SQLite3

SQLite3 on moduuli, jonka avulla pystytään ottamaan yhteys SQLite-tietokantaan Node.js-sovelluksesta. SQLite3-moduuli asennetaan projektiin syöttämällä komentorivikehotteeseen kuvassa 10 esiintyvä komento.

```
C:\>npm install sqlite3 --save
```

Kuva 10. SQLite3-moduulin asennuskomento.

Seuraava koodinpätkä esittää SQLite3-moduulin peruskäyttöä (kuva 11). Käyttö vaatii ensin käyttöönoton. Tämän jälkeen luodaan muuttuja, johon luodaan tietokannasta instanssi. Parametriksi tähän instanssiin annetaan polku, jossa tietokantatiedosto fyysisesti sijaitsee. Jotta tietokannasta saadaan valittua vain halutut tiedot, täytyy luoda SQL-kyselykielellä SQL-lause. Lopuksi tietokantayhteys on aina hyvä sulkea.

testi.js

```
const sqlite3 = require('sqlite3').verbose();

let db = new sqlite3.Database('D:/Tietokannat/Kaavoitus_ta.db');

let sql = "SELECT Kaavannimi FROM Kaavatiedot WHERE Kaavatunnus = 99999";

db.all(sql, [], (err, rows) => {
    if (err){
        throw err;
    }
    rows.forEach((row) => {
        console.log(row);
    })
});

db.close();
```

Kuva 11. SQLite-tietokantahakusovelluksen esimerkki.

Opinnäytetyöhön tehdystä tietokannasta kuvan 11 mukainen tietokantahaku palauttaa kuvan 12 mukaisen objektin.

```
{ Kaavannimi: 'Testikaava' }
```

Kuva 12. Esimerkki tietokantahaun tuloksesta.

4.5.3 Middleware

Middleware-ohjelmistoilla tarkoitetaan web-sovelluksissa erilaisia kirjastoja, joita käytetään asiakkaiden yhdistämiseen palvelimen tarjoamiin resursseihin. Näiden ohjelmistojen etuna on tehokkaampien ja nopeampien yhteyksien luonti palveluihin ja resurssien muuttaminen käyttäjille erilaisten käyttäjältä saatujen tietojen, kuten sijainnin, avulla. Middleware-ohjelmistot voivat myös auttaa ohjelmia skaalautumaan paremmin. (Rouse [Viitattu 23.3.2019].)

Tähän opinnäytetyöhön tehtyyn web-palvelimeen liitettiin kaksi middleware-ohjelmistoa, jotka ovat body-parser ja CORS.

HTTP-pyyntöjen yhteydessä lähetettävää dataa ei lähetetä palvelimelle yhtenä isona tiedon palana, vaan se lähetetään pienissä osissa, jotka kohteeseen saavuttuaan kootaan takaisin yhteen. Body-parser on middleware-ohjelmisto, joka suorittaa tämän yhteen koonnin automaattisesti pyynnön datalle. (Upadhyay 2017).

Body-parser asennetaan syöttämällä kuvan 13 mukainen komento komentorivikehotteeseen.

```
C:\>npm install body-parser
```

Kuva 13. Body-parser-asennuskomento.

Kuvassa 14 otetaan body-parser käyttöön Express-sovelluskehykseen. Body-parser.json()-metodi määrittää, mihin muotoon selaimesta tuleva data halutaan.

testi.js

```
1  const express = require('express');
2
3  const app = express();
4
5  const bodyParser = require('body-parser')
6  app.use(bodyParser.json())
```

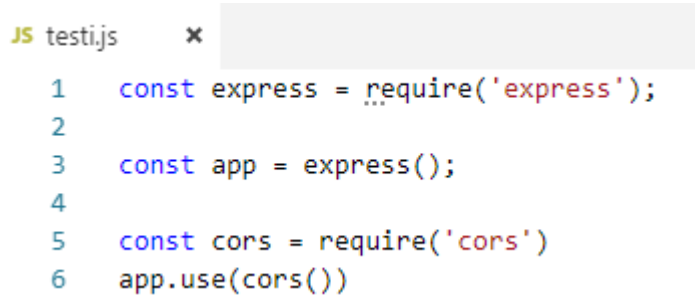
Kuva 14. Body-parser-ohjelmiston käyttöönotto.

CORS (cross-origin resource sharing) mahdollistaa pyyntöjen vastaanottamisen ulkopuolisista alkuperistä. Pyynnön alkuperä koostuu kolmesta asiasta, jotka ovat protokolla, isäntä (host) ja porttinumero. Jos kaikki pyynnön alkuperän tiedot vastaavat palvelimen alkuperän tietoja, on pyyntö samaa alkuperää (same-origin). Jos tiedot ovat eriävät, on kyseessä ulkopuolinen alkuperä (cross-origin). (Codecademy [Viitattu 23.3.2019].)
CORS asennetaan kuvan 15 mukaisella komennolla.

```
C:\>npm install cors
```

Kuva 15. CORS-asennuskomento.

Kuvassa 16 on CORS-ohjelmiston käyttöönotto ja sen lisääminen Express-sovelluskehykseen.



Kuva 16. CORS-ohjelmiston käyttöönotto.

4.5.4 Resurssien päivitys tietokantaan

Tähän opinnäytetyöhön tehdyn web-palvelimen tärkeimpiä tehtäviä on resurssien hakeminen tietokannasta, resurssien lähettäminen selaimelle ja resurssien tietojen päivittäminen tietokantaan. Kuvassa 17 on node.js-ohjelma, joka suorittaa resurssin päivityksen tietokantaan selaimesta lähetettävällä tiedolla. Ohjelman toiminta:

- Aluksi otetaan käyttöön kaikki tarpeelliset ohjelmistot (rivit 1–8).
- Luodaan Express-käsittelijä, joka vastaanottaa HTTP-avainsanalla PUT tulevia pyyntöjä osoitepolkuun ”/API/Kaavannimi” (rivi 10).
- Sijoitetaan body-parser-ohjelmiston luoma JSON-muotoinen data *body*-nimiseen muuttujaan. Tulostetaan konsoliin näkyville muuttuja *body* (rivit 11–13).
- Luodaan instanssi tietokannasta, SQL-kysely, joka päivittää kaavan nimen, ja muuttuja, jonka arvoksi annetaan pyynnössä annettu uusi nimi kaavalle (rivit 15–17).
- Suoritetaan SQL-kysely tietokantaan, jos ilmenee virhetilanteita, keskeytetään ohjelman suoritus. Jos virhetilanteita ei synny, lähetetään pyynnön tekijälle viesti päivityksen onnistumisesta (rivit 19–27).
- Luodaan kuuntelija kuuntelemaan yhteyksiä porttiin 3001 (rivit 30–32).

testi1.js

```
const express = require('express');
const app = express();
const cors = require('cors');
const bodyParser = require('body-parser');
const sqlite3 = require('sqlite3').verbose();

app.use(cors());
app.use(bodyParser.json());

app.put('/API/Kaavannimi', (req, res) => {
    let body = req.body;

    console.log(body);

    let db = new sqlite3.Database('D:/Tietokannat/Kaavoitus_ta.db');
    let sql = `UPDATE Kaavatiedot SET Kaavannimi = ? WHERE Kaavatunnus = 99999`;
    let updateData = [body.Kaavannimi];

    db.run(sql, updateData, (err) => {
        if (err) {
            throw err;
        }


        res.send('Tieto päivitetty!');
    });
    db.close();
});

app.listen(3001, () => {
    console.log("Kuunnellaan porttia 3001");
});
```

Kuva 17. Esimerkkiohjelma resurssin päivityksestä.

Kun kuvan 18 mukaiselle palvelimelle lähetetään pyynnön mukana kuvan 18 mukainen JSON-objekti, saadaan tietokantaan päivitettyä kyseisen kaavan nimeksi *testikaavaamuutettu*. Ohjelman suoriuduttua pystytään kuvan 12 mukaisella tietokantahaulla tarkistamaan, onko tieto saatu oikeaan paikkaan tietokannassa. Kyseinen tietokantahaku palauttaa nyt kuvan 19 mukaisen objektin.

```
{ Kaavannimi: 'testikaavaamuutettu' }
```

Kuva 18. Esimerkki palvelimelle lähettävästä resurssista.

```
{ Kaavannimi: 'testikaavaamuutettu' }
```

Kuva 19. Kaavan nimi päivityksen jälkeen.

5 REACTJS

5.1 ReactJS yleisesti

ReactJS on Facebookin ohjelmistokehittäjän Jordan Walken vuonna 2011 kehittämä JavaScript-kirjasto, jonka perimmäinen tarkoitus on luoda nopeita ja tehokkaita käyttöliittymiä. ReactJS yhdistää JavaScriptin nopeuden ja uuden tavan tehdä verkkosivujen renderöintiä. Tämä yhdistelmä tekee internetsivuista sekä dynaamisia että hyvin responsiivisia käyttäjien syötteille. (AltexSoft 2018.)

ReactJS perustuu komponentteihin, joilla pystytään jakamaan käyttöliittymä itsenäisiin ja uudelleenkäytettäviin osiin. Tämä mahdollistaa jokaisen käyttöliittymän osan tarkastelun ja käytön erillään toisistaan. Komponenttien voidaan ajatella olevan JavaScript-funktioita, jotka ottavat vastaan mielivaltaista dataa ja palauttavat React-elementtejä esittämään, mitä näytöllä pitäisi nähdä. (React [Viitattu 10.2.2019].)

Virtual DOM on ReactJS-kirjastolle ominainen rakenne, jolla on pyritty optimoimaan DOM-manipulaatiosta sellainen, että se tekisi mahdollisimman vähän manipulaatiota pitääkseen sivun ajantasaisena. Kun sovelluksessa tapahtuu muutoksia, Virtual DOM osaa päivittää vain muuttuneet osat. Wheeler (2014) käyttää metaforisena esimerkkinä Virtual DOM -rakenteen toiminnasta ihmisen luomista, jossa jokainen ruumiinosa olisi oma kokonaisuutensa (ReactJS-kirjaston tapauksessa komponentti). Muuttaessa ihmisestä esimerkiksi käsien kokoa tehokkain tapa on muuttaa vain käsiä, eikä luoda ihmistä alusta erikokoisilla käsillä. Virtual DOM vastaa vain käsien muuttamisesta ja perinteinen DOM-manipulaatio vastaa koko ihmisen uudelleen luomisesta uusilla käsillä.

ReactJS-kirjastolle ominaista on myös datan käyttö kahdella eri tavalla, tilalla (state) ja ominaisuuksilla (properties). Tila on komponentin objekti, joka sisältää tietoa. Tätä tietoa voidaan muuttaa komponentin elinkaaren aikana tai funktioiden avulla. Ominaisuudet taas ovat komponentin objekti, jonka tiedolla komponentin käytöstä voidaan muuttaa haluttuun suuntaan. Vaikka niiden käyttötarkoitus kuulostaa samalta, ne eivät ole identtisiä. Tärkein ero näiden kahden välillä on niiden muutettavuus. Ominaisuudet ovat muuttumatonta tietoa, jonka komponentti perii yleensä toiselta

ylemmän tason komponentilta. Tilan tieto taasen on yleensä muuttuvaa ja sitä usein muutetaan tapahtumakäsittelijöiden avulla. Hyvä esimerkki tapahtumakäsittelijästä on ruudulla olevan napin painallus. (GeeksforGeeks [Viitattu 10.2.2019].)

5.2 Komponentit

Nykyisin monet modernit JavaScript-sovelluskehykset (framework) ovat komponenttipohjaisia, niin myös ReactJS. ReactJS-kirjaston komponentit voidaan kuvailla funktioiksi, joita määrittävät niiden tila ja ominaisuudet. ReactJS käsittää kahdenlaisia komponentteja, jotka tunnetaan parhaiten nimillä funktionaali- ja luokkakomponentti. (Chinda [Viitattu 12.2.2019]).

Suurin ero näillä komponenteilla on funktionaalisen komponentin yksinkertaisuus. Se on pelkkä JavaScript-funktio, joka vastaanottaa ominaisuuksia, ja palauttaa React-elementin. Funktionaalisilla komponenteilla ei myöskään ole tilaa, tästä syystä niitä kutsutaan usein myös tilattomiksi funktionaalikomponenteiksi (stateless functional component). Tilattomuuden vuoksi funktionaalikomponenteilla ei myöskään ole elinkaarimetodeja. Funktionaalikomponentit ovat hyödyllisiä, kun tehdään komponentteja, joiden tarkoitus on vain esittää tietoa, jota ei tarvitse itse komponentissa muuttaa. Muissa tapauksissa suositellaan käyttämään luokkakomponentteja. (Jöch 2018.)

5.3 Virtual DOM

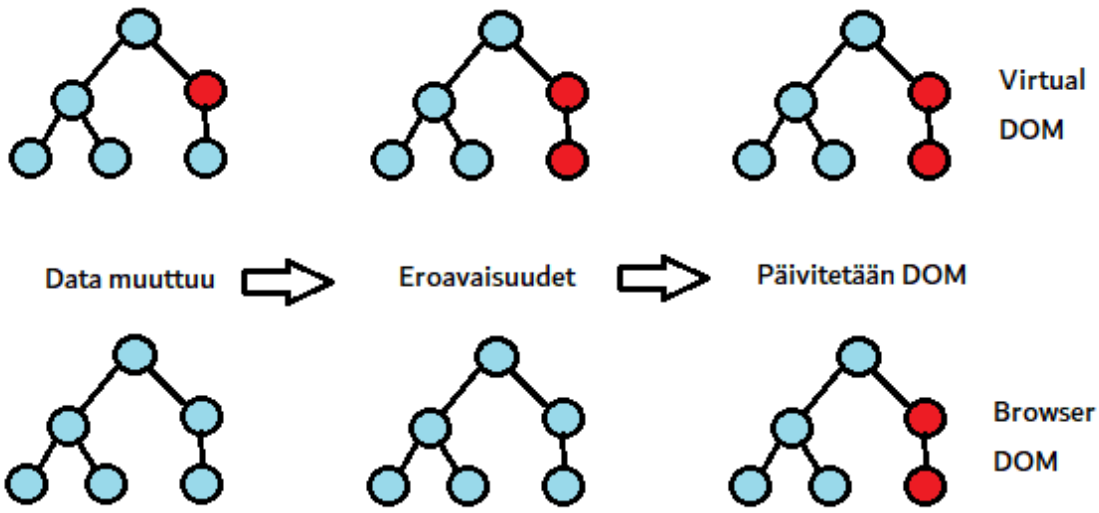
Erilaisten DOM-manipulaatio ratkaisuiden ilmestyttyä on alettu perinteistä DOM-manipulaatiota ja sen nopeutta kyseenalaistamaan. Itse perinteinen DOM-manipulaatio ei ole hidas toimenpide, mutta siitä seuraava sivun uudelleen rakentaminen sen näytöllä esittämiseksi on aikaa vievää. Ratkaisuja kehittäessä suurin tavoite on minimalisoida tarvittavien DOM-muutoksien määrä. ReactJS lähestyy tätä ongelmaa Virtual DOM nimisellä ratkaisulla. (Minnick 2016.)

DOM-rakenne sisältää kaikki käyttöliittymän HTML-elementit ja näin myös niiden ominaisuudet ja tiedot. Virtual DOM on kopio tästä puusta. Kun jonkin elementin

data muuttuu, ReactJS luo uuden Virtual DOM -rakenteen. Varsinaisen DOM-rakenteen päivitys on kolmivaiheinen prosessi ReactJS-kirjastossa:

1. Kun jotakin muuttuu, luodaan uusi Virtual DOM -rakenne.
2. ReactJS laskee uuden ja vanhan Virtual DOM -rakenteen väliset eroavaisuudet.
3. Varsinaiseen DOM-rakenteeseen päivitetään ainoastaan muuttunut tieto. (Minnick 2016.)

Virtual DOM tuo myös uudenlaisen näkökulman itse käyttöliittymän ohjelmoimiseen. Koska se on ohjelmoijan ulottumattomissa, ei hänen tarvitse huolehtia varsinaisesta DOM-manipulaatiosta, vaan hän voi keskittyä vain tekemään käyttöliittymästä halutun näköistä. ReactJS hoitaa Virtual DOM -rakenteen avulla muutokset varsinaiseen DOM-rakenteeseen. (Minnick 2016.)



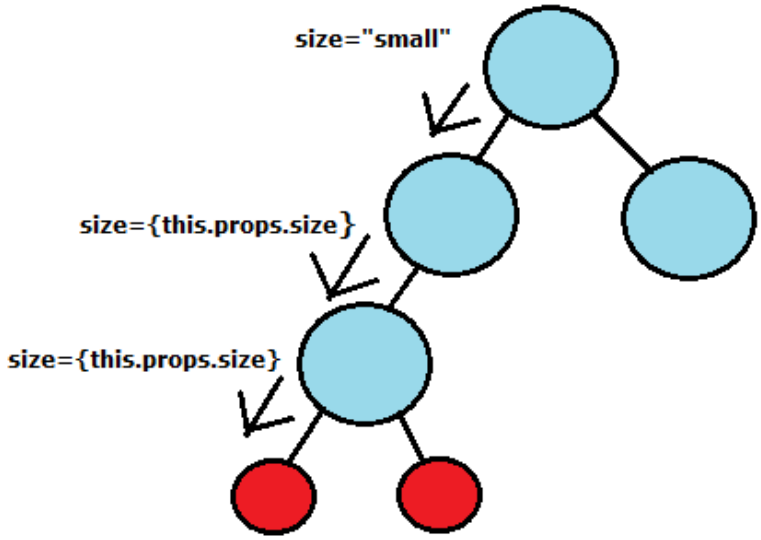
Kuva 20. Virtual DOM-rakenteen toimintaperiaate (perustuu Hamedani 2018)

Kuvassa 20 havainnollistetaan Virtual DOM -rakenteen toimintaa. Kuvassa esitetään myös Virtual DOM- ja DOM-rakenteita toiminnan eri vaiheissa. Punaiset solmut

kertovat, mikä Virtual DOM -rakenteessa on muuttunut. Nykyisen ja edellisen Virtual DOM -rakenteen eroavaisuudet lasketaan ja lopuksi ne päivitetään varsinaiseen DOM-rakenteeseen.

5.4 Properties

Yksi tapa siirtää dataa komponenttien välillä on ominaisuudet (properties, lyhyemmin props). Ominaisuudet ovat muuttumatonta tietoa. Komponentit saavat ominaisuudet isäntäkomponentilta. ReactJS-kirjaston tietovirta on yksisuuntaista, sillä se liikkuu aina isäntäkomponentilta lapsikomponentille. Ominaisuudet luovat uuden ulottuvuuden komponentin uudelleen käytölle. Useassa paikassa esiintyvä tietty komponentti on ominaisuuksien avulla mahdollista saada näyttämään erilaiselta antamalla sille eri ominaisuudet. (Ravichandran 2018.)



Kuva 21. Propertiesin toimintamalli (perustuu Chinnathambi 2016)

Kuva 21 havainnollistaa ominaisuuksien toimintaa ja esittää, kuinka alikomponentit saavat tietoa isäntäkomponenteillaan. Kuvassa mallina on komponentin Koko-tieto, joka halutaan siirtää alimmalle lapsikomponentille.

5.5 State

ReactJS-kirjastossa tilalla (state) tarkoitetaan objektia, jolla pystytään kuvaamaan sovelluksen osat, joissa voi tapahtua muutoksia. Jokainen komponentti voi hallita omaa tilaansa. Tila-objektilla pystytään hallitsemaan sovelluksen interaktiivisuutta ja ulkonäköä. Tämä tapahtuu luomalla sovellukseen tapahtumakäsittelijöitä, jotka muuttavat tila-objektin dataa. (Ceddia 2016).

Kun tilaa muutetaan, sovelluksen tulisi päivittyä vastaamaan uutta tilaa. ReactJS ei turvaudu jatkuvaan muutosten kyselyyn tai seuraamaan tapahtumia. ReactJS päivittää sovellusta vain silloin, kun sitä käsketään. Tämä käskytys tapahtuu komponenteissa setState-metodilla. (Ceddia 2016).

5.6 Lifecycle

Yksi ReactJS-kirjaston tärkeimpiä ominaisuuksia on komponentin elinkaari (lifecycle). Komponentin elinkaari antaa mahdollisuuden tehdä asioita, kun se lisätään DOM-rakenteeseen, kun sitä päivitetään DOM-rakenteeseen ja kun se poistetaan DOM-rakenteesta. (Ighodaro 2018.)

Kun komponentti luodaan ja syötetään DOM-rakenteeseen, tapahtuu niin sanottu kiinnitys (mounting). Kiinnityksessä ReactJS-komponenteilla on käytettävissä neljä elinkaarimetodia:

1. constructor(): constructor()-metodia käytetään määrittämään komponentin ensimmäinen tila. (Lathiya [Viitattu 14.2.2019].)

2. componentWillMount(): componentWillMount()-metodia kutsutaan juuri ennen kuin komponentti kiinnitetään ja render()-metodia kutsutaan. Tällä metodilla ei ole monia käyttötarkoituksia. Käyttötarkoituksia rajoittaa metodin sijainti elinkaarella. componentWillMount()-metodissa voitaisiin tehdä komponenttimäärittelyjä, mutta niitä tehdään jo constructor()-metodissa. Lisäksi DOM-rakenteeseen ei ole vielä tullut mitään, sillä vasta render()-metodissa

tapahtuu varsinainen lisäys. Yksi käyttötarkoitus voisi olla hakea tietoa ulkoisista lähteistä, mutta tätä ei suositella sen epävarmuuden vuoksi. (Ighodaro 2018.)

3. render(): render()-metodilla syötetään komponentti varsinaiseen DOM-rakenteeseen. render()-metodin tulee aina palauttaa jotain. Jos se halutaan jättää tyhjäksi, täytyy sen palauttaa null-arvo. render()-metodin tulisi olla puhdas, eli se ei saa muuttaa komponentin tilaa. Lisäksi sen täytyy palauttaa aina sama lopputulos, kun sitä kutsutaan. (Ighodaro 2018.)

4. componentDidMount(): componentDidMount()-metodi on käytettävissä komponentin kiinnityttyä ja render()-metodin valmistuttua. Sitä kutsutaan vain kerran elinkaaren aikana, ja se merkitsee komponentin ja alikomponenttien onnistunutta renderöintia. Tässä metodissa voidaan tehdä kaikki ohjelman vaatimat asetukset DOM-rakenteeseen. (Ighodaro 2018.)

Kiinnittymisen jälkeen komponenttien tila voi muuttua. Joskus ominaisuudet saattavat muuttua, jolloin tarvitaan uudelleen renderöinti. Päivitykseen liittyvät elinkaarimetodit antavat kehittäjälle mahdollisuuden hallita koska ja miten päivityksiä tehdään. ReactJS-kirjastossa on viisi päivitykseen liittyvää elinkaarimetodia:

1. componentWillReceiveProps(): Joskus alikomponentin propsit voivat olla yhteydessä isäntäkomponentin tilaan. Tämän vuoksi voi syntyä tilanne, jolloin komponentti saa uudet propsit isäntäkomponentin päivittyessä. componentWillReceiveProps()-metodia kutsutaan, ennen kuin komponentti tekee mitään uusilla propseilla. Metodi ottaa argumentiksi uudet propsit. Tämä mahdollistaa uusien ja vanhojen propsien tarkistelun yhtä aikaa. (Ighodaro 2018.)

2. shouldComponentUpdate(): shouldComponentUpdate()-metodia kutsutaan juuri ennen kuin komponentti uudelleen renderöityy tilan muuttuessa tai saadessaan uudet propsit. Metodi saa argumentikseen kaksi argumenttia, uudet propsit ja uuden tilan. Tällä metodilla pystytään vähentämään tarpeettomia uudelleen renderöintejä vertaamalla uusia ja vanhoja propseja ja tiloja. (Ighodaro 2018.)

3. componentWillUpdate(): componentWillUpdate()-metodia käytetään suorittamaan valmisteluja ennen uudelleen renderöinnin tapahtumista. Yleisimpiä käyttökohteita on käsitellä asioita ReactJS-arkkitehtuurin ulkopuolella. Tällaisia asioita voi olla vuorovaikutus ohjelmointirajapinnan kanssa tai selainikkunan koon tarkistaminen. (Ighodaro 2018.)

4. render(): Toimii kuten aiemmin mainittu (Ighodaro 2018).

5. componentDidUpdate(): componentDidUpdate()-metodia kutsutaan, kun renderöity HTML on saatu ladattua. Se saa argumentikseen edelliset propsit ja tilan. Tässä metodissa suositellaan tehtävän käsittelyt ympäristöihin, kuten selain ja HTTP-pyynnöt, jotka eivät ole osa ReactJS-kirjastoa. Huomioon pitää ottaa uusien ja vanhojen propsien ja tilojen vertailu turhien verkkopyyntöjen välttämiseksi. (Ighodaro 2018.)

Kaikki komponentit eivät elä ikuisesti DOM-rakenteessa. Syystä tai toisesta niitä joudutaan välillä poistamaan. Tätä poistamista kutsutaan irrottamiseksi (unmounting). Irrottamisessa kutsutaan vain yhtä metodia, joka on componentWillUnmount(). Tätä metodia kutsutaan juuri ennen kuin komponentti poistetaan DOM-rakenteesta. Tässä metodissa on hyvä tehdä tarvittavat siivoukset, kuten verkkopyyntöjen katkaiseminen ja tapahtumakäsittelijöiden poisto. (Ighodaro 2018.)

Komponenteilla on olemassa myös componentDidCatch()-metodi. Tällä metodilla pystytään hallitsemaan virhetilanteita, joita voi syntyä renderöinnissä, elinkaarimetodeissa tai constructor-metodeissa. Metodilla pystytään estämään tilanteet, joissa lapsielementin virhetilanne voi rikkoa koko sovelluksen. Tämä metodi huomaa virheet vain lapsielementeissä, eikä itse siinä komponentissa, jossa se määritetään. componentDidCatch()-metodi saa kaksi parametria, itse virheilmoituksen ja objektin, jossa on komponenttipinon informaatio. Näillä parametreilla pystytään muodostamaan käyttäjälle viesti virhetilanteesta, sen sijaan että virhetilanne rikkoisi koko sovelluksen. (Ighodaro 2018.)

5.7 ReactJS-sovelluksen luonti

Tässä osiossa käydään läpi ReactJS-sovelluksen aloittaminen ja komponentit. Lisäksi osiossa tutustutaan ReactJS-sovellusten komponenttien väliseen tiedon välitykseen. Lopuksi käydään läpi HTTP-pyyntöjen luominen Axios-ohjelmiston avulla.

5.7.1 Aloitus

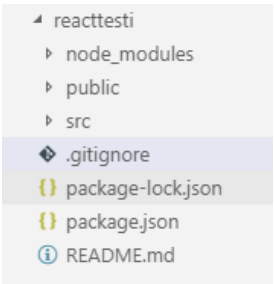
Yksinkertaisin tapa aloittaa ReactJS-sovellus on käyttää create-react-app-työkalua. Tämä työkalu luo turvallisen kehitysympäristön, jossa on helppo opetella ReactJS-kirjaston käyttöä tai aloittaa täysin uuden ReactJS-sovelluksen luonti. (React [Viitattu 23.3.2019].)

Create-react-app-työkalua käytetään syöttämällä komentorivikehotteelle kuvan 22 mukainen komento, jossa sille myös annetaan nimi. Komento luo annetulla nimellä varustetun kansion, johon avautuu kansiorakenne, joka sisältää kaikki tarpeelliset tiedostot, jota ReactJS-kehitysympäristö vaatii. Kehitysympäristö käynnistetään siirtymällä komentorivikehotteessa luotuun kansioon ja syöttämällä siihen komento *npm start*.

```
C:\>npx create-react-app reacttesti
```

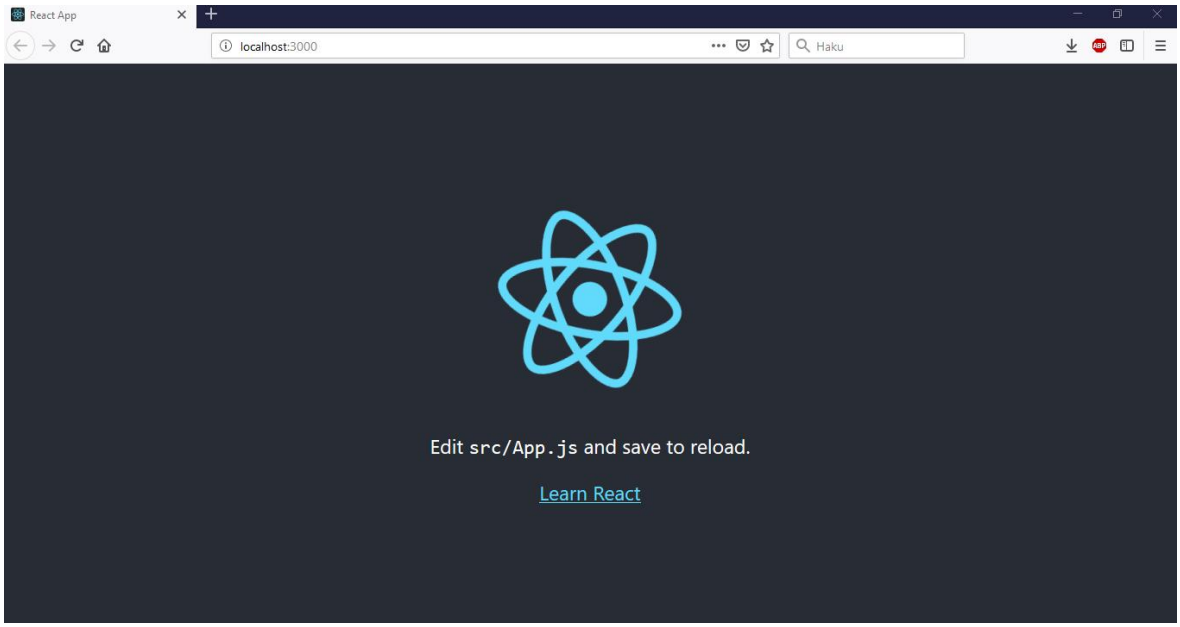
Kuva 22. Create-react-app-komento.

Kuvassa 23 nähdään työkalun luoma kansiorakenne.



Kuva 23. Uuden projektin kansiorakenne.

Kuvassa 24 on työkalun luoman kehitysympäristön oletusnäkymä.



Kuva 24. Uuden projektin oletusnäkymä.

5.7.2 JSX

JSX on XML- ja HTML-tyyppinen syntaksi, jota voidaan käyttää ReactJS-kirjastossa komponenttien luomiseen. JSX mahdollistaa XML- ja HTML-tyyppisen tekstin yhdistämisen JavaScript-ohjelmointikieleen. Sitä kuuluu käyttää yhdessä jonkin koodinkäsittelijän (preprocessor) kanssa, mikä muuttaa sen selaimelle ymmärrettävään muotoon. (Lindley [Viitattu 26.3.2019].)

Kuvassa 25 on esimerkki JSX-koodista.

```
const App = () => {
  return (
    <div>
      <p>Tekstiä</p>
      <p>Lisää tekstiä</p>
      <p>Kolmas rivi tekstiä</p>
    </div>
  )
}
```

Kuva 25. JSX-esimerkki.

5.7.3 Komponentit

Komponentteja on ReactJS-kirjastossa kahdenlaisia, funktionaali- ja luokkakomponentteja. Alla olevissa kuvissa 26 ja 27 on mallit samasta komponentista funktionaali- ja luokkamuodossa.

App.js

```
1   import React from 'react';
2
3   const App = () => {
4     return (
5       <h1>Ensimmäinen komponenttini</h1>
6     )
7   }
8
9   export default App;
```

Kuva 26. Funktionaalikomponentti.

App.js

```
1   import React, { Component } from 'react';
2
3   class App extends Component{
4
5     render(){
6       return(
7         <h1>Ensimmäinen komponenttini</h1>
8       )
9     }
10  }
11
12  export default App;
```

Kuva 27. Luokkakomponentti.

5.7.4 Properties

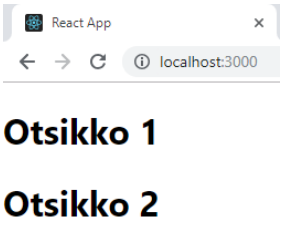
Ominaisuuksien (properties) avulla pystytään siirtämään dataa komponentilta toiselle. Kuvassa 28 on malli komponentista, joka antaa ominaisuuksia alikomponentilleen.

App.js

```
1   import React, { Component } from 'react';
2
3   const Otsikko = (props) => {
4     return (
5       <h1>{props.nimi}</h1>
6     )
7   }
8
9   const App = () => {
10    return (
11      <div>
12        <Otsikko nimi="Otsikko 1"></Otsikko>
13        <Otsikko nimi="Otsikko 2"></Otsikko>
14      </div>
15    )
16  }
17
18  export default App;
```

Kuva 28. Properties-esimerkki.

- Rivillä 3 luodaan Otsikko-komponentti, jolle annetaan parametriksi props-muuttuja. Komponentti palauttaa h1-elementin, jonka arvoksi tulee props-muuttujan Nimi-tieto.
- Rivillä 9 luodaan App-komponentti, joka palauttaa div-elementin, jossa on kaksi Otsikko-komponenttia. Näille komponenteille annetaan Nimi-tieto, jolla ne pystytään erottamaan toisistaan.
- Rivillä 18 määritellään, mikä komponentti palautetaan, kun App.js-tiedostoon viitataan toisessa tiedostossa.
- Kuvassa 29 nähdään, miten annetut Nimi-tiedot vaikuttavat komponentin ulkonäköön.



Kuva 29. Ominaisuuksien vaikutus komponentteihin.

5.7.5 State

ReactJS-kirjastossa tilalla (state) on mahdollista muuttaa komponentissa olevia tietoja. Kuvassa 30 on tilallinen komponentti, joka nappia painamalla kasvattaa tila-objektin Luku-tietoa yhdellä.

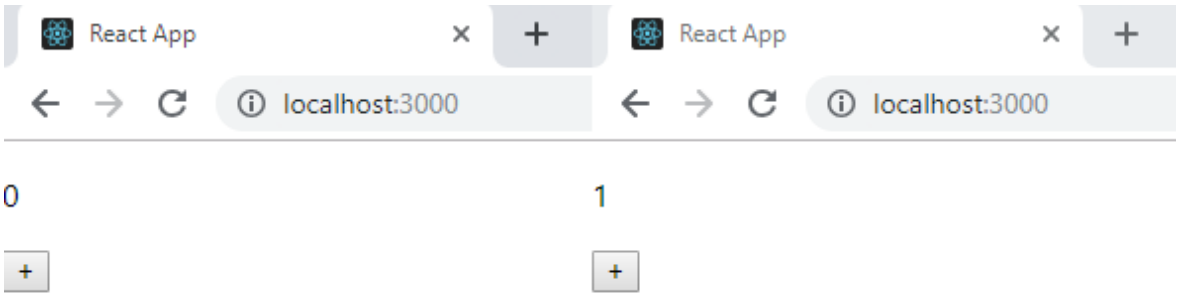
App.js

```
1   import React, { Component } from 'react';
2
3   class App extends Component {
4       state = {
5         Luku: 0
6       }
7
8     addToLuku = () => {
9       this.setState({ Luku: this.state.Luku + 1 });
10    }
11
12    render() {
13      return (
14        <div>
15          <p>{this.state.Luku}</p>
16          <button onClick={this.addToLuku}>+</button>
17        </div>
18      )
19    }
20  }
21
22  export default App;
```

Kuva 30. Yksinkertainen tilallinen komponentti.

- Rivillä 4 luodaan state-objekti ja syötetään sinne Luku-tieto, jonka arvoksi annetaan 0.
- Rivillä 8 luodaan yksinkertainen funktio, joka kasvattaa state-objektin Luku-tietoa yhdellä. Jotta tietoa voi muuttaa state-objektissa, tulee kutsua setState()-funktiota. Tämä funktio renderöi komponentin vastaamaan uutta tilaa.
- Rivillä 12 kutsutaan render()-metodia, joka palauttaa div-elementin, jossa on p-elementti, jonka arvona on state-objektin Luku-tieto. Lisäksi div-elementissä on button-elementti, jolle annettu onClick-tieto kutsuu addToLuku-funktiota, joka kasvattaa Luku-tietoa yhdellä.

- Kuvassa 31 nähdään vasemmalla puolella alkutilanne ja oikealla puolella tilanne, kun nappia on painettu kerran.



Kuva 31. Tilallinen komponentti selaimessa.

5.7.6 Axios

Axios-ohjelmisto on promise-pohjainen HTTP-asiakas (HTTP client), joka toimii sekä selaimessa että Node.js-ympäristössä. Yksinkertaisuudessaan Axios tarjoaa ohjelmointirajapinnan HTTP-pyyntöjen käsittelemiseen. (Bernardes 2015.)

Axios asennetaan NPM-ohjelmiston avulla komentokehotteen kautta (kuva 32).

```
C:\reacttesti>npm install axios --save
```

Kuva 32. Axios-asennuskomento.

Axios otetaan käyttöön ReactJS-projektissa kuvan 33 mukaisesti.

```
import axios from 'axios';
```

Kuva 33. Axios-ohjelmiston käyttöönotto ReactJS-projektissa.

Kuvassa 34 on esitetty yksinkertaisen Axios-pyynnön lähettävän ReactJS-sovelluksen malli:

- Luodaan getKaavannimi()-metodi, jossa tehdään Axios-kutsu, jolle asetetaan HTTP-avainsana ja osoitepolku. Näiden tietojen avulla web-palvelin osaa käsitellä halutun pyynnön. then()-metodissa käsitellään web-palvelimelta tuleva vastaus ja tässä tapauksessa kirjoitetaan se konsoliin. (Rivit 10–18)

- Asetetaan luotu metodi kutsuttavaksi componentDidMount()-elinkaarimetodissa. Aina kun App-komponentti kiinnittyy (mount), lähetetään palvelimelle axios-pyyntö. (Rivit 6–8)

App.js

```
1  import React, { Component } from 'react';
2  import axios from 'axios';
3
4  class App extends Component {
5
6    componentDidMount() {
7      this.getKaavannimi();
8    }
9
10   getKaavannimi() {
11     axios({
12       method: 'get',
13       url: 'http://localhost:3001/API/Kaavannimi',
14     })
15       .then(response => {
16         console.log(response);
17       })
18   }
19
20   render() {
21     return (
22       <div>Testi</div>
23     )
24   }
25 }
26
27 export default App;
```

Kuva 34. Yksinkertainen axios-pyyntö.

Web-palvelin palauttaa objektin, jossa on tietoja pyynnöstä ja sen onnistumisesta. Tärkeimpänä objektissa on kuitenkin data-objekti, johon on sijoitettu pyyntöön liittyvät tiedot. Kuvan 34 mukaisessa axios-pyynnössä vastaanotetaan data-objektissa tietokantaan luodun testikaavan nimi (kuva 35).

Kuva 35. Web-palvelimen palauttama objekti.

6 TYÖN ETENEMINEN

Opinnäytetyössä pyrittiin luomaan Seinäjoen kaupungille Kaavakortisto-ohjelma, joka toimii selaimen kautta. Vanha versio Kaavakortisto-ohjelmasta toimii Microsoft Access Runtime -käyttöliittymällä. Selaimeen vienti mahdollistaa Kaavakortisto-ohjelman käyttämisen ilman Microsoft Access Runtime -sovelluksen asentamista, mikä on aiheuttanut välillä ongelmia Seinäjoen kaupungilla.

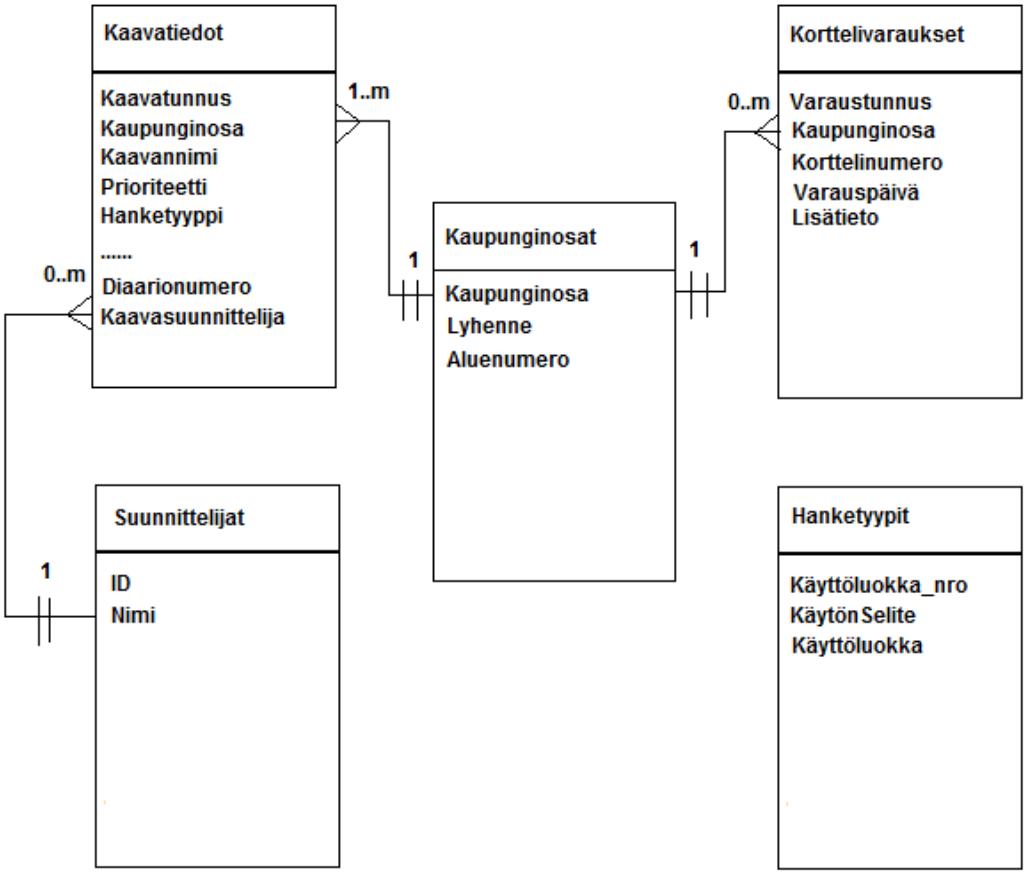
Kaavakortisto-ohjelman vienti selaimeen vaatii kolme kokonaisuutta: tietokannan, web-palvelimen ja käyttöliittymän. Tietokannaksi vaadittiin Seinäjoen kaupungilta SQLite-tietokantaa. Web-palvelin tehtiin Node.JS-tekniikalla ja käyttöliittymä tehtiin ReactJS-kirjastolla. Käyttöliittymässä pyrittiin jäljittelemään vanhan käyttöliittymän mallia.

6.1 Tietokanta

Seinäjoen kaupungin käyttöön tuleva tietokanta tehtiin vanhan Microsoft Access -tietokannan pohjalta. Vanhassa tietokannassa on yhteensä 15 taulua, joista viisi on relaatiossa keskenään. Loppujen taulujen tehtävänä on tuoda Microsoft Access Runtime -käyttöliittymälle erilaisiin kenttiin vaihtoehtoja käytettäväksi. Uutta tietokantaa suunnitellessa tehtiin kartoitus siitä, mitkä taulut tulevat uudessa sovelluksessa tarpeellisiksi. Tämän kartoituksen avulla pystyttiin luopumaan kymmenestä taulusta, joten uusi tietokanta koostuu viidestä taulusta, joista neljä on relaatiossa. Viides taulu luotiin antamaan vaihtoehtoja käyttöliittymällä olevaan valintalaatikkoon. Vaihtoehdot tehtiin tietokantaan helpottamaan niiden mahdollista lisäämistä.

Kuvassa 36 on kuvaus syntyneestä tietokannasta. Tärkein taulu tietokannassa on Kaavatiedot. Kaavatiedot-taulussa on kaikki mahdolliset tiedot kaavoista. Suunnittelijat-taululla pyritään pitämään Kaavatiedot-taulun Kaavasuunnittelija-kolumnin tieto ehyenä, sallimalla siihen vain arvoja, jotka löytyvät Suunnittelijat-taulusta. Kaupunginosat-taululla on samanlainen rooli kuin Suunnittelijat-taululla, mutta sen lisäksi Kaupunginosat-taulu on relaatiossa Korttelivaraukset-taulun kanssa, missä jokaiselle kaupunginosalle luodaan korttelit. Hanketyypit-taululla luodaan vaihtoehtoja

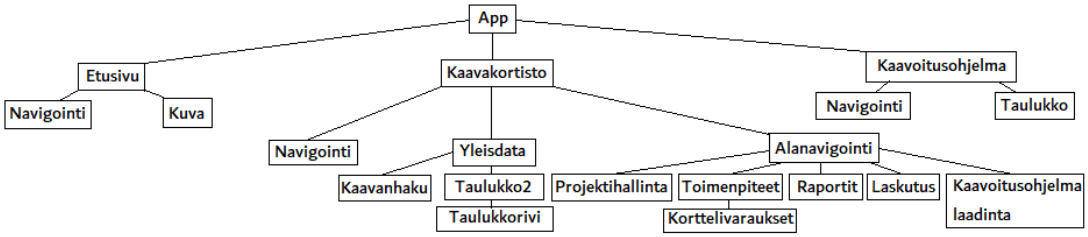
käyttöliittymään Kaavatiedot-taulun Hanketyyppi-kolumnin tiedon valintaan. Hanketyypit-taululle ei tehty relaatiota Kaavatiedot-tauluun, sillä vanhassa tietokannassa sitä ei ollut. Tämän relaation puuttumisen seurauksena Hanketyyppi-kolumnissa on tietylle hanketyypille monia erilaisia variaatioita, mikä vaikeuttaa tiedon täsmällisen oikeellisuuden hallintaa.



Kuva 36. Valmistuneen tietokannan kuvaus

6.2 Web-palvelin ja käyttöliittymä

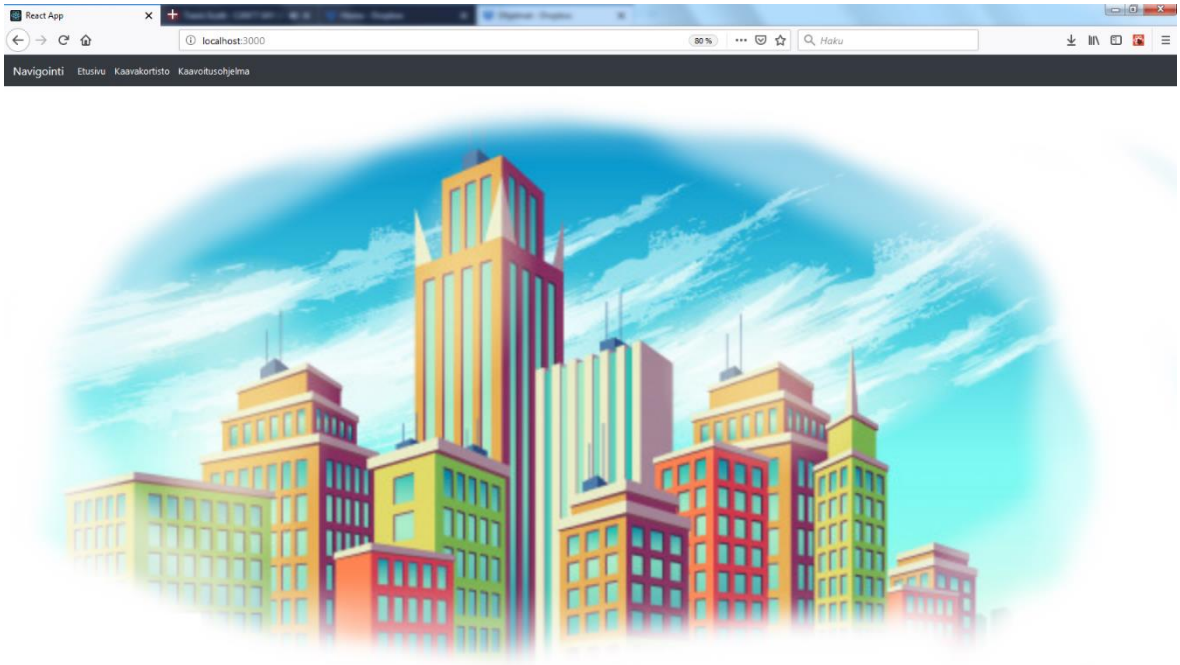
Kuvassa 37 on käyttöliittymään suunniteltu komponenttirakenne. Pääkomponenttina toimii App-komponentti. Käyttöliittymä on jaettu kolmeen komponenttiin: Etusivu, Kaavakortisto ja Kaavoitusohjelma.



Kuva 37 Käyttöliittymän komponenttirakenne.

6.2.1 Etusivu

Käyttöliittymän Etusivu-komponentti on erittäin yksinkertainen. Se sisältää ainoastaan Navigointi-komponentin ja Kuva-komponentin. Navigointi-komponentti sisältää ohjelman sisäisen navigoinnin, jonka avulla pystytään liikkumaan eri sivujen välillä. Kuva-komponentti sisältää kuvan vanhan Kaavakortisto-ohjelman etusivulta. Etusivu-komponentissa ei tehdä yhtään pyyntöä web-palvelimelle.



Kuva 38. Etusivu-komponentti.

6.2.2 Kaavakortisto

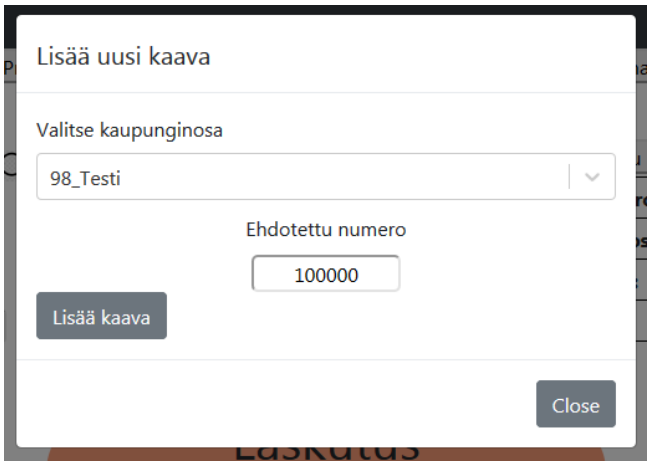
Kaavakortisto-komponentti on sovelluksen tärkein osa. Sen kautta hallitaan kaavojen tietoja. Kaavakortisto-komponentti jakautuu kolmeen alakomponenttiin: Navigointi, Yleisdata ja Alanavigointi. Kun Kaavakortisto-komponentti kiinnittyy, lähetetään palvelimelle pyyntö resursseista. Palvelimelta pyydetään kaavaan liittyvät tiedot, kaavoittajien nimet Aineistovastuu-valikkoa varten, kaavatunnukset Kaavanhaku-valikolle ja hanketyypit Kaavoitusohjelma-komponentin Kaavahanke-valikkoon.

Yleisdata-komponentille ja Alanavigointi-komponentin alikomponenteille on tehty jokaiselle oma tallennuspainike. Tästä painikkeesta tallennetaan tietokantaan vain komponenttiin liittyvät tiedot. Tämä ominaisuus tehtiin minimoimaan tietokannan käyttöaikaa, sillä SQLite-tietokanta ei tue monen käyttäjän yhtäaikaista palvelua.

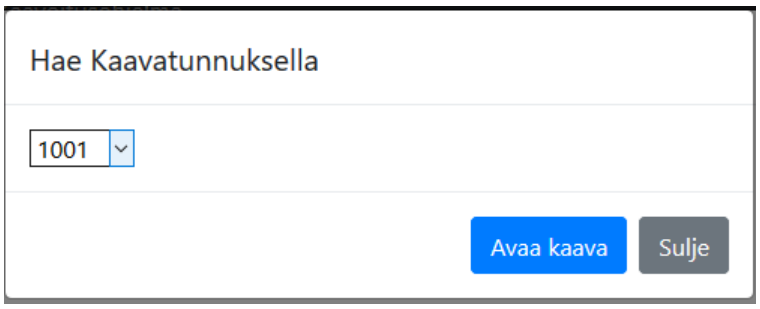
Navigointi-komponentti sisältää ohjelman sisäisen navigoinnin. Yleisdata-komponentti sisältää nimensä mukaan yleistä tietoa kaavoista, kuten kaavanumeron ja kaupunginosan. Lisäksi siinä on painike uuden kaavan lisäämiselle. Uusi kaava lisätään valitsemalla kaupunginosa ja painamalla painiketta tekstillä ”Lisää kaava”. Ehdotettu numero lasketaan automaattisesti, jotta käyttäjä ei pysty lisäämään tietokantaan kaavaa numerolla, joka on jo olemassa (kuva 40).

Muokattavaa kaavaa pystytään vaihtamaan suurennuslasin muotoisesta painikkeesta. Painikkeesta aukeaa kuvan 41 mukainen ponnahdusikkuna, jossa on lista kaikista kaavoista. Avaa kaava -painikkeesta lähetetään palvelimelle pyyntö valitun kaavan tietojen saamiseksi.

Kuva 39. Yleisdata-komponentti.



Kuva 40. Kaavan lisäys.

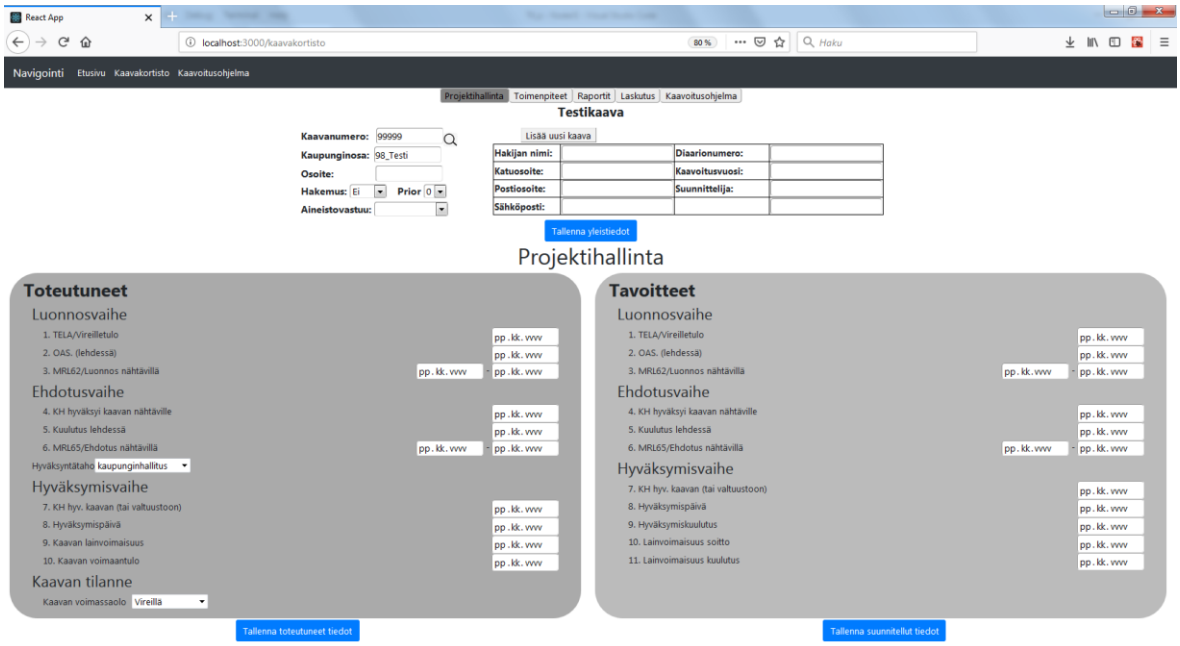


Kuva 41. Kaavan haku -ponnahdusikkuna.

Alanavigointi-komponentti jakaantuu viiteen alikomponenttiin: Projektihallinta, Toimenpiteet, Raportit, Laskutus ja Kaavoitusohjelma. Alanavigointi-komponentilla pystytään liikkumaan eri alikomponenttien välillä.

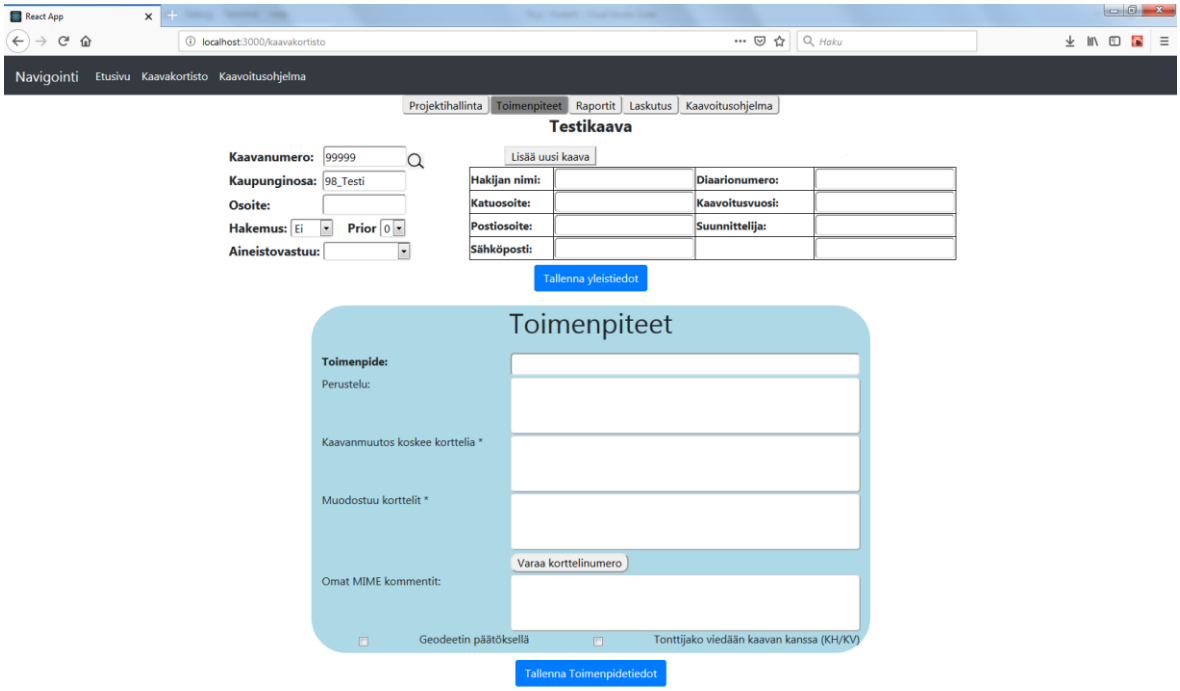
Kuva 42. Alanavigointi-komponentti.

Projektihallinta-komponentissa syötetään kaavoille niiden eri vaiheisiin liittyviä päivämääriä. Kuvassa 43 esitetään Projektihallinta-komponenttia. Päivämäärät on jaettu kahteen osaan: tavoitteet ja toteutuneet. Tavoitteisiin kirjataan tavoitepäivämäärät eri vaiheille. Toteutuneisiin taas luonnollisesti kirjataan kaavalle toteutuneet päivämäärät eri vaiheille. Tämän avulla pystytään seuraamaan projektien etenemistä.



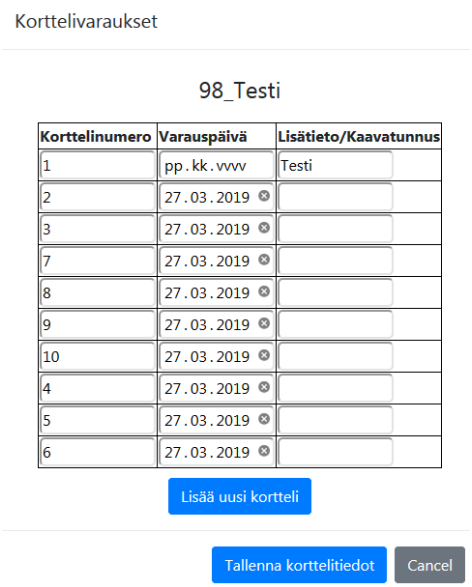
Kuva 43. Projektihallinta-komponentti.

Toimenpiteet-komponentissa kirjataan ylös kaavalle tehtäviä toimenpiteitä. Tässä komponentissa hallitaan kortteleita ja niiden varauksia. Kuvassa 44 esitetään Toimenpiteet-komponentti.



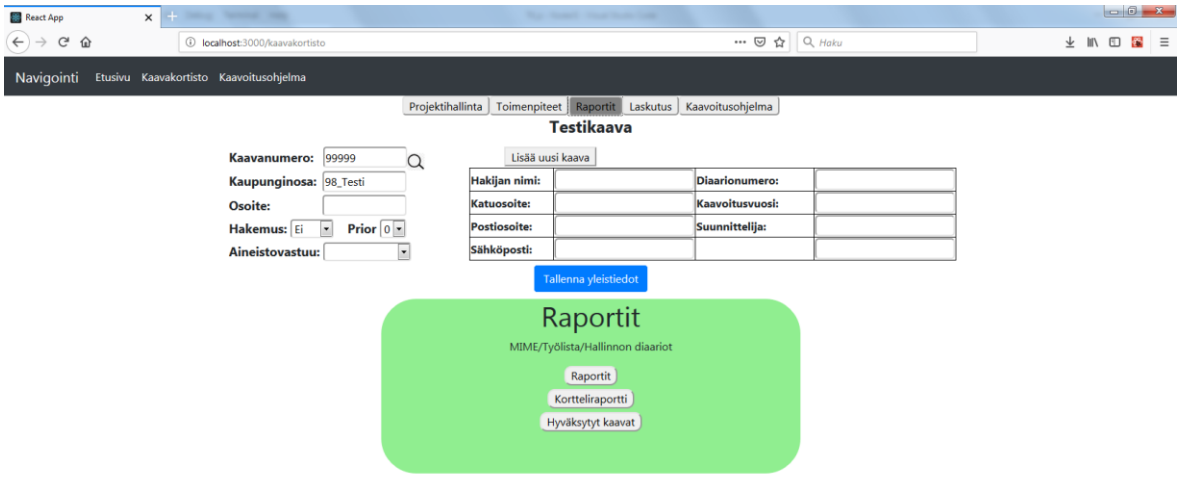
Kuva 44. Toimenpiteet-komponentti.

Toimenpiteet-komponenttiin kuuluu korttelivarausten hallinta. Tähän hallintaan päästään painamalla Varaa korttelinumero -painiketta. Painikkeesta aukeaa kuvan 45 mukainen Korttelivaraukset-komponentti. Komponentin auetessa tehdään palvelimelle pyyntö, joka hakee tietokannasta valittuna olevan kaavan kaupunginosaan kuuluvat olemassa olevat korttelit. Tietokannasta tulevia korttelitietoja pystytään muokkaamaan tässä komponentissa. Lisää uusi kortteli -painikkeesta pystytään lisäämään uusi kortteli. Uudelle korttelille lasketaan automaattisesti uusi numero, jota ei vielä ole olemassa. Varauspäiväksi tulee automaattisesti lisäyshetken päivämäärä. Tietokantaan korttelitiedot tallennetaan vasta, kun Tallenna korttelitiedot -painiketta painetaan.



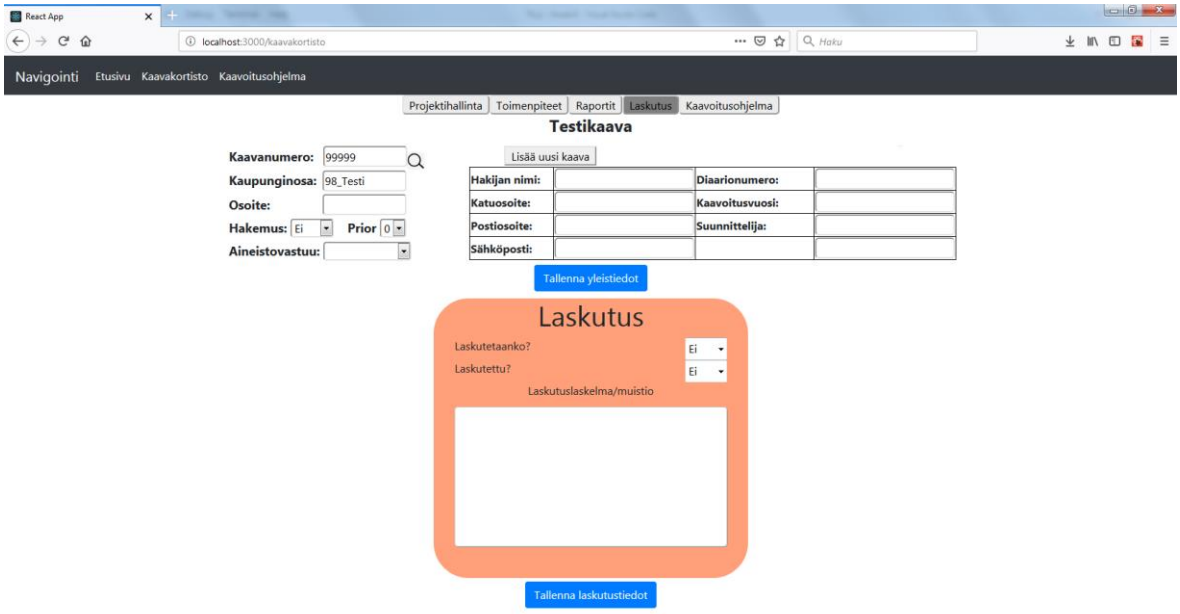
Kuva 45. Korttelivaraukset-komponentti.

Raportit-komponentti mahdollistaa tietokannassa olevasta datasta erilaisten raporttien tekemisen. Tässä komponentissa ei vielä ole toiminnallisuutta, sillä Seinäjoen kaupungilla ei vielä valittu, millä tekniikalla raportit muodostetaan.



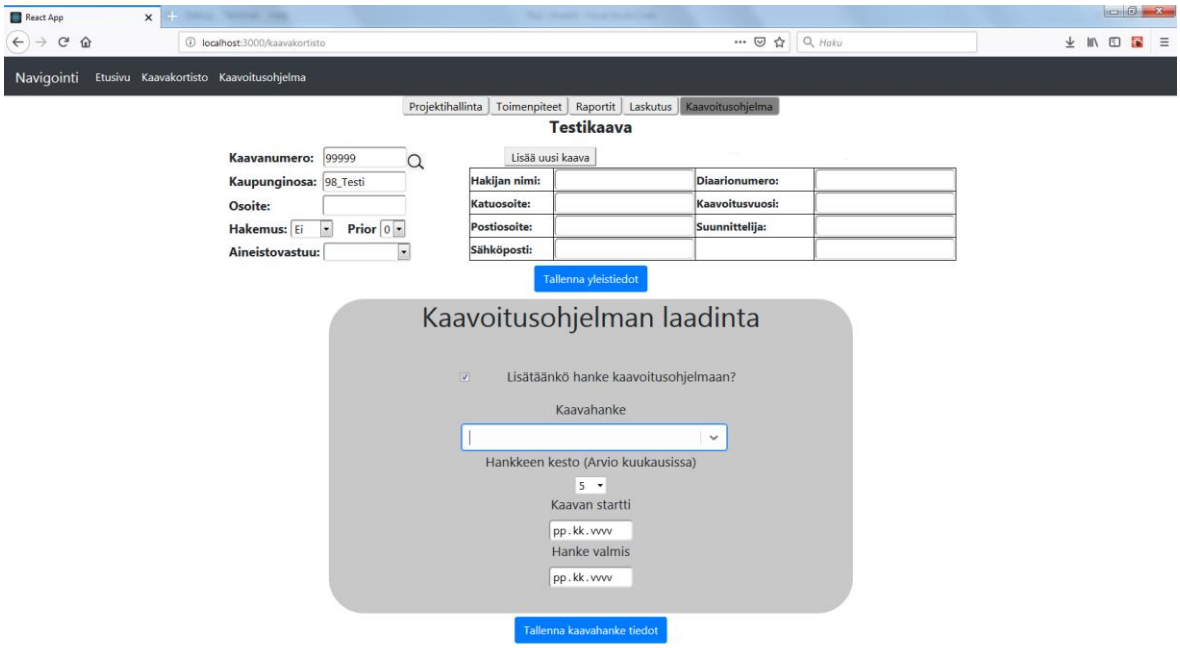
Kuva 46. Raportit-komponentti

Laskutus-komponentti on suoraviivainen. Se sisältää tiedon, onko kaavasta tarpeellista laskuttaa. Lisäksi komponentti sisältää tiedon siitä, onko laskutus jo tehty, ja onko Laskutus-kentässä itsekirjoitettua tietoa laskutukseen liittyen. Kuvassa 47 on esitetty Laskutus-komponentti.



Kuva 47. Laskutus-komponentti.

Kaavoitusohjelma-komponentissa päätetään, lisätäänkö kaava kaavoitusohjelmaan. Jos kaava lisätään kaavoitusohjelmaan, tulee sille valita hanketyyppi Kaavahanke-valikosta. Hankkeen kesto tulee myös arvioida. Kaavoitusohjelmaan lisääminen tarkoittaa myös sitä, että kaava tulee näkyville Kaavoitusohjelma-komponentissa olevaan taulukkoon.



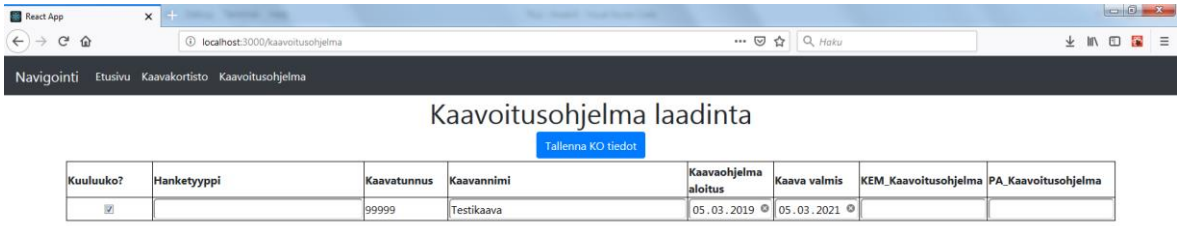
Kuva 48. Kaavoitusohjelman laadinta -komponentti.

6.3 Kaavoitusohjelma

Kaavoitusohjelma-komponentti koostuu kahdesta alikomponentista, Navigointi- ja Taulukko-komponentista. Tämän komponentin kiinnittyessä tehdään web-palvelimelle pyyntö, jolla saadaan kaikki kaavat resursseiksi, mitkä kuuluvat kaavoitusohjelmaan.

Kaavoitusohjelma-komponentin tarkoituksena on helpottaa kaavoitusohjelmassa olevien kaavojen hallintaa. Siinä pystytään muokkaamaan kaavoitusohjelmassa olevien kaavojen tietoja tai jopa poistamaan ne kaavoitusohjelmasta. Tiedot lähetetään tallennettavaksi vasta Tallenna KO tiedot -painiketta painaessa.

Kuvassa 49 esitetään Kaavoitusohjelma-komponentin ulkonäköä.



Kuva 49. Kaavoitusohjelma-komponentti.

7 POHDINTA

Tässä opinnäytetyössä luotiin Seinäjoen kaupungille Kaavakortisto-ohjelma, joka toimii selaimessa. Ohjelma luotiin olemassa olevan Kaavakortisto-ohjelman pohjalta. Vanhan Kaavakortisto-ohjelman tietokantana on Microsoft Access -tietokanta ja tietoja tietokantaan muokataan Microsoft Access Runtime -käyttöliittymän kautta. Seinäjoen kaupungin ainoa vaatimus oli SQLite-tietokannan käyttö tietokantana.

Tässä opinnäytetyössä tutustuttiin tietokannan perusteisiin, Node.js-ympäristöön ja kuinka sillä tehdään yksinkertainen REST-ohjelmointirajapinta, joka hakee tietoa tietokannasta, sekä ReactJS-kirjastoon, joka on JavaScript-ohjelmointikielellä toteutettu kirjasto, jota käytetään pääasiassa komponenttimallisien käyttöliittymien luomiseen.

Suurimmaksi haasteeksi tässä opinnäytetyössä syntyi käyttöliittymän tekeminen tehokkaaksi työkaluksi käyttäjälle. Käyttöliittymän tulisi olla helppokäyttöinen ja ominaisuuksien tulisi olla vain muutaman napin painalluksen päässä. Esimerkiksi kaavan vaihtaminen Kaavakortisto-komponentissa ei tunnu kovin tehokkaalta. Toisena ongelmana näkisin suunnittelun olleen kohtalaista. Kohtalaisesta suunnittelusta hyvä esimerkki on käyttöliittymän komponenttirakenne, jossa täysin identtinen Navigointi-komponentti asetettiin kolmeen komponenttiin. Tietokannan suunnittelussa olisi voinut Kaavatiedot-taulun jakaa useampaan osaan yhden massiivisen taulun sijaan.

Tätä opinnäytetyötä voisi kehittää tekemällä käyttöliittymästä käyttäjäystävällisemmän ja lisäämällä kaavojen vaihtoon tehokkaammat haut, esimerkiksi mahdollistamalla kaupunginosittaisen kaavaluettelon tai antamalla käyttäjille mahdollisuuden lisätä listaan omia työnalla olevia kaavoja.

Opin tästä opinnäytetyöstä paljon ReactJS-kirjaston käyttöä ja käyttöliittymien suunnittelua ja luomista. Uskon opinnäytetyön myös kasvattaneen taitoja tietokantojen hallinnassa, Node.js-ympäristön käytössä ja REST-ohjelmointirajapintojen luomisessa.

LÄHTEET

Altexsoft. 2018. the Good and the Bad of ReactJS and React Native. [Verkkosivu]. Altexsoft. [Viitattu 30.1.2019]. Saatavana: https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-reactjs-and-react-native/

Bernardes, Marlon. 2015. How to Use Axios as Your http Client. [Verkkosivu]. CodeHeaven. [Viitattu 13.3.2019]. Saatavana: http://codeheaven.io/how-to-use-axios-as-your-http-client/

Ceddia, D. 2016. A Visual Guide to State in React. [Verkkosivu]. [Viitattu 14.2.2019]. Saatavana: https://daveceddia.com/visual-guide-to-state-in-react/

Chandrayan, P. 2017. All About Node.Js You Wanted To Know?. [Verkkosivu]. codeburst.io. [Viitattu 27.2.2019]. Saatavana: https://codeburst.io/all-about-node-js-you-wanted-to-know-25f3374e0be7

Chinda, G. Ei päiväystä. Pure Functional Components in React 16.6. [Verkkosivu]. LogRocket. [Viitattu 12.2.2019]. Saatavana: https://logrocket.com/blog/pure-functional-components/

Chinnathambi, K. 2016. Transferring Properties. [Verkkosivu]. Kirupa. [Viitattu 16.4.2019]. Saatavana: https://www.kirupa.com/react/transferring_properties.htm

Codecademy. Ei päiväystä. What is CORS? [Verkkosivu]. Codecademy. [Viitattu 23.3.2019]. Saatavana: https://www.codecademy.com/articles/what-is-cors

Codecademy. Ei päiväystä. What is REST? [Verkkosivu]. Codecademy. [Viitattu 13.3.2019]. Saatavana: https://www.codecademy.com/articles/what-is-rest

Database.Guide. 2016. What is Microsoft Access? [Verkkosivu]. Database.Guide. [Viitattu 27.2.2019]. Saatavana: https://database.guide/what-is-microsoft-access/

GeeksforGeeks. Ei päiväystä. ReactJS | State in React. [Verkkosivu]. GeeksforGeeks. [Viitattu 10.2.2019]. Saatavana: https://www.geeksforgeeks.org/reactjs-state-react/

Gray, T. 2018. What the heck is npm? [Verkkosivu]. Medium. [Viitattu 13.3.2019]. Saatavana: https://medium.com/@tanya/what-the-heck-is-npm-b8168f61e3b5

Hamedani, M. 2018. React Virtual DOM Explained in Simple English. [Verkkosivu]. Programming with Mosh. [Viitattu 16.4.2019]. Saatavana: https://programming-withmosh.com/react/react-virtual-dom-explained/

Hock-Chuan, C. 2010. A Quick Start Tutorial on Relational Database Design. [Verkkosivu]. Nanyang Technological University Signapore. [Viitattu 25.2.2019]. Saatavana: https://www.ntu.edu.sg/home/ehchua/programming/sql/relational_database_design.html

Hovi, A., Huotari, J. & Lahdenmäki, T. 2005. Tietokantojen suunnittelu & indeksointi. Porvoo: Docendo Finland Oy.

Ighodaro, N. 2018. A beginner's guide to the React component lifecycle. [Verkkosivu]. Pusher. [Viitattu 14.2.2019]. Saatavana: https://blog.pusher.com/beginners-guide-react-component-lifecycle/

Jöch, D. 2018. Functional vs Class-Components in React. [Verkkosivu]. Medium. [Viitattu 12.2.2019]. Saatavana: https://medium.com/@Zwenza/functional-vs-class-components-in-react-231e3fbd7108

Lathiya, K. Ei päiväystä. ReactJS Constructor Tutorial. [Verkkosivu]. CodinGame. [Viitattu 14.2.2019]. Saatavana: https://www.codingame.com/playgrounds/8508/reactjs-constructor-tutorial

Lindley, C. Ei päiväystä. What is JSX? [Verkkosivu]. Frontend Masters. [Viitattu 26.3.2019]. Saatavana: https://www.reactenlightenment.com/react-jsx/5.1.html

MDN web docs. Ei päiväystä. Express/Node introduction. [Verkkosivu]. Mozilla. [Viitattu 19.3.2019]. Saatavana: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction

Minnick, C. 2016. The Real Benefits of the Virtual DOM in React.js. [Verkkosivu]. Accelebrate. [Viitattu 12.2.2019]. Saatavana: https://www.accelebrate.com/blog/the-real-benefits-of-the-virtual-dom-in-react-js/

Node.js Foundation. 2011. What is npm? [Verkkosivu]. Node.js Foundation. [Viitattu 13.3.2019]. Saatavana: https://nodejs.org/en/knowledge/getting-started/npm/what-is-npm/

Ofoegbu, V. 2018. The only NodeJs introduction you'll ever need. [Verkkosivu]. codeburst.io. [Viitattu 26.2.2019]. Saatavana: https://codeburst.io/the-only-nodejs-introduction-youll-ever-need-d969a47ef219

Pineault, D. 2018. MS Access-Runtime. [Verkkosivu]. DevHut. [Viitattu 27.2.2019]. Saatavana: https://www.devhut.net/2018/02/10/ms-access-runtime/

Poolet, M. 1999. SQL By Design: The Foreign Key. [Verkkosivu]. ITProToday. [Viitattu 25.2.2019]. Saatavana: https://www.itprotoday.com/sql-server/sql-design-foreign-key

Ravichandran, A. 2018. Props and State in React Native explained in Simple English. [Verkkosivu]. codeburst.io. [Viitattu 13.2.2019]. Saatavana: https://codeburst.io/props-and-state-in-react-native-explained-in-simple-english-8ea73b1d224e

React. Ei päiväystä. Components and Props. [Verkkosivu]. Facebook. [Viitattu 10.2.2019]. Saatavana: https://reactjs.org/docs/components-and-props.html

React. Ei päiväystä. Create a New React App. [Verkkosivu]. Facebook. [Viitattu 23.3.2019]. Saatavana: https://reactjs.org/docs/create-a-new-react-app.html

Rouse, M. Ei päiväystä. middleware. [Verkkosivu]. TechTarget. [Viitattu 23.3.2019]. Saatavana: https://searchmicroservices.techtarget.com/definition/middleware

Seinäjoki. 2016. Tietoa Taskuun. [PDF-tiedosto]. Seinäjoki. [Viitattu 5.2.2019]. Saatavana: https://www.seinajoki.fi/material/attachments/seinajokifi/seinajoenkaupunki/tietoaseinajoesta/vK7tIJRXn/Tietoataskuun_2016.pdf

Seinäjoki. 2017. Kaavoitusohjelma 2017-2021. [PDF-tiedosto]. Seinäjoki. [Viitattu 10.2.2019]. Saatavana: https://www.seinajoki.fi/material/attachments/seinajokifi/asuminenjaymparisto/kaavoitus/xFv8uanfB/Kaavoitusohjelma_2017_-_2021.pdf

Seinäjoki. Ei päiväystä. Kaupunkisuunnittelu ja kaavoitus. [Verkkosivu]. Seinäjoki. [Viitattu 5.2.2019]. Saatavana: http://www.seinajoki.fi/asuminenjaymparisto/kaupunkisuunnittelujakaavoitus.html

Seinäjoki. Ei päiväystä. Organisaatio ja hallinto. [Verkkosivu]. Seinäjoki. [Viitattu 6.2.2019]. Saatavana: https://www.seinajoki.fi/seinajoenkaupunki/organisaatio.html

SQLite Tutorial. Ei päiväystä. What is SQLite. [Verkkosivu]. SQLite Tutorial. [Viitattu 14.3.2019]. Saatavana: http://www.sqlitetutorial.net/what-is-sqlite/

TechTarget. 2018. relational database. [Verkkosivu]. TechTarget. [Viitattu 16.1.2019]. Saatavana: https://searchdatamanagement.techtarget.com/definition/relational-database

Tutorialspoint. Ei päiväystä. Node.js – Event loop. [Verkkosivu]. tutorialspoint. [Viitattu 4.3.2019]. Saatavana: https://www.tutorialspoint.com/nodejs/nodejs_event_loop.htm

TutorialsTeacher. Ei päiväystä. Node.js Process Model? [Verkkosivu]. TutorialsTeacher. [Viitattu 27.2.2019]. Saatavana: https://www.tutorialsteacher.com/nodejs/nodejs-process-model

TutorialsTeacher. Ei päiväystä. What is Node.js? [Verkkosivu]. TutorialsTeacher. [Viitattu 28.2.2019]. Saatavana: https://www.tutorialsteacher.com/nodejs/what-is-nodejs

Upadhyay, P. 2017. What exactly does body-parser do with express.js and why do I need it? [Verkkosivu]. Quora. Saatavana: https://www.quora.com/What-exactly-does-body-parser-do-with-express-js-and-why-do-I-need-it

Wheeler, K. 2014. Learning React.js: Getting Started and Concepts. [Verkkosivu]. Scotch.io. [Viitattu 10.2.2019]. Saatavana: https://scotch.io/tutorials/learning-react-getting-started-and-concepts