Antti Halonen

# Web-pohjainen tiedonhaku tuotannonseuranta-järjestelmästä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

31.3.2018

Metropolia

| | |
|---|---|
| Tekijä<br>Otsikko<br><br>Sivumäärä<br>Aika | Antti Halonen<br>Web-pohjainen tiedonhaku tuotannonseurantajärjestelmästä<br><br>46 sivua + 5 liitettä<br>31.3.2018 |
| Tutkinto | insinööri (AMK) |
| Tutkinto-ohjelma | Tietotekniikka |
| Ammatillinen pääaine | Ohjelmistotekniikka |
| Ohjaajat | R&D Senior Engineer Matti Saavalainen<br>Yliopettaja Petri Vesikivi |


Insinöörityön tarkoituksena oli selvittää, onko mahdollista tehdä ohjelma, joka hakisi tiedot tietokannasta niin, että tiedot olisivat helposti luettavissa niin tietokoneen kuin mobiililaitteen selaimella.

Työssä käytettiin relaatiotietokantaa ensisijaisena tietokantana ja kannan tiedot haettiin teollisuuden tuotannonseurantajärjestelmästä. Tarvittaessa tutkittiin myös muita tietokantavaihtoehtoja. Ohjelman ajoympäristöksi valikoitui avoimen lähdekoodin Node.js ja tietokannaksi Oracle Database. Työssä selvitettiin, voidaanko ohjelma rakentaa käyttämällä Node.js:n moduuleita ja mikä Node.js:n taulukkomoduuleista olisi paras ratkaisu ohjelman tekemiseen. Insinöörityössä oli tarkoituksena tehdä esimerkkitoteutus käyttämällä tutkimuksesta saatuja tietoja.

Tutkimukset osoittivat, että Node.js ja sen laaja moduulikirjasto olivat erittäin hyvä valinta, kun haluttiin tehdä sekä palvelin- että asiakaspään ohjelma. Tietokannaksi valittu Oraclen tietokanta osoittautui hyväksi valinnaksi, koska sille oli Oraclen itse tekemä ajurimoduuli Node.js:lle. Oraclen tietokanta on myös vakiintunut teollisuuden käyttöön. Vertailussa eri taulukkomoduuleiden välillä parhaaksi taulukkomoduuliksi osoittautui jsGrid sen hyvän ominaisuusmäärän, hyvän lisenssin ja lokalisaatiotuen ansiosta.

Tutkimustulosten pohjalta pystyttiin tekemään ohjelma, joka täytti sille asetetut vaatimukset tiedon hakemisesta ja näyttämisestä selaimessa. Ohjelma tehtiin tutkimuksissa hyväksi osoittautuneen ohjelmointitavan mukaisesti modulaarista ohjelmointia käyttäen.




| | |
|---|---|
| Avainsanat | Node.js, avoin lähdekoodi, web-pohjainen tiedonhaku |

| | |
|---|---|
| Author | Antti Halonen |
| Title | Web-based data retrieval from production monitoring system |
| Number of Pages | 46 pages + 5 appendices |
| Date | 31.3.2018 |
| Degree | Bachelor of Engineering |
| Degree Programme | Information Technology |
| Professional Major | Software Engineering |
| Instructors | Matti Saavalainen, R&D Senior Engineer<br>Petri Vesikivi, Principal Lecturer |

Objective of this thesis was to investigate whether it is possible to make an application that would get the data from database in such a way, that the data would be easily read on the desktop and mobile browsers.

The application would be using a relational database as a primary database and the Data to the database would come from industrial production monitoring system. Other database options would be investigated, if necessary.

The open-source Node.js was chosen as the runtime environment for the example application and the Oracle Database was chosen as the relational database.

The thesis should tell, if is possible to build an application by using the modules from Node.js. And to find out what is the best grid-module in Node.js for this kind of application. A program will be made using the data from the study in this thesis.

The studies showed that Node.js and its vast module library had been a very good choice when making both the server-side and the client-side software. The chosen database, Oracle Database, turned out to be a good choice, because Oracle had made a driver module for the database themselves. Oracle Database is also almost a industry standard. When comparing the different Node.js grid modules the best one turned out to be the jsGrid, because it had a good feature list, a good license, and support for localization.

It was possible to make an application with the results of the study that met the requirements set for it to retrieve and display information in a browser on desktop and mobile. The application was made by using the modular programming design technique. The technique was found to be good for this kind of application.

| | |
|---|---|
| Keywords | Node.js, Open-source, Web-based data retrieval |

## Sisällys

Liitteet

Liite 1. Server.js

Liite 2. DBtoJSON.js

Liite 3. Index.html

Liite 4. grid_jsGrid.js

Liite 5. Taulukkomoduulien ominaisuudet

## Lyhenteet

| | |
|---|---|
| I/O | Input/Output eli siirräntä tarkoittaa tiedon siirtämistä tietokoneen komponenttien välillä. Esimerkiksi näppäimistöltä luettava teksti tai CD-asemasta luettava CD-levy ovat siirräntää. |
| npm | Node Package Manager on Node.js-pakettienhallintajärjestelmä. |
| JSON | JavaScript Object Notation on yksinkertainen avoimen standardin tiedostomuoto tiedonvälitykseen. Nimestään ja JavaScript-perustastaan huolimatta JSON on JavaScriptistä riippumaton. Sen tiedostopääte on .json. |
| REST | Representational State Transfer on HTTP-protokollaan perustuva arkkitehtuurimalli ohjelmointirajapintojen toteuttamiseen. |
| V8 | The V8 JavaScript Engine on avoimen lähdekoodin JavaScript-moottori, jonka The Chromium Project kehitti Google Chrome -selainta varten. |
| HTTP | Hypertext Transfer Protocol (eli hypertekstin siirtoprotokolla) on protokolla, jota selaimet ja WWW-palvelimet käyttävät tiedonsiirtoon. |
| HTML | Hypertext Markup Language (eli hypertekstin merkintäkieli) on avoimesti standardoitu kuvauskieli, jolla voidaan kuvata hyperlinkkejä sisältävää tekstiä eli hypertekstiä. HTML:llä voidaan myös merkitä tekstin rakenne eli esimerkiksi se, mikä osa tekstistä on otsikkoa ja mikä leipätekstiä. HTML tunnetaan erityisesti kielenä, jolla internetsivut on kirjoitettu. |
| PHP | Hypertext Preprocessor on ohjelmointikieli, jota käytetään erityisesti web-palvelinympäristöissä dynaamisten verkko-sivujen luonnissa. |
| CSS | Cascading Style Sheets on erityisesti WWW-dokumenteille kehitetty tyyliohjeiden laji. CSS:ssä dokumentille voi määritellä useita tyyliohjeita, jotka yhdistetään tietyllä tavalla yhdeksi säännöstöksi. |
| XML | Extensible Markup Language on tietynlaisten merkintäkielien yläkäsite tai standardi, jolla tiedon merkitys on kuvattavissa tiedon sekaan. XML-kieliä |

käytetään sekä formaattina tiedonvälitykseen järjestelmien välillä että formaattina dokumenttien tallentamiseen. XML-kieli on rakenteellinen kuvauskieli, joka auttaa jäsentämään laajoja tietomassoja selkeämmin.

AJAX Asynchronous JavaScript and XML on joukko tekniikoita, joilla pystyy tekemään nopeita ja dynaamisia web-sovelluksia

.

# 1 Johdanto

Tiedonhaku tuotannon tietokannoista on monesti hidas ja työläs prosessi. Jos tarvitsee vain näyttää tietoja esimerkiksi tehtaiden taukotiloissa tai valvomoissa eikä tietojen käsitteleminen tai muokkaaminen ole olennaista, ovat tietokantatyökalujen monet ominaisuudet vain tiellä ja hidasteena. Ne eivät myöskään sovellu selain- tai mobiilikäyttöön.

Tämän insinöörityön toimeksiantajalla, ABB:llä, on monta asiakasta, joilla olisi tarvetta ohjelmalle, jolla pystyisi hakemaan ja näyttämään tietokannan tietoja niin, että tiedot olisivat helposti luettavissa niin tietokoneen kuin mobiililaitteen selaimella. Insinöörityössä oli tarkoituksena tutkia, onko tällaisen ohjelman tekeminen mahdollista käyttäen relaatiotietokantaa ensisijaisena tietokantana ja tarvittaessa tutkittaisiin myös muita tietokantavaihtoehtoja. Tieto haettaisiin teollisuuden tuotannonseurantajärjestelmästä. Tutkimuksen jälkeen oli tarkoitus tehdä yksi ohjelma esimerkkitoteutuksena tutkimuksesta saatuja tietoja hyväksi käyttäen. Ohjelman ajoympäristöksi valikoitui avoimen lähdekoodin Node.js ja tietokannaksi Oracle Database.

Insinöörityöraportti on jaettu kolmeen osaan. Ensimmäisessä osassa käsitellään toteutustapojen sopivuutta esimerkkitoteutuksen tekemiseen sekä taulukkomoduulin valintaprosessia. Toisessa osassa käydään läpi, mitä esimerkkitoteutusta tehdessä pitää huomioida: ohjelman käytettävyys, tietoturva ja lisenssit. Kolmas osa käsittelee esimerkkitoteutusta, ja siinä käydään läpi, miten ohjelma toimii. Sen seuraamisen helpottamiseksi ohjelman tiedostot ovat työn liitteinä kokonaisuudessaan.

# 2 Toteutustapojen sopivuus web-pohjaiseen tiedonhakuun

## 2.1 Avoimen lähdekoodin käyttö

Yleensä avoimesta lähdekoodista puhuttaessa tarkoitetaan tietokoneohjelmia, joiden lähdekoodi on kaikkien saatavilla käytettäväksi, muokattavaksi ja jaettavaksi eteenpäin. Sen tarkoituksena on olla yhteinen projekti, jossa kaikki voivat yrittää parantaa lähdekoodia ja jakaa muutokset muiden kanssa. Monesti näin ei kuitenkaan ole, vaan lähdekoodi on vain julkaistu jollain avoimella lisenssillä.

Lähdekoodi on se osa ohjelmasta, jonka ohjelmoija on kirjoittanut jollain ohjelmointikielellä ja joka määrittää, miten ohjelma toimii. Tämä luettavissa oleva teksti käännetään konekielisiksi käskyiksi, joita tietokone sitten seuraa. Lähdekoodia muokkaamalla ohjelmaan pystyy lisäämään ominaisuuksia ja korjaamaan virheitä. Kaupallisten ohjelmien lähdekoodia ei yleensä näytetä käyttäjälle, koska muuten ohjelman kopioiminen ja muokkaaminen olisi helppoa. [1.]

Avoimen lähdekoodin käyttö yleistyi internetin mukana. Internet mahdollisti helpon tavan jakaa ja hallita avoimen lähdekoodin projekteja. Nykypäivänä ehkä tunnetuin sivusto avoimen lähdekoodin jakamiseen ja hallintaan on GitHub. Sieltä löytyy jopa Yhdysvaltojen ilmailu- ja avaruushallintoviraston NASA:n avoimia projekteja. [2]

Ensimmäinen etu, joka yleensä huomataan avoimen lähdekoodin käytössä, on sen olematon hinta. Avoimet ohjelmat ovat lähtökohtaisesti aina ilmaisia, ja jos niissä on oikeanlainen lisenssi, saa niitä, tai osia niistä, jopa myydä itse eteenpäin.

Avoimen lähdekoodin suurimmat edut ovat kuitenkin muokattavuus ja iso kehitystiimi. Isosta määrästä kehittäjiä seuraa se, että koodin laatu ja tietoturvallisuus on hyvää. Vuoden 2013 Coverity Scan Open Source -raportissa [3, s. 8–9] todetaan, että avoimen lähdekoodin ohjelmien koodin laatu on ylittänyt suljettujen lähdekoodien laadun. Eric Raymonin [4, s. 9–11] mukaan tämä johtuu siitä, että kun iso joukko kehittäjiä käy koodia läpi virheet ja epäoptimaaliset koodin pätkät löydetään varmemmin ja nopeammin kuin pienellä kehitystiimillä. Isossa kehitystiimissä on myös se etu, että kaikkea ei tarvitse tehdä itse, vaan joku muu on voinut jo tehdä sen. Tähän ei kuitenkaan voi luottaa sokeasti, koska yleensä vapaan lähdekoodin ohjelmaa kehitetään omiin käyttötarkoituksiin

eikä voi olettaa, että juuri tietty ongelma koskee muita käyttäjiä. Muokattavuuden ansiosta ohjelmaa voidaan käyttää monella eri tavalla ja ohjelmaa voi muokata omiin käyttötarkoituksiin sopivaksi. Tämä taas saa useamman käyttäjän ja kehittäjän kiinnostumaan ohjelmasta, mikä taas kasvattaa kehittäjien ja testaajien määrää parantaen laatua ja ominaisuuksia. Muokattavuuden ansiosta vaihtoehtoja eri ohjelmille tai niiden osille on valtavasti, ja siksi ongelmaan, jota ohjelmalla pyritään ratkaisemaan, löytyy jo valmis ratkaisu eikä sitä tarvitse tehdä itse. Vaihtoehtojen määrä tuo kuitenkin mukanaan myös ongelmia löytää juuri se paras ratkaisu. Tässä onneksi auttaa avoimen lähdekoodin käyttäjien yhteisöllisyys. Yhteisöllä on tapana erottaa jyvät akanoista ja nostaa esille toimivimmat ratkaisut.

Avoimen lähdekoodin ohjelmat eivät myöskään ole vain yhden ohjelmistokehittäjän takana, joten on epätodennäköisempää, että ohjelman tuki tai saatavuus loppuu, vaikka alkuperäinen kehittäjä lopettaisi ohjelman tukemisen. Jos alkuperäinen kehittäjä lopettaa ohjelman kehittämisen avoimen lähdekoodin ohjelmassa, ottaa yleensä joku toinen asiasta kiinnostunut sen kehitettäväkseen näin jatkaen ohjelman kehittämistä.

Vaikka lupaukset ilmaisuudesta ja paremmasta laadusta ja tietoturvasta kuulostavatkin hyviltä, eivät avoimen lähdekoodin ohjelmat ole silti ongelmattomia. Yksi yleisimmistä ongelmista, johon kokemattomat käyttäjät törmäävät, on se, että avoimen lähdekoodin ohjelmat eivät ole aina niin helppoja käyttää kuin suljetut ohjelmat. Suljettujen ohjelmien takana on monesti erittäin paljon käyttäjätestausta, jotta ohjelmasta saataisiin mahdollisimman helppo käyttää. Koska avoimet ohjelmat ovat muokattavia, ei tämä ole yleensä mahdollista, sillä jokaisen käyttäjän ohjelma voi olla periaatteessa erilainen. Käytön vaikeus korostuu etenkin silloin, kun suljetusta ohjelmasta on tullut vakiintunut ohjelma, esimerkiksi Microsoft Office ja vastaava avoimen lähdekoodin Apache OpenOffice.

Teknisen tuen puute voi olla ongelma avoimissa ohjelmissa. Vaikka internetin keskustelupalstat ovatkin täynnä ratkaisuja ja auttajia, voi juuri se ongelma, mikä itsellä on, olla vain oma ongelma. Parempi siis valmistautua siihen, että ongelmat on itse ratkaistava. Suljettujen ohjelmien mukana tulee yleensä jonkinlainen lupaus tuen antamisesta, joten niissä ei tätä ongelmaa yleensä esiinny.

Ongelmaksi avoimen lähdekoodin ohjelmia käyttäessä voi myös muodostua, ohjelman liittäminen toisiin ohjelmiin ja työnkulkuun. On mahdollista, että avoimen lähdekoodiin ohjelma ei pysty esimerkiksi tallentamaan tietoja juuri siinä muodossa, jota jokin suljetun

lähdekoodin ohjelma käyttää, tai muuten toimimaan yhteistyössä suljetun ohjelman kanssa. Avoimen lähdekoodin ohjelmaa pystyy tietysti muokkaamaan, jotta ongelma ratkeaisi, mutta se tietää lisää töitä eikä ole takuita, että se silloinkaan toimisi. Avoimen lähdekoodin ohjelmat ovat onneksi nykyään niin yleisiä, että ohjelmien kehittäjät tekevät monesti itse tarvittavat muutokset, avoimet moduulit ja ajurit omien suljettujen ohjelmien käyttöön avoimien ohjelmien kanssa. Tästä on esimerkkinä insinöörityön esimerkkitoteutuksessa käytetty Oraclen tekemä oracledb-ajuri, joka sallii Oraclen tietokantojen käytön Node.js-ympäristössä [5].

Jos avoimen ohjelman tulee toimia jonkin harvinaisen suljetun laitteen kanssa, ei sille välttämättä löydy ajureita valmistajalta tai avoimen lähdekoodin yhteisöltä. Tässäkin korostuu se, että avoimen lähdekoodin ohjelmissa kaikki ongelmat voivat olla vain yhden käyttäjän ongelmia.

Koska avoimen lähdekoodin ohjelmissa on lähtökohtaisesti monta kehittäjää, on niissä myös melkein yhtä monta tapaa ohjelmoida. Saman projektin sisässä saattaa koodin luettavuus vaihdella paljon, koodin kommentoinnista puhumattakaan. Myös dokumentoinnin taso on vaihteleva ja dokumentointi monesti kirjoitettu niin, että se olettaa lukijan tuntevan projektin miltei läpikotaisin.

Jos avoimen lähdekoodin ohjelmaa on tarkoitus käyttää osana kaupallista ohjelmaa, pitää avoimen ohjelman lisensseihin tutustua tarkkaan. Osa avoimista lisensseistä sallii avoimen lähdekoodin käytön suljetuissa kaupallisissa ohjelmissa ilman ongelmia, mutta osa lisensseistä pakottaa kaikki sen alaista lähdekoodia käyttävät ohjelmat avoimiksi.

Avoimen lähdekoodin ohjelmat yleistyvät joka paikassa, etenkin kehittyvissä maissa, joissa avoimien ohjelmien ilmaisuus on iso tekijä. Avoimia ohjelmia käyttävät tällä hetkellä myös monet isot yhtiöt, kuten esimerkiksi Google, PayPal ja Netflix. Google on myös kehittänyt ja julkaissut monia avoimia projekteja, joista tunnetuimpia ovat Android-mobiilikäyttöjärjestelmä ja Chrome-selain. [6.]

Jos haluaa pysyä mukana kiihtyvässä kehitystahdissa tietotekniikan alalla, tulee mielestäni siirtyä niiltä osin, kuin se on järkevää, avoimen lähdekoodin käyttöön. Sen tuomat edut olivat tässä esimerkkitoteutuksessa suuremmat kuin haitat. Avoimen lähdekoodin ohjelmien käyttö mahdollisti esimerkkitoteutuksen tekemisen ja eri ratkaisujen kokeile-

misen ilman, että piti ostaa kalliita ohjelmia tai lisenssejä. Avoin lähdekoodi myös mahdollisti esimerkkitoteutuksessa käytettyjen ohjelmien ja niiden osien muokkaamisen esimerkkitoteutukseen sopivammiksi. Ilman avoimia ohjelmia esimerkkitoteutuksen tekeminen olisi ollut kallista ja hidasta, ellei jopa mahdotonta. Ei tulisi kuitenkaan olettaa, että kaikki ongelmat ratkeaisivat siirtymällä avoimiin ohjelmiin, vaan tulisi pyrkiä tasapainoon suljettujen ja avoimien ohjelmien käytössä.

## 2.2 Node.js-ajoympäristö

Node.js on avoimen lähdekoodin ajoympäristö (runtime environment) palvelinpään ohjelmien kehittämiseen. Se on hajautettu kehitysprojekti (distributed development), jonka kehitystä valvoo Node.js Foundation, ja se kuuluu Linux Foundationin yhteistyöprojektiohjelmaan. Vaikka monet sen perusmoduuleista on kirjoitettu JavaScriptillä, se ei kuitenkaan ole JavaScript-kehys (framework). Node.js on rakennettu Googlen avoimen lähdekoodin V8-JavaScript-moottorin päälle [7]. V8 on erittäin nopea ja hyvä suoriutumaan internetin perustehtävistä (esimerkiksi HTTP, DNS, TCP) [8]. JavaScriptin suosio ja tunnettavuus on saanut web-kehittäjät nopeasti omaksumaan Node.js:n omakseen.

Node.js:ssä on tapahtumavetoinen arkkitehtuuri (event-driven architecture). Siksi sovellukset ohjelmoidaan yleensä käyttämään epäsynkronista siirräntää (asynchronous input/output). Tällä pyritään optimoimaan suoritustehoa ja skaalautuvuutta verkkopohjaisissa sovelluksissa, joissa on paljon siirräntäoperaatioita tai reaaliaikaisia toimintoja.

Monet isot yhtiöt käyttävät Node.js:ää. Näihin kuuluvat esim. IBM, Microsoft ja Netflix.

Node.js mahdollistaa palvelimien ja verkkovälineiden (networking tools) tekemisen JavaScriptillä ja moduulikokoelmilla. Moduulit pystyvät suorittamaan monenlaisia ydintoimintoja, kuten esimerkiksi tiedostojen siirräntä, verkon käyttö (DNS, HTTP), binääridatan puskurointi ja tietovirran (data stream) hallinta. Node.js:n moduulit käyttävät ohjelmointirajapintaa, joka on suunniteltu vähentämään palvelinohjelmien tekemisen monimutkaisuutta.

Node.js-ohjelmia ajetaan OS X-, Windows-, NonStop- sekä Unix-ympäristöissä. JavaScriptin lisäksi ohjelmat voidaan kirjoittaa CoffeeScriptinä, Dartina, TypeScriptinä tai millä tahansa muulla kielellä, joka kääntyy JavaScriptiksi.

Node.js:ää käytetään enimmäkseen verkko-ohjelmien (esimerkiksi verkkopalvelimien) tekemiseen. Node.js:n ja PHP:n (toinen yleinen ajoympäristö verkko-ohjelmille) suurin ero on se, että PHP estää muita käskyjä suorittumasta, ennen kuin kesken oleva käsky on valmis, kun taas Node.js:ssä käskyt suoritetaan rinnakkain. Node.js toimii parhaiten, kun sitä ylläpidetään pilvialustoilla. Se on teknisiltä vaatimuksiltaan niin kevyt, että sen ylläpitoon riittää kevytkin pilvialusta.

Node.js:n tapahtumavetoinen arkkitehtuuri mahdollistaa helposti skaalautuvien palvelimien tekemisen ilman säikeistämistä (threading) käyttämällä yksinkertaistettua mallia tapahtumavetoisesta ohjelmoinnista. Mallissa käytetään takaisinkutsufunktiota (callback) kertomaan tehtävien valmistumisesta. Node.js kehitettiin, koska rinnakkaisuus on vaikeaa monille palvelinpään ohjelmointikielille, mikä johtaa yleensä huonoon suorituskykyyn. [9, s. 15–19.]

Node.js toimii yhdellä säikeellä käyttäen epäsynkronista siirräntää. Tämä sallii kymmenien tuhansien samanaikaisten yhteyksien toimimisen ilman säikeen vaihtamisesta tulevaa hidastusta. Tämän yksisäikeisen toiminnan huonona puolena on se, että Node.js ei salli vertikaalista skaalausta lisäämällä prosessorin ytimien määrää ilman erillisen moduulin käyttöä.

Npm on Node.js:n tehokas ja monipuolinen pakettienhallintajärjestelmä. Node.js:n perusajatus on, että sen ydin on mahdollisimman kevyt ja vain tarvittavat toiminnallisuudet otetaan käyttöön ulkoisten pakettien ja moduulien (modules) avulla. Melkein kaikki paketeista sisältävät monia moduuleja. Moduulit ovat se osa paketista, jota käytetään tekemään tarvittavat toiminnot Node.js-ohjelmassa. Siksi npm on välttämätön työkalu Node.js:n tehokkaaseen käyttöön. Esimerkiksi pakettien välisiä riippuvuuksia voidaan hallita moduulikohtaisesti ja riippuvuudet asentuvat automaattisesti. Lisäksi yksittäinen sovellus voi sisältää komponenttiensa kautta riippuvuuksia samojen kirjastojen eri versioihin, vaikka versiot olisivat keskenään epäyhteensopivia. Versioinnista saa myös semanttisen semver-lisäsovelluksella. Myös pakettien asennus, listaus ja poistaminen ovat triviaaleja toimenpiteitä npm:llä. [10]

Ajoympäristöksi esimerkkitoteutukseen valittiin Node.js, koska se on avoimen lähdekoodin ajoympäristö, joka tukee modulaarista ohjelmointia. Tämän ansioista se sopii ketterään ja nopeaan kehittämiseen. Esimerkkitoteutuksessa tulee tehdä ohjelman asiakas- ja palvelinpää. Node.js:ssä asiakas ja palvelin käyttävät samaa kieltä, mikä helpottaa

kehittämistä. Node.js tukee epäsynkronista ajoa, mikä sopii erinomaisesti palvelinkäyttöön. Node.js:n takana on vakuuttavia tekijöitä, ja se on monilla isoilla yrityksillä käytössä. Tämä toimii todisteena siitä, että Node.js toimii ja skaalautuu isoihinkin tarpeisiin.

## 2.3 Modulaarinen ohjelmointi

Modulaarinen ohjelmointi on ohjelmiston suunnittelumalli, joka erottaa ohjelman toiminnallisuuden itsenäisiksi vaihdettavissa oleviksi moduuleiksi, niin että jokainen moduuli sisältää kaiken tarvittavan suorittaakseen vain yhden osan halutusta toiminnallisuudesta. Moduulin rajapinta kertoo, mitä muuttujia ja vakioita moduuli tarvitsee parametreina ja mitä se antaa ulos. Muut moduulit näkevät vain nämä rajapinnassa määritellyt asiat. Näin vältytään tekemästä ylimääräisiä sidoksia muuhun koodiin ja moduuli voidaan tarvittaessa siirtää muualle ilman, että tulee virheitä sidosten puuttumisesta. [11, s. 9–16]

Moduulit tulisi pyrkiä järjestämään hierarkkisesti, niin että alimman tason moduulit olisivat täysin itsenäisiä, eikä niiden toiminta riippuisi mistään muusta moduulista ja ylemmän tason moduulit olisivat riippuvaisia alemman tason moduuleista. Kun nämä kaikki eri tason moduulit yhdistetään kokonaisuudeksi, tulee siitä toimiva ohjelma. Tämän ansiosta oikein tehty modulaarinen ohjelma ja sen osat on paljon helpompi käyttää uudelleen toisessa projektissa ilman, että niihin tarvitsee tehdä paljon muutoksia. Tämä mahdollistaa myös projektin pilkkomisen useampaan pieneen projektiin, mikä tekee siitä erinomaisen mallin avoimien lähdekoodien projekteihin. Jokainen kehittäjä voi tehdä omaa moduuliaan ilman, että se vaikuttaa muiden moduuleihin, kunhan sisään ja ulos tulevat parametrit tiedetään. [11, s. 9–16]

Modulaarinen ohjelmointi on myös mahdollistanut sen, että esimerkkitoteutuksessa käytetyssä Node.js-ympäristössä on käytössä oma kirjasto valmiita moduuleita melkein jokaiseen ongelmaan ja valmiit työkalut moduulien hallinnoimiseen. Myös moduulien väliset riippuvuudet, jopa moduulien eri versioiden välillä, hoituvat automaattisesti moduulia ladattaessa.

Tässä insinöörityössä eri moduulien lähdekoodeja tutkimalla selvisi, että ohjelman tekeminen moduuleita käyttämällä on nopeaa ja helpompaa kuin tehdä kaikki itse. Monien moduulien lähdekoodit olivat hyvinkin monimutkaisia ja vastaavien ominaisuuksien tekeminen itse olisi vaatinut paljon työtä. Moduulien käyttörajapinnat olivat kuitenkin yleensä

selkeitä ja dokumentoituja. Valinnan varaa moduuleissa on enemmän kuin tarpeeksi, ja tämän vuoksi on välillä vaikeaa löytää sitä juuri sopivaa sen hetkiseen ongelmaan. Eri moduulien taso vaihtelee paljonkin, etenkin mitä tulee dokumentoimiseen ja koodin selkeyteen. Suurimmalla osalla moduuleista on kuitenkin GitHub-sivut, joista avun saaminen on yleensä melko nopeaa.

Moduulien yhteensopivuudessa oli toisinaan ongelmia, mutta suosittuihin moduuleihin löytyy yleensä yhteensopivuus päivityksiä valmiina. Avoin lähdekoodi toki mahdollistaa ongelmien korjaamisen myös itse.

Modulaarinen ohjelmointi sopii erinomaisen hyvin nopeiden prototyyppien tekemiseen ja uusien ideoiden kokeilemiseen, mikä sopii esimerkkitoteutuksen tekemiseen. Pienistä yksinkertaisista kokonaisuuksista pystytään kuitenkin muodostamaan monimutkaisia rakenteita, jos niin halutaan. Tekemällä esimerkkitoteutus modulaariseksi pystytään siinä kokeilemaan eri taulukko-ohjainmoduuleja ilman, että koko ohjelmaa tarvitsisi muokata. Modulaarinen ohjelmointi auttaa myös välttämään epäsynkronisen siirrännän sisäkkäisten takaisinkutsufunktioiden aiheuttamaa koodin epäselvyyttä ja tehottomuutta. Kun ohjelma on jaettu pienempiin modulaarisiin osiin, on kokonaisuus helpompi hahmottaa ja ylläpitää.

Node.js:n moduulikirjastosta löytyy moduuleja miltei jokaiseen käyttötarkoitukseen, joten niiden avulla esimerkkitoteutus pystytään tekemään.

## 2.4 Epäsynkroninen siirräntä

Siirräntä (I/O tai Input/Output) tarkoittaa tiedon siirtämistä tietokoneen komponenttien välillä. Esimerkiksi näppäimistöltä luettava teksti tai CD-asemasta luettava CD-levy ovat siirräntää. Epäsynkroninen siirräntä (asynchronous I/O) on siirräntätapa, joka sallii muiden prosessien jatkaa suorittamistaan, vaikka siirräntä olisi vielä kesken. Tästä tulee sen toinen nimitys, epäestävä siirräntä (non-blocking I/O).

Tietokoneiden siirräntäoperaatiot voivat olla erittäin hitaita verrattuna niiden tietojenkäsittelynopeuteen. I/O-laite voi joutua fyysisesti liikuttamaan jotain osaansa, esimerkiksi kiintolevy lukupäätään, ja se on yleensä huomattavasti hitaampaa kuin valon nopeudella

tapahtuva sähkövirran muuttaminen. Esimerkiksi kymmenen millisekuntia kestävän levyoperaation aikana yhden gigahertsin prosessori on ehtinyt suorittaa kymmenen miljoonaa käskyjaksoa.

Estävä siirräntä on yksinkertaisin tapa siirrännälle. Siinä tarvitsee vain aloittaa siirräntä ja sitten odottaa sen valmistumista. Tästä tavasta seuraa kuitenkin se, että muun ohjelman suoritus pysähtyy siksi aikaa, kunnes siirräntä on valmis. Jos ohjelman pitää tehdä monta siirräntäoperaatiota, voi prosessori joutua odottamaan toimettomana pitkiä aikoja. Etenkin palvelinohjelmissa tämä voi muodostua ongelmaksi, kun monta eri asiakasta lähettää samaan aikaan HTTP-pyyntöjä palvelimelle. [12; 13.]

Toinen tapa, ja palvelimen tapauksessa parempi tapa, suorittaa siirräntää on aloittaa se ja suorittaa saamaan aikaan muita prosesseja, jotka eivät ole riippuvaisia siirrännästä saatavista tiedoista. Tätä tapaa kutsutaan epäsynkroniseksi siirrännäksi tai epäestäväksi siirrännäksi. Siirrännästä riippuvaiset prosessit joutuvat silti odottamaan siirrännän valmistumista, mutta siirrännästä riippumattomat prosessit voidaan suorittaa siirrännän aikana. Näin esimerkiksi kesken olevat yksittäiset HTTP-pyynnöt eivät estä muita pyyntöjä toteutumasta, vaan ne suoritetaan rinnakkain. [12; 13.]

Epäsynkroninen siirräntä soveltuu hyvin sovelluksiin, joissa prosessi voi kestää pitkään ja aiheuttaa näin viivytyksen muiden prosessien suorittamiseen. Näitä ovat esimerkiksi peräkkäisten likiarvojen laskenta tai luokittelu. Tällaisissa tapauksissa on tehokkaampaa lähettää pyyntö toiselle palvelulle, joka on erikoistunut kyseisiin tehtäviin. Sillä aikaa, kun toinen palvelu suorittaa pyyntöä, voi muu ohjelma jatkaa toimintaansa.

Epäsynkronisen siirrännän ohjelmoinnissa pitää ottaa huomioon tapa, millä epäsynkroninen siirräntä on tarkoitus toteuttaa. Eri tavoissa on omat hyvät ja huonot puolensa. Tässä insinöörityössä keskitytään kuitenkin vain esimerkkitoteutuksessa käytetyn ajoympäristön, Node.js:n, takaisinkutsufunktioita (callback) käyttävään tapaan. Epäsynkroninen I/O on myös vaikeampaa ohjelmoida kuin synkroninen I/O, koska ohjelma ei etene perinteiseen tyyliin rivi riviltä alaspäin. On esimerkiksi mahdollista ja hyvinkin todennäköistä, että takaisinkutsufunktioiden sisällä on lisää takaisinkutsufunktioita, joiden sisällä on vielä lisää takaisinkutsufunktioita ja niin edelleen. Tähän auttaa oikeanlaisen ohjelmointitavan käyttäminen. Koodissa tulisi välttää sisäkkäisiä funktioita, ja funktioille tulisi antaa niitä kuvaavat nimet. Ohjelman tekeminen modulaariseksi auttaa välttämään sisäkkäisiä takaisinkutsufunktioita.

Epäsynkronisen siirrännän tarvitsema samanaikaisten tehtävien suoritus Node.js:ssä hoidetaan käyttämällä säieallasta (thread pool). Pääsäiekutsufunktiot (main thread call functions) lähettävät tehtäviä jaetulle tehtäväjonolle, josta säiealtaassa olevat säikeet lukevat ja suorittavat tehtäviä. Epäestävät tehtävät siirretään kernel-puolen epäestäviin kantoihin (socket), ja estävät tehtävät suoritetaan omissa säikeissään. Kun säiealtaan säie saa tehtävänsä tehtyä, se ilmoittaa siitä pääsäikeelle. Pääsäie herää ja suorittaa rekisteröidyn takaisinkutsufunktion. Koska takaisinkutsufunktiot suoritetaan peräjälkeen pääsäikeessä, pitkäkestoiset laskennat ja muut prosessoria kuormittavat tehtävät voivat pysäyttää koko tapahtumaketjun, kunnes tehtävä on saatu suoritettua. [14.]

Käytännön esimerkki siitä, miten epäsynkroninen siirräntä toimii Node.js:ssä:

1. Selain pyytää jotain HTML-sivua Node.js-palvelimelta.

2. Palvelin hyväksyy pyynnön ja kutsuu funktiota hakemaan ”/index.html”-tiedoston.

3. Sillä aikaa, kun tiedostoa haetaan, palvelin käsittelee jo seuraavaa pyyntöä.

4. Kun tiedosto on haettu, sijoitetaan Node.js palvelimen jonoon takaisinkutsufunktio.

5. Palvelin suorittaa funktion eli tässä tapauksessa piirtää ”/index.html”-sivun ja lähettää sen takaisin selaimeen, joka sitä pyysi. [14.]

Koska esimerkkitoteutuksessa on tarkoituksena tehdä palvelinsovellus ja Node.js tukee epäsynkronista siirräntää, parantaisi epäsynkronisuus esimerkkitoteutuksen suorituskykyä. Epäsynkronisuuden tuomia aikaisemmin mainittuja ”ohjelmointiansoja” tulisi kuitenkin välttää.

## 2.5 Oracle Database -tietokanta

Relaatiotietokanta on kokoelma tietoja, joilla on yhteys toisiinsa. Se muodostuu kaavioista, taulukoista, kyselyistä, raporteista, näkymistä ja muista objekteista. Tietokannan tarkoituksena on tallentaa ja hakea yhteyksissä olevia tietoja. Tiedot on yleensä järjestetty mallintamaan todellisuuden osaa tavalla, joka tukee tietoja vaativia prosesseja. [15.]

Tietokannanhallintajärjestelmä (database management system, DBMS) on tietokonesovellus, joka toimii käyttäjän, muiden sovellusten ja itse tietokannan kanssa tallentaen ja analysoiden tietoja. Hallintajärjestelmä on suunniteltu siten, että sillä pystyy vähintään määrittämään, luomaan, hakemaan, päivittämään ja hallinnoimaan tietokantaa. Tietokannat eivät yleensä ole yhteensopivia eri hallintajärjestelmien välillä, mutta käyttämällä standardeja rajapintoja, kuten esimerkiksi SQL ja ODBC, voi yksittäinen sovellus käyttää monia hallintajärjestelmiä samanaikaisesti. [16.]

Tietokannanhallintajärjestelmät luokitellaan yleensä sen mukaan, mitä tietokantamallia ne käyttävät. Suosituin malli 1980-luvulta lähtien on ollut relaatiomalli SQL-kielellä esitettynä.

Tietokannaksi valittiin Oraclen tietokanta ja sen käyttämiseen Oracle Database (yleisesti tunnetaan nimellä Oracle RDBMS tai Oracle). Oracle Database on Oracle Corporationin kehittämä objekti-relaatiotietokannanhallintajärjestelmä. Oraclen tietokanta valittiin, koska se on todettu käytössä parhaaksi valinnaksi teollisuuden käyttöön. Se on myös ABB:llä vakiintunut käyttöön 20 vuoden aikana. Tietokanta on ollut ABB:n asiakkailla käytössä vuodesta 1987 lähtien [17]. Oracle on myös itse kehittänyt Node.js-ajurin tietokantansa käyttöön, joten yhteensopivuus tai tietoturvaongelmia ei pitäisi olla.

## *2.6* JSON tiedon esitysmuotona

JSON (JavaScript Object Notation) on yksinkertainen tiedon esitysmuoto tiedon välitykseen. Esitysmuodon rakenne on yksinkertainen ja helppo ihmisen lukea sekä kirjoittaa. Myös tietokoneiden on helppoa luoda ja jäsentää sitä. JavaScript-juuristaan riippumatta se on täysin kieliriippumaton. JavaScript-juuret mahdollistavat kuitenkin JSON:n helpon jäsentämisen JavaScript-ympäristössä.[18.]

JSON-olio koostuu pilkuilla erotetuista nimi-arvo pareista, jotka on suljettu aaltosulkeisiin, kuten esimerkkikoodista 1 näkee:

```
{"nimi1":arvo1, "nimi2":arvo2 ...}
```

Esimerkkikoodi 1. Esimerkki yksinkertaisesta JSON-oliosta.

Esimerkkikoodista 2 näkee, kuinka JSON-olioilla voidaan myös muodostaa monimutkaisempia olioita useammilla sisäkkäisillä kerroksilla:

```
{"user":
  {
   "userid": 444,
   "username": "user",
   "password": "Pa5Sw0rd",
   "groups": [ "admins", "users", "management"]
  }
}
```

Esimerkkikoodi 2. Sisäkkäisiä JSON-olioita.

JSON:a pidetään kevyempänä vaihtoehtona XML:lle ja molemmilla onkin laaja kannatus tiedon tallentamismuotona internetpalveluissa. JSON:n keveys suhteessa XML:ään tulee sen keskittymisestä tiedon tallentamiseen eikä niinkään sen muotoiluun. Tämän ansiosta JSON-tiedostot ovat lähtökohtaisesti pienempiä kuin XML:n tiedostot. Pienemmät paketit ovat tärkeä etu internetohjelmissa. JSON:a on myös helpompi jäsentää iOS-ympäristössä kuin XML:ää. JSON toimii myös paremmin epäsynkronisesti kuin XML. [19.]

JSON:n JavaScript-juurista nousee kuitenkin myös vakava huoli tietoturvasta. Nämä ongelmat keskittyvät JavaScripttulkin kykyyn suorittaa dynaamisesti JSON-tekstiä upotettuna JavaScriptinä. Tämä on helppo ja paljon käytetty tekniikka, jossa käytetään hyväksi JavaScriptin eval()-funktiota. Tekniikka altistaa ohjelman haitallisille koodeille. Haitalliset koodit ovat vakava ongelma etenkin, kun haetaan tietoa internetistä.

Sittemmin eval()-funktio on korvattu JSON.parse()-funktiolla, joka on tarkoitettu käyttämään nimenomaan JSON-tietoja eikä JavaScript-koodia. Käyttämällä uudempaa JSON.parse()-funktiota estetään JSON-tekstin suorittaminen dynaamisesti upotettuna JavaScriptinä. [20, s. 754–755.]

JSON.parse() on silti altis DoS-hyökkäyksille (Denial of Service) ja massatoimeksiantohaavoittuvuuksille (Mass assignment vulnerability). JSON.parse()-funktio voidaan pakottaa tekemään Ruby-symboleita (Ruby symbols) kohdejärjestelmään. Koska Ruby-symboleja ei poisteta turhina, tästä aiheutuu DoS-hyökkäys. Samaa tekniikkaa käyttämällä pystytään myös tekemään olioita kohdejärjestelmään. Näitä olioita voidaan sen jälkeen

käyttää kiertämään tietoturva ja suorittamaan massatoimeksiantohyökkäys. [21; 22.]

Selaimeen vietävän tiedon formaatiksi valittiin JSON, koska sille oli eniten tukea eri taulukko-ohjaimissa ja se on hyvin yleinen tiedon esitysmuoto web-sovelluksissa. Syitä sen suosioon on muun muassa se, että JSON:n esitysmuodon rakenne on yksinkertainen ja sitä on ihmisen helppo lukea ja kirjoittaa. JSON:a on myös helppo luoda ja jäsentää automaattisesti koodissa.

## 2.7 Taulukkomoduulin valinta

Esimerkkitoteutuksen tuli näyttää tietokannan tiedot selaimessa, ja helpoiten se onnistuisi käyttämällä valmiita moduuleita, jotka on tehty näyttämään tietoja taulukkona selaimessa. Vaihtoehtoja eri taulukkomoduuleille oli erittäin paljon Node.js-ympäristössä, joten oikean taulukkomoduulin valinta toteutukseen oli tärkeää. Valitsin kahdeksan eri moduulia tarkempaa tutkimista varten käyttäen hyväksi muiden tekemiä listauksia eri taulukkomoduulien paremmuudesta ja etsimällä muuten vain mielenkiintoisen oloisia taulukkomoduuleita. Niillä tuli myös olla kokeiltava demo internetissä, jotta niiden testaaminen olisi helppoa. Valitut moduulit olivat jsGrid, Backgrid.js, Angular UI Grid, SlickGrid, React Data Grid, OpenJS Grid v2, Grid ja dhtmlxGrid. Liitteessä 5 on erittely valittujen moduulien ominaisuuksista.

Valitut taulukkomoduulit olivat perustoiminnoiltaan hyvin samanlaisia, joten ensimmäinen karsinta tehtiin lisenssin sopivuuden perusteella. Tarkempaan tarkasteluun valituista moduuleista vain yksi ei käyttänyt MIT-lisenssiä vaan GNU GPL -lisenssiä. GPL-lisenssi vaatii, että kaikkien sen alaisia moduuleja käyttävien ohjelmien tulisi olla avointa lähdekoodia, mikä ei estä moduulin käyttämistä tässä insinöörityössä. Siitä voi kuitenkin olla haittaa, jos esimerkkitoteutuksena tehtävää ohjelmaa käytetään kaupallisessa ohjelmassa. Kyseessä oli ominaisuusrikkaan kaupallisen moduulin ilmaisversio, dhtmlxGrid, joten halusin pitää sen vielä mukana, jotta voisin vertailla sitä muihin ei-kaupallisiin moduuleihin.

Seuraavaksi karsin moduuleita dokumentoinnin tason mukaan. Kaikissa, paitsi Grid-moduulissa, dokumentoinnin taso oli hyvää tai kelpaavaa. Grid karsiutui siis pois.

Seuraava karsinta tehtiin selaintuen määrän perusteella. Kolmessa moduulissa ei ollut tukea mobiiliselaimille, ja lopuilla neljällä oli tuki lähes kaikille selaimille myös mobiilipuolella. Jäljelle jäivät jsGrid, Backgrid.js, Angular UI Grid ja kaupallinen dhtmlxGrid.

Angular UI Gridissä taulukon jakaminen useampaan sivuun oli vasta alpha-vaiheessa ja se ominaisuus on tärkeää, kun näytetään isoa tietomäärää pienellä mobiilinäytöllä.

Jäljelle jääneillä kolmella moduulilla oli paljon samoja ominaisuuksia, mutta vain jsGridillä oli lokalisointi tuettuna valmiina. Sen kehitystä myös jatkettiin vielä, toisin kuin Backgrid.js:n. Oli siis mahdollista, että jsGridiin tulisi vielä uusia ominaisuuksia. dhtmlxGridissä oli jo valmiina ominaisuuksia, joita ei löytynyt kummastakaan moduulista. Näistä esimerkkinä taulukon esittäminen puunäkymänä ja tiedon ulosvienti Excel- tai pdf-muodossa.

Valinta oli siis tehtävä jsGridin ja kaupallisen dhtmlxGridin välillä. Mielestäni parempi lisenssi ja lokalisaatio tuki oli tärkeämpiä kuin iso ominaisuuslista ja huonompi lisenssi. JsGridiin oli myös mahdollista saada teknistä tukea sen GitHub sivuilta, kun taas dhtmlxGridin tuesta joutuisi maksamaan. Valitsin siis jsGrid-moduulin käytettäväksi esimerkkitoteutuksessa.

# 3 Esimerkkitoteutuksessa huomioitavat asiat

## 3.1 Vaatimukset

Insinöörityön osana tehdyn esimerkkitoteutuksen vaatimuksena oli toteuttaa sovellus, joka hakee tietoa tietokannasta selaimeen ja mobiililaitteisiin helposti ja luettavasti. Tieto haettaisiin teollisuuden tuotannon seurantajärjestelmästä relaatiotietokannasta. Sovelluksen tulisi olla laajennettavissa, ja sen tulisi kestää tietokannan muutoksia. Käytettävyyteen tulisi kiinnittää huomiota sekä asiakkaan että ylläpidon kannalta. Lisenssiehtojen sopivuus ja tietoturva tulisi huomioida esimerkkitoteutuksessa.

## 3.2 Käytettävyys

Vaikka tehtävä ohjelma on vain esimerkkitoteutus, tulee sen käytettävyyteen silti kiinnittää huomiota. Käytettävyydellä yleisesti tarkoitetaan jonkin työkalun tai laitteen helppokäyttöisyyttä ja opittavuutta. Tietotekniikassa käytettävyydellä tarkoitetaan ISO-standardin [23] mukaan tätä:

> "Se vaikuttavuus, tehokkuus ja tyytyväisyys, jolla tietyt määritellyt käyttäjät saavuttavat määritellyt tavoitteet tietyssä ympäristössä".

Selkokielellä käytettävyys on se vaikuttavuus, tehokkuus ja tyytyväisyys, mitä käyttäjä kokee, kun hän käyttää jotain ohjelmaa.

Esimerkkitoteutusta tehdessä on siis huomioitava, millaista ohjelmaa on käyttää. Ohjelman ulkoasun tulisi olla siisti ja ilman ylimääräistä sotkua. Haluttujen toimintojen tulisi olla tehokkaasti käytettävissä. Laajennetussa käytettävyyden ISO-määritelmässä on vielä lisätty kohdat opittavuus, muistettavuus ja virheiden vähyys.

Tehtävän esimerkkitoteutuksen tulisi siis olla helposti opittavissa ja muistettavissa. Myös käyttäjän mahdollisuudet tehdä virheitä tulisi minimoida, kuitenkin niin, että ohjelman muu käytettävyys ei kärsisi siitä.

## 3.3 Tietoturva

Tietoturvalla tarkoitetaan yleensä tietoteknistä tietoturvaa, eli tietoturvaa, joka koskee jonkinlaista teknologiaa, aina taskulaskimesta, matkapuhelimin ja supertietokoneisiin. Tietoturvan tarkoituksena on suojata tietoja esimerkiksi luvattomalta käytöltä tai muokkaukselta. Haittaohjelmilta suojaaminen kuuluu myös tietoturvaan.

Vaikka esimerkkitoteutus ei tulekaan julkiseen internetiin, sen tulee silti noudattaa hyvän ohjelmointitavan mukaisia tietoturvavaatimuksia. Esimerkkitoteutuksessa tulisi välttää näyttämästä mitään tarpeetonta tietoa selaimessa. Ainoastaan julkiset tiedot saavat näkyä. Kaikki tietokantaan kirjautumiseen tarvittavat tiedot tulee pitää palvelimella, ja kaikki tietokantaa käsittelevät toiminnot tulee tehdä palvelimella. Selain ei saa olla suoraan yhteydessä tietokantaan, vaan selaimen tulee näyttää vain haetun taulukon tiedot.

## 3.4 Lisenssit

Ohjelmistonkäyttölisenssit ovat oikeudellinen väline, joilla säädellään ohjelmien käyttö- ja levitysoikeuksia. Tyypillinen ohjelmistonkäyttölisenssi antaa lisenssinsaajalle, tyypillisesti loppukäyttäjälle, luvan käyttää yhtä tai useampaa kopiota ohjelmistosta tavoilla, joista muuten voisi seurata tekijänoikeusrikkomus. [1.]

Tässä insinöörityössä pyrittiin käyttämään vapaita ohjelmistolisenssejä, koska ne sallivat ohjelmien käytön, kopioinnin, muokkaamisen ja levittämisen ilman erillisiä lupia. Melkein kaikki moduulit, joita tässä työssä tutkittiin tai käytettiin, olivat MIT-lisenssin alaisia. Ainoastaan yksi tutkituista taulukko moduuleista käytti GNU GPL -lisenssiä.

### MIT-lisenssi

MIT-lisenssi on vapaa ohjelmistolisenssi, joka rajoittaa ohjelmiston käyttöä erityisen vähän. Kuten nimestä voi päätellä, se on kehitetty Massachusetts Institute of Technologyssä (MIT). MIT-lisenssi antaa käyttäjälle täydet oikeudet ohjelmaan, kunhan lisenssin ehdot ja tekijänoikeudet löytyvät ohjelman kaikista kopioista. MIT-lisenssi sallii myös ohjelman käytön kaupallisessa ohjelmassa ilman, että lähdekoodin pitäisi olla avointa. [24.] Näiden kahden kohdan ansiosta lisenssi sopii erinomaisesti tämän insinöörityön tarkoituksiin.

MIT-lisenssi on edellä mainituista syistä erittäin suosittu, ellei jopa suosituin, lisenssi vapaissa ohjelmissa. Black Duckin omassa tietokannassaan tekemän tutkimuksen mukaan 32 % avoimen lähdekoodin projekteista käyttävät MIT-lisenssiä [25]. Huomioitavia projekteja, jotka käyttävät MIT-lisenssiä ovat esimerkiksi Node.js, jQuery, Ruby on Rails ja X Window System, jolle lisenssi oli alun perin tehty.

Lisenssiehdot:

> Copyright (c) <year> <copyright holders>
>
> Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:
>
> The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
>
> THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE. [24.]

GNU General Public License -lisenssi

GNU General Public License (GNU GPL tai pelkkä GPL) on GNU-hankkeen yleinen lisenssi. GNU-hanke on Richard Stallmanin MIT:ssä aloittama hanke, jonka tarkoituksena on taata tietokoneen käyttäjille vapaus ja kontrolli omiin tietokoneisiin ja ohjelmistoihin. Se perustuu siihen, että kaikilla on oikeus käyttää, kopioida, muuttaa ja jakaa edelleen ohjelmia ja niiden lähdekoodia, jos ohjelmat käyttävät GPL-lisenssiä. Lisenssi takaa myös sen, että GPL-koodiin pohjautuvat ohjelmat ovat myös avointa lähdekoodia, vaikka olisivatkin kaupallisia ohjelmia. [26]

GPL-lisenssi on myös erittäin suosittu vapaiden ohjelmien lisenssi, ja sitä käytetään esimerkiksi Linux-käyttöjärjestelmissä. Black Duckin tekemän tutkimuksen mukaan 7 % avoimen lähdekoodin projekteista käyttää uusinta GNU 3.0 -lisenssiä ja vanhempaa 2.0-lisenssiä 18 % [25].

Tämä lisenssi sopisi insinöörityön tekemiseen, mutta estäisi insinöörityön aikana tehdyn esimerkkitoteutuksen käytön kaupallisessa käytössä, jos ei olisi valmis jakamaan koko ohjelman lähdekoodia avoimena.

Lisenssiehdot:

> <one line to give the program's name and a brief idea of what it does.>
> Copyright (C) <year> <name of author>
>
> This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>. [26.]

# 4 Esimerkkitoteutus

## 4.1 Demo-versiot

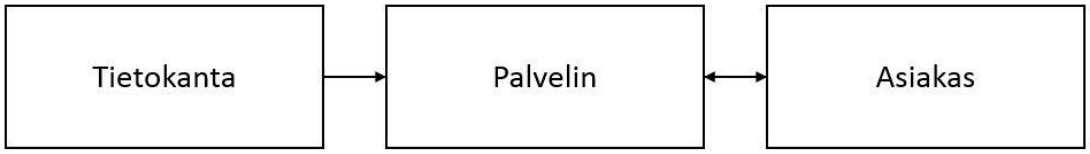
Insinöörityön esimerkkitoteutuksesta oli tarkoitus tehdä kaksi demoversiota ohjelmasta ennen lopullista ohjelmaa.

Ensimmäisessä demossa todennettiin, että valitulla toteutustavalla on mahdollista tehdä halutunlainen ohjelma. Data haettiin pienestä testitietokannasta OracleDB-moduulilla JSON-muotoon, jonka taulukko ohjain jsGrid sitten piirsi selaimeen taulukkona. Haut ja taulukon alustus oli kovakoodattu demoon. Verkkosivussa ei ollut minkäänlaista dynaamisuutta, vaan se vain näytti haetun taulukon.

Toisessa demossa luotiin tarvittava SQL-haku ja alustettiin taulukko oikeanlaiseksi dynaamisesti. Tietokantana käytettiin lopullista tietokantaa. Hakujen tulisi teoriassa toimia millä tahansa Oracle-tietokannasta haetulla taululla. Verkkosivu oli vieläkin täysin staattinen.

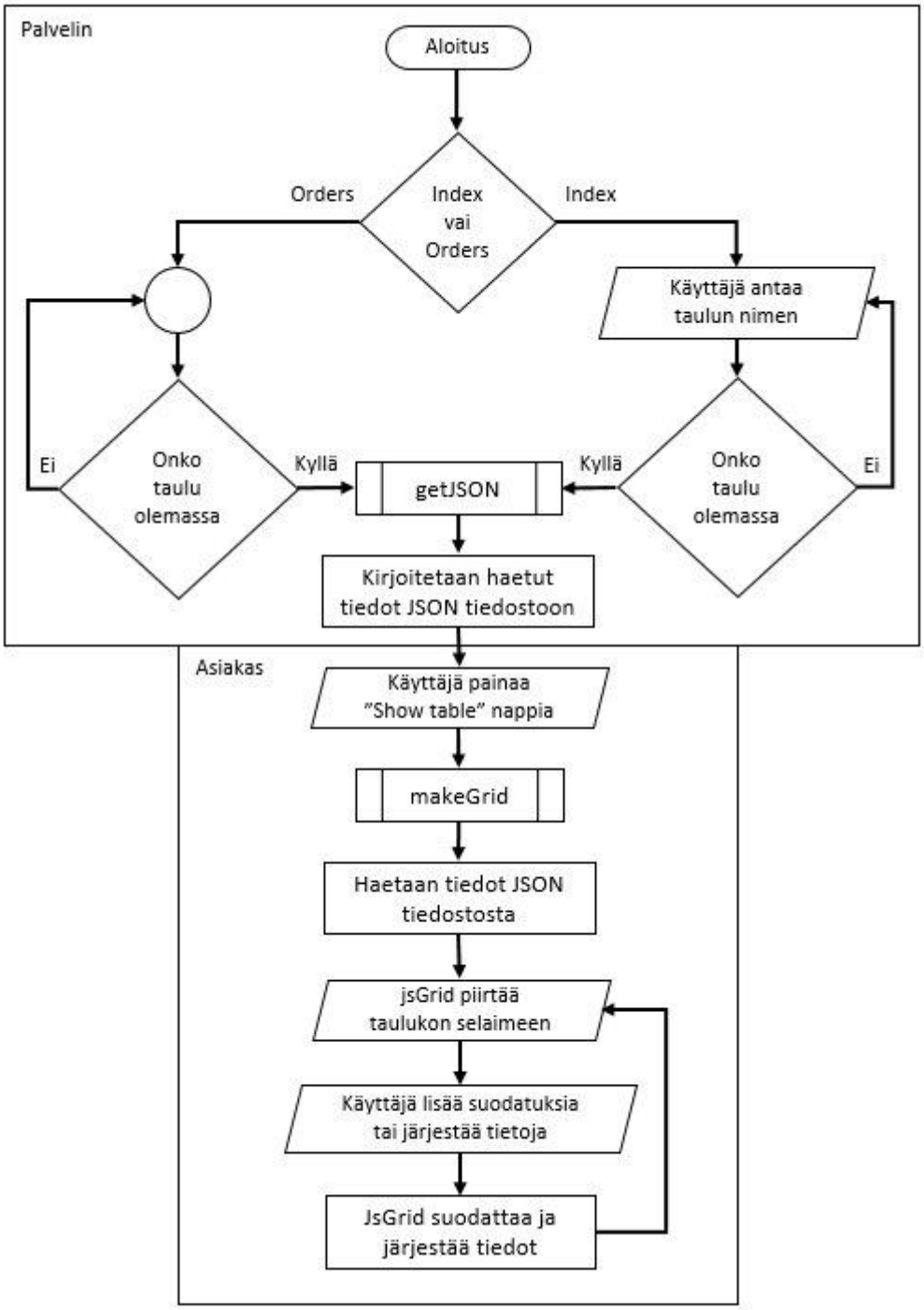
## 4.2 Ohjelman kuvaus

Ohjelman tulisi hakea ja näyttää selaimessa käyttäjän haluaman taulukko tietokannasta. Käyttäjä sai itse syöttää taulun nimen, ja ohjelma hakisi sen ennalta määritellystä tietokannasta. Tietokannan voi vaihtaa muokkaamalla palvelimella olevaa tiedostoa, jossa on tietokantaan kirjautumiseen tarvittavat tiedot. Kuvassa 1 näkyy systeemitason ohjelma kaavio tehdystä esimerkkitoteutuksesta.




Kuva 1. Esimerkkitoteutuksen systeemitason ohjelmakaavio.

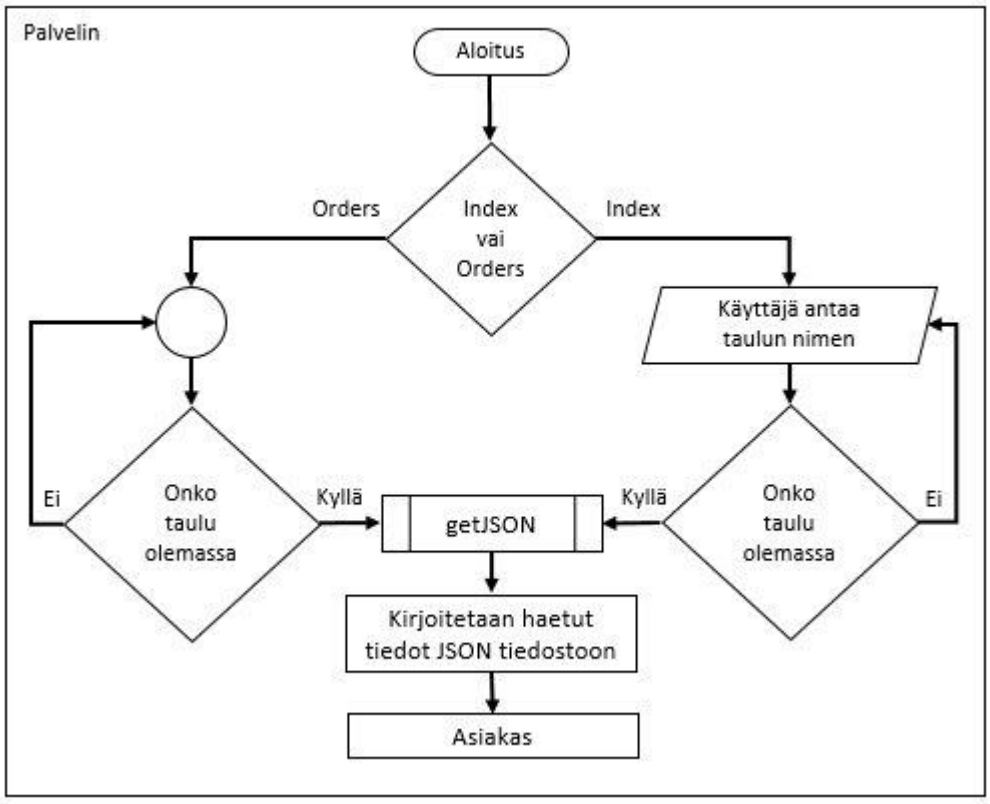
Kuten kuvasta 2 näkee, koostuu ohjelma kahdesta osasta, palvelimesta ja asiakaspäästä. Palvelimella tapahtuu kaikki tiedonhaku ja muokkaus, kun taas asiakkaan päässä tapahtuu tiedon ja verkkosivun näyttäminen selaimessa ja käyttäjän pyyntöjen välittäminen palvelimelle.




Kuva 2. Esimerkkitoteutuksen vuokaavio.

### 4.2.1 Palvelimen puoli

Palvelinpuoli koostuu kahdesta moduulista, server ja DBtoJSON. Server-moduulissa on palvelimen toiminnot ja DBtoJSON-moduulia käytetään tiedonhakuun tietokannasta. Lisäksi on myös tietokannan määrittelemiseen käytetty Dbconfig.js-tiedosto. Kuvassa 3 on vuokaavio palvelimen toiminnoista.




Kuva 3. Palvelimen vuokaavio.

Server.js

Server.js on kokonaisuudessaan liitteessä 1. Itse palvelin tehtiin käyttämällä suosittua Express-moduulia. Sillä palvelimen tekeminen oli erittäin helppoa, eikä se vaatinut kuin muutaman rivin koodia, kuten esimerkkikoodista 3 näkee.

```
var express = require('express');
var app = express();

//Sets the server to listen to port 3000
var port = 3000;    //Address is http://localhost:3000
app.listen(port, function () {
```

```
    console.log('Server on localhost:' + port);
});
```

Esimerkkikoodi 3. Palvelimen tekeminen Express-moduulilla.

Esimerkkikoodissa 4 käydään läpi, miten palvelimen tulisi vastata asiakkaan pyyntöön lähettämällä sille haluttu verkkosivu, tässä tapauksessa index.html ja orders.html, jotka löytyvät palvelimelle määritellystä julkisesta kansiosta. Index-sivulla voi itse määrittää taulun, josta tiedot haetaan taulukkoon. Sille pääsee, kun menee palvelimen aloitussivulle http://localhost:3000/. Orders-sivulla näytetään räätälöity taulukko, joka on muodostettu useasta taulusta. Sille pääsee osoitteesta http://localhost:3000/orders. HTML-tiedostojen sisältöön palataan tarkemmin asiakkaan puolta läpi käytäessä.

```
//Folder name of the public app
app.use(express.static(__dirname + '/public/'));

//Show Index page
//Runs when client calls the '/' page
app.get('/', function (req, res) {
    console.log("Client called")

    //Sends the index.html to client
    res.sendFile('public/index.html', {root: __dirname });
}); //End of app.get


//Show Orders page
//Runs when client calls the '/orders' page
app.get('/orders', function (req, res) {
    console.log("Client called")

    //Sends the orders.html to client
    res.sendFile('public/orders.html', {root: __dirname });
}); //End of app.get
```

Esimerkkikoodi 4. Verkkosivujen lähetys asiakkaalle.

Index-sivulla palvelimen tulisi hakea asiakkaan määrittelemä taulu tietokannasta. Esimerkkikoodissa 4 olevaan Index-funktioon pitäisi siis lisätä vielä toiminnot taulun nimen vastaanottamiseen ja kyseisen taulun hakemiseen tietokannasta. Taulun nimi muutetaan isoilla kirjaimilla kirjoitetuksi, jotta se vastaa OracleDB-moduulin vaatimuksia taulun nimestä. Nämä toiminnot näkyvät esimerkkikoodissa 5.

```
//Gets the table name from the query box in the browser
tableName = req.query.tablename;
```

```
//Gets the data from DB
//Puts the data to './public/data.json'
if (tableName != undefined) {
    tableName = tableName.toLocaleUpperCase();
    console.log("Get table: " + tableName);

    db.getJSON(tableName, function (err, data) {
        var fileName = './public/data.json';
        dataJSON = data;

        //Writes the data to './public/data.json'
        fs.writeFile(fileName, dataJSON, function (err) {
            if (err) return console.log(err);
            console.log('Data is in ' + fileName);
        });
    });
}
```

Esimerkkikoodi 5. Tietokantahakutoiminnot Index-funktioon.

Orders-sivulla taulut on jo ennalta määritelty, joten siihen ei tarvitse lisätä kuin dbOrders.getJSON-funktio taulujen hakuun. Esimerkkikoodissa 6 on Orders-funktioon lisätty dbOrders.getJSON-funktio.

```
//Show Orders page
app.get('/orders', function (req, res) {

    console.log("Client called")

    //Gets the data from DB
    //Puts the data to './public/data.json'

    console.log("Show orders");

    dbOrders.getJSON(function (err, data) {
        var fileName = './public/data.json';
        dataJSON = data;

        //Writes the data to './public/data.json'
        fs.writeFile(fileName, dataJSON, function (err) {
            if (err) return console.log(err);
            console.log('Data is in ' + fileName);
        });
    });

    //Sends the orders.html to client
    res.sendFile('public/orders.html', {root: __dirname });
}); //End of app.get
```

Esimerkkikoodi 6. Valmis Orders-funktio.

GetJson-funktiot hakevat tiedot kannasta JSON-muotoon, ja ne tallennetaan julkiseen hakemistoon data.json nimellä. Db.getJson-funktio on määritelty sivulla 25 esiteltävässä DBtoJSON-tiedostossa, ja dbOrders.getJson-funktio on määritelty getOrders-tiedostossa. Fs-moduuli on Node.js:n sisäänrakennettu moduuli tiedostojen lukemiseen ja kirjoittamiseen. Jotta kaikki funktiot toimisivat, tulisi server-moduulin alkuun lisätä niiden vaatimiset. Myös tableName- ja dataJSON-muuttujat pitäisi määritellä. Tarvittavat server-moduulin alustukset ovat esimerkkikoodissa 7.

```
var db = require('./dbtoJSON');
var dbOrders = require('./getOrders');
var fs = require('fs');

var tableName = '';
var dataJSON = '';
```

Esimerkkikoodi 7. Server-moduulin alustukset.

Asiakaspuolella oleva taulukon piirtämiseen käytetty grid_jsGrid.js tarvitsee toimiakseen gridConf.js-tiedoston, joten se tulisi tehdä aina palvelimen käynnistyessä, jotta vältytään tiedoston puuttumisen aiheuttamilta virheiltä. Tämä tehdään esimerkkikoodissa 8. GridConf-tiedoston sisältöä käydään läpi myöhemmin tässä luvussa ja grid_jsGrid-tiedostoa luvussa 4.2.2 Asiakaspuoli.

```
//Makes a empty grid config file
var gridConfFile = './public/gridConf.js';
var text = 'data = [{"name":"numberOfColumns","sorter":0}]' ;
fs.writeFile(gridConfFile , text, function (err) {
    if (err) return console.log(err);
});
```

Esimerkkikoodi 8. Tyhjän taulukkokonfiguraatiotiedoston tekeminen.

DBtoJSON

DBtoJSON-moduuli koostuu neljästä funktiosta: getTable, joka hakee taulukon sarakkeiden nimet ja sarakkeiden ja rivien lukumäärän, getData, joka hakee tiedot tietokannasta JSON-muotoon, doRelease, joka vapauttaa yhteyden tietokantaan, sekä kaiken yhdistävä ja ulkoapäin kutsuttava getJSON. Liitteessä 2 on DBtoJSON.js-tiedosto kokonaisena, ja sieltä on helpompi nähdä, miltä funktiot näyttävät kokonaisuudessaan.

Yhteys tietokantaan muodostetaan Oraclen itse tekemällä ajurilla OracleDB. Se tarvitsee toimiakseen DBconfig.js-tiedoston, jossa on määritelty tietokannan tiedot. Esimerkkikoodissa 9 näkyy, mitä DBconfig.js-tiedostossa on määritelty.

```
module.exports = {
  user: process.env.NODE_ORACLEDB_USER || "user",
  password: process.env.NODE_ORACLEDB_PASSWORD || "password",
  connectString: process.env.NODE_ORACLEDB_CONNECTIONSTRING ||
"0.0.0.0/example",
};
```

Esimerkkikoodi 9. DBconfig.js-tiedosto.

OracleDB ja Dbconfig tulisi lisätä moduulin vaatimuksiin. Myös tiedostojen lukemiseen ja kirjoittamiseen käytettävä fs-moduuli vaaditaan. Koska getJSON-funktiota kutsutaan tämän moduulin ulkopuolelta, se tarvitsee lisätä moduulin ulkoisiin funktioihin. Esimerkkikoodissa 10 on tarvittavat alustukset.

```
var oracledb = require('oracledb');
var dbConfig = require('./dbconfig.js');
var fs = require('fs');

module.exports.getJSON = getJSON;
```

Esimerkkikoodi 10. DBtoJSON-moduulin alustukset.

Koska getJSON-funktio kutsuu getTable- ja getData-funktioita, on parempi käydä ensin ne läpi. Esimerkkikoodissa 11 oleva GetTable-funktio tarvitsee parametrinaan taulukon nimen ja epäsynkronisuuden takia takaisinkutsufunktion callback. Sitä kutsutaan, kun getTable on saanut haettua kaikki callback-funktion tarvitsemat tiedot. Callback-funktion ensimmäinen parametri on tarkoitettu virheille, joten se on null. Funktion viimeinen parametri connection on se auki oleva yhteys, jota funktio käytti. Se annetaan eteenpäin

callback-funktiossa, jotta yhteys katkaistaan vasta, kun halutut tiedot on saatu ja yhteyttä ei enää tarvita.

```
//Gets the table layout
//Returns column names and grid configuration array and
//the number of columns and rows
function getTable(tableName, callback) {

    var columnNames = [];
    var numberOfColumns = 0;
    var numberOfRows = 0;
    var gridConf = [];

    callback(null, columnNames, numberOfColumns, numberOfRows,
            gridConf, connection);
}
```

Esimerkkikoodi 11. GetTable-funktio.

Yhteys tietokantaan otetaan käyttämällä esimerkkikoodissa 12 olevaa OracleDB:n funktiota getConnection, jolle annetaan parametreinä DBConfig:n tiedot ja funktio, jossa suoritetaan halutut tietokanta haut. Esimerkkikoodiin 12 lisättävien hakujen tekeminen käydään seuraavaksi läpi.

```
//Gets column names and grid configuration array and
//the number of columns and rows
oracledb.getConnection(
  {
    user          : dbConfig.user,
    password      : dbConfig.password,
    connectString : dbConfig.connectString
  },
  function(err, connection) {
    if (err) {
      console.log(err);
      throw err;
    }
    //Queries:
    //Get the number of columns in table to numberOfColumns
    //Get the number of rows in table to numberOfRows
    //Get the column names and types in table to gridConf array
  }
)
```

Esimerkkikoodi 12. getConnection-funktio.

Haut tehdään käyttämällä connection.execute-funktiota ja antamalla sille haluttu SQL-haku. Haun tulos löytyy results-muuttujasta. Hauissa käytetty tableName on asiakkaan

antama taulukon nimi. Esimerkkikoodissa 13 olevassa ensimmäisessä haussa haetaan sarakkeiden lukumäärä SQL-haulla ”SELECT COUNT(column_id) FROM user_tab_columns WHERE table_name = tableName”. Tulos tallennetaan numberOfColumns-muuttujalle.

```
//Gets the number of columns in table to numberOfColumns
connection.execute(
  'SELECT COUNT(column_id) ' +
  'FROM user_tab_columns ' +
  'WHERE table_name = \'' + tableName + '\'',
  {},
  function(err, results) {
    if (err) {
      console.log(err);
      throw err;
    }

    if (results != undefined) {
      numberOfColumns = results.rows[0][0];
    }
  }
);
```

Esimerkkikoodi 13. Taulukossa olevien sarakkeiden lukumäärän hakeminen numberOfColumns-muuttujalle.

Seuraavaksi haetaan rivien lukumäärä haulla ” SELECT COUNT(*) FROM tableName”. Tulos tallennetaan numberOfRows-muuttujaan, kuten esimerkkikoodissa 14 näkyy.

```
//Gets the number of rows in table to numberOfRows
connection.execute(
  'SELECT COUNT(*) ' +
  'FROM ' + tableName,
  {},
  function (err, results) {
    if (err) {
      console.log(err);
      throw err;
    }

    if (results != undefined) {
      numberOfRows = results.rows[0][0];
    }
  }
);
```

Esimerkkikoodi 14. Taulukossa olevien rivien lukumäärän hakeminen numberOfRows-muuttujalle.

Esimerkkikoodissa 15 oleva viimeinen haku tehdään komennolla ”SELECT column_name,data_type FROM user_tab_columns WHERE table_name = tableName”. Haun tulokset tallennetaan grifConf-taulukkoon ja columnNames-muuttujalle. Aikaisemmin tehty callback-funktio tulisi siirtää tämän haun sisään.

```
//Gets the column names and types in table to gridConf and
//columNames
connection.execute(
  'SELECT column_name,data_type ' +
  'FROM user_tab_columns ' +
  'WHERE table_name = \'' + tableName + '\'',
  {},
  function (err, results) {
    if (err) {
      console.log(err);
      throw err;
    }

    //Puts the number of columns to the first value of gridConf
    gridConf[0] = {
      name: 'numberOfColumns',
      sorter: numberOfColumns
    };

    //Puts the name and type to gridConf and columnNames
    if(results != undefined) {
      var type = '';

      for (var i = 0; i < numberOfColumns; i++) {
        if (results.rows[i][1] == 'NUMBER') {
          type = 'number';
        }
        else {
          type = 'string';
        }

        gridConf[i + 1] = {
          name: results.rows[i][0],
          sorter: type
        };

        columnNames[i] = {
          name: results.rows[i][0],
          sorter: type
        };
      }
    }
    else{
      console.log("ERROR: Results undefined!")
```

```
    }
    callback(null, columnNames, numberOfColumns, numberOfRows,
             gridConf, connection);
  }
);  //End of connection.execute
```

Esimerkkikoodi 15. Taulukoissa olevien sarakkeiden nimien ja tyyppien haku.

Funktio getData hakee halutun taulukon tiedot tietokannasta ja palauttaa tiedot JSON-muodossa. Parametreinä se tarvitsee rivien lukumäärän, taulukon nimen, sarakkeiden nimet, sarakkeiden lukumäärän ja callback-funktion. Yhteys tietokantaan muodostetaan samalla tavalla kuin getTable-funktiossa.

```
//Gets data from DB
//Returns a JSON format string of the data
function getData(nroOfRows, tableName, columnNames,
                 numberOfColumns, callback) {

  //Builds and makes a query to DB
  //and returns the data in JSON format string
  oracledb.getConnection(
    {
      user          : dbConfig.user,
      password      : dbConfig.password,
      connectString : dbConfig.connectString
    },
    function(err, connection) {
      if (err) {
        console.log(err);
        throw err;
      }

      //query = SELECT columnName1,columnName2... FROM tableName

      //Makes the query to DB
      connection.execute(
        query,
        {},
        function(err, results) {
          if (err) {
            console.log(err);
            throw err;
          }
          callback(null, jsonString, connection);
        }
      );  //End of connection.execute
    }
  );  //End of oracledb.getConnection
}
```

Esimerkkikoodi 16. getData-funktio.

Mutta koska SQL-haku riippuu taulukosta, pitää haku rakentaa sarakkeiden nimistä, kuten esimerkkikoodissa 17 on tehty.

```
//Build query using column names
var query = 'SELECT ';

//Loops through the column names
for(var i = 0; i < tableLength; i++){
  query += table[i]["name"];

  if(tableLength!= i +1){
    query += ', ';
  }
}
query += ' FROM ' + tableName;
//query = SELECT columnName1, columnName2… FROM tableName
```

Esimerkkikoodi 17. SQL-haun rakentaminen sarakkeiden nimistä.

Haun tulos muokataan JSON-formaattiin, ja siitä poistetaan ylimääräiset välilyönnit ja lainausmerkit, koska ne sotkevat JSON-tiedoston tekemistä. Tämä on tehty esimerkkikoodissa 18. JSON-formaatti on tämän näköinen: [{"name1":"value1","name2":"value2"}],{...}].

```
//Loops through the query result and puts it in
//JSON format string
for(var j = 0; j < nroOfRows; j++) {
  values = "{";
  for (var i = 0; i < tableLength; i++) {
    name = table[i]["name"];
    value = results.rows[j][i];

    //Removes extra spaces and " from result string
    if (value != null){
      str = results.rows[j][i].toString();
      str = str.replace(/"/g, null);
      str = str.replace(/\s+/g," ");
      value = str;
    }

    values += "\"" + name + "\":\"" + value + "\"";
    //"name":"value"

    //Adds comma if not last value else ends the object
    if ((i+1) < tableLength)
    values +=  ",";
    else
    values +=  "}";
  }
```

```
  jsonString += values;
  //[{"name1":"value1","name2":"value2"},{...}

  //Adds a comma if not last object
  if(j+1 < nroOfRows)
    jsonString += ",";
}

//This ends the JSON string
jsonString += "]";
//[{"name1":"value1","name2":"value2"}],{...}]
```

Esimerkkikoodi 18. Hakutuloksen läpikäyminen ja tallentaminen JSON-formaattiin.

getData-funktion alkuun pitää vielä lisätä tarvittavat muuttujan alustukset. Tarvittavat alustukset näkyvät esimerkkikoodissa 19. jsonString-muuttujalle laitetaan JSON-formaatin vaatima ensimmäinen merkki ”[”.

```
var variable = [];
variable = "[";
var values = "";
var name = "";
var value = "";
var str = "";
```

Esimerkkikoodi 19. getData-funktion alustukset.

Yhteys tietokantaan katkaistaan funktiossa doRelease käyttämällä OracleDB:n funktiota connection.release. doRelease-funktio on esimerkkikoodissa 20. Connection-muuttuja saadaan parametrinä doRelease-funktiolle. Funktiota kutsutaan getJSON-funktiossa, kun halutut tiedot on saatu ja yhteyttä ei enää tarvita.

```
//Releases the connection to DB
function doRelease(connection)
{
  connection.release(
    function(err) {
      if (err) {
        console.error(err.message);
      }
      console.log('Connection released.');
    }
  );
}
```

Esimerkkikoodi 20. doRelease-funktio.

Esimerkkikoodissa 21 olevassa funktiossa getJSON kutsutaan ensin getTable-funktiota, ja kun siltä on saatu callback-funktion vastaus, se kirjoitetaan julkisessa hakemistossa olevaan gridConf.js-tiedostoon.

```
//Gets the grid layout and data from DB
//Puts the grid layout to './public/gridConf.js'
//Returns data as JSON format string
function getJSON(tName, callback) {

  var JSONData = '';
  var table = {};
  var tableLength;
  var tableName = tName;
  var rows = '';
  var data = '';

  var rowsToGet = 1000;
  var getAll = false;

  //Gets the grid layout and data from DB
  //Puts the grid layout to './public/gridConf.js'
  //Returns data as JSON format string
  getTable(tableName, function (err, columnNames,
           numberOfColumns, numberOfRows, gridConf, connection){

    table = columnName;
    tableLength = numberOfColumns;
    rows = numberOfRows;

    doRelease(connection);

    var fileName = './public/gridConf.js';
    data = 'data = ';
    data += JSON.stringify(gridConf);
    data += ';';

    //Writes the grid layout to './public/gridConf.js'
    fs.writeFile(fileName, data, function (err) {
      if (err) return console.log(err);
      console.log('Grid config is in ' + fileName);
    });
```

Esimerkkikoodi 21. getJSON-funktio.

GetData-funktiota kutsutaan vain, jos getTable-funktion avulla saatu rivien lukumäärä on yli nolla eli kyseinen taulukko on olemassa. GetJSON:n alussa alustettavaa muuttujaa rowsToGet käytetään siihen, jos ei haluta hakea joka ikistä riviä tietokannasta. GetAll-muuttujalla määritellään, haetaanko kaikki muuttujat vai käytetäänkö rowsToGet-muuttujan arvoja. Ennen kuin haettavien rivien määrä määritellään OracleDB:lle, tarkistetaan,

että haettavien rivien määrä ei ylitä olemassa olevien rivien määrää. Tämä kaikki näkyy esimerkkikoodissa 22.

```
//Gets the data if the table has more than 0 rows i.e. it exists
if(rows > 0) {
  if (getAll)
    rowsToGet = rows;
  else if (rowsToGet > rows)
    rowsToGet = rows;

  //Sets the maximum rows that the query can return to rowsToGet
  if (rowsToGet > 0)
    oracledb.maxRows = rowsToGet;

  console.log("Rows to get: " + rowsToGet);
  console.log("Rows in DB: " + rows);

  //Gets the data from DB
  //Returns the data in JSON format string
  getData(rowsToGet, tableName, table, tableLenght,
          function(err, jsonString, connection) {
```

Esimerkkikoodi 22. getData-funktion haettavien rivien määrän tarkistus.

Esimerkkikoodissa 23 käydään läpi, mitä tehdään, kun getData-funktion takaisinkutsusta function(err, jsonString, connection) saadaan vastaus. Silloin poistetaan jsonString-muuttujasta kaikki rivin vaihdot, jotta ne eivät sotke tallennusta JSON-formaattiin. Tämän jälkeen getDatan käyttämä yhteys suljetaan ja JSONData lähetetään getJSON-funktion callback-funktion avulla eteenpäin siihen kohtaan, missä getJSON:a kutsuttiin.

```
    //Removes all row changes and puts the data to JSONData
    JSONData = jsonString.replace(new RegExp('\r\n|\n|\r',
               'gm'), null);
    doRelease(connection);
    callback(null, JSONData);
  }); //End of getData
```

Esimerkkikoodi 23. Rivinvaihtojen poisto ja tiedon tallentaminen JSONData-muuttujaan.

Esimerkkikoodissa 24 näkyvä GridConf.js-tiedoston tulisi näyttää ensimmäisessä kohdassa sarakkeiden lukumäärä ja seuraavissa kohdissa sarakkeiden nimet ja niiden tyyppi. GridConf-tiedostoa käytetään myöhemmin asiakkaan puolella määrittelemään, millainen taulukko tulisi piirtää.

```
data = [
{"name":"numberOfColumns","sorter":2},
```

```
{"name":"NAME1","sorter":"string"},
{"name":"NAME2","sorter":"number"},
{...}
];
```

Esimerkkikoodi 24. gridConj.js-tiedosto.

Data.json-tiedostossa pitäisi olla tietokannasta haetut tiedot JSON-formaatissa.

```
[{"name1":"value1","name2":"value2"}],{...}]
```

Esimerkkikoodi 25. Data.json-tiedosto.

getOrders

GetOrders on muuten samanlainen kuin DBtoJson, mutta siinä taulukko ja haku on ennalta määrätty eli getTable funktiota ei tarvita ollenkaan vaan se on korvattu koodissa tehtävällä taulukon alustuksella, joka näkyy esimerkkikoodissa 26.

```
var tableLength = 17;
var gridConf = [];

gridConf[0] = {
    name: 'numberOfColumns',
    sorter: tableLength
};
gridConf[1] = {
    name: 'ORDER_CODE',
    sorter: 'string'
    };
gridConf[2] = {
    name: 'ORDER_ITEM',
    sorter: 'number'
    };
...
gridConf[17] = {
    name: 'CITY',
    sorter: 'string'
};
```

Esimerkkikoodi 26. getOrders-moduulin taulukon alustus.

GetData-funktiossa SQL-hakua ei tarvitse rakentaa paloista, vaan se on vain määritelty koodissa, niin kuin esimerkkikoodissa 27 näkyy.

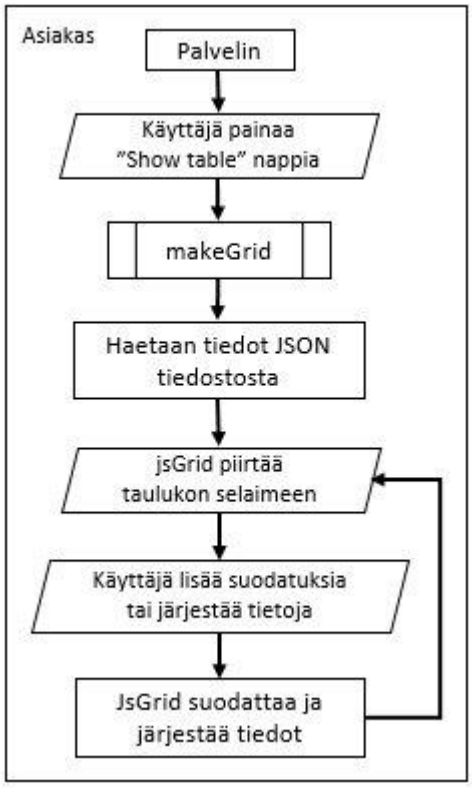
```
//Build query
var query =
```

```
'SELECT DISTINCT oh.ORDER_CODE, orderitem.ORDER_ITEM, ' +
'orderitem.PRODUCT_CODE, product.PRODUCT_NAME, ' +
'orderitem.QTY_ORDERED, s1.SCODE_NAME_SHORT as type, ' +
's2.SCODE_NAME_SHORT as order_status, ' +
'to_char(orderitem.DATE_DELIVERY,\'YYYY-MM-DD\') ' +
...
'order by DATE_DELIVERY DESC'
;
```

Esimerkkikoodi 27. getOrders-moduulin SQL-haku.

### 4.2.2 Asiakaspuoli

Ohjelman asiakaspuoli koostuu kahdesta JavaScript-moduulista: grid_Jsgrid.js ja grid_Orders.js, jotka määrittelevät ja piirtävät taulukot verkkosivuille. HTML-tiedostot määrittelevät millaiset verkkosivut asiakkaalle näytetään ja mitä JavaScript-moduuleita käytetään. Verkkosivun ulkoasu on määritelty style.css-tiedostossa. Grid_jsGrid.js on liitteessä 3 ja index.html liitteessä 4 kokonaisuudessaan. Kuvassa 4 näkyy asiakaspuolen vuokaavio.




Kuva 4. Asiakaspään vuokaavio.

Grid_jsGrid.js

Grid_jsGridin sisällä olevaa makeGrid-funktiota kutsutaan, kun selaimessa painetaan ”Show table”-nappia. Funktio on esimerkkikoodissa 28. Funktion alussa haetaan taulukon tiedot gridConf.js,tiedostosta ja ne asetetaan field-muuttujalle jsGridin haluamassa muodossa.

```
//Makes the grid component for the browser using jsGrid
function makeGrid() {
  //Requires the grid config file to work
  require(['../gridConf'], function (config) {});

  var field = [];
  var gridConfig = data;
  var columns = {name:[], width:[],sorter:[]};
  var columnWidth = 100;

  //Sets values to column object from gridConfig
  for (var i = 0; i <= gridConfig[0].sorter; i++) {
      columns.name[i] = gridConfig[i].name;
      columns.sorter[i] = gridConfig[i].sorter;
      columns.width[i] = columnWidth;
  }

  //Sets values to fields array from gridConf
  for (var i = 1; i <= numberOfColumns; i++) {
      field.push({
          name: gridConfig[i].name,
          width: columnWidth,
          sorter: gridConfig[i].sorter,
          type: function () {
            if (gridConfig[i].sorter == "string") return "text";
            else return "number";
          },
          align: "center"
      });
  }
```

Esimerkkikoodi 28. makeGrid-funktio.

$("#jsGrid").jsGrid piirtää taulukon selaimeen. Tämä funktio on esimerkkikoodissa 29. Controller-kohdassa määritellään, miten ja mistä tieto haetaan. Tässä tapauksessa tieto haetaan käyttämällä AJAXia, ja tiedon muoto on JSON. Tieto sijaitsee yleisessä hakemistossa data.json-tiedostossa. Filter-muuttujaa käytetään, jos haettavaa tietoa halutaan suodattaa. Suodatusta ei käytetty grid_Jsgridissä, koska sen saaminen toimimaan dynaamisesti eri taulukoissa osoittautui liian aikaa vieväksi. Suodatusta käytetään kuitenkin grid_Ordersissa ja suodattimen toiminnot käydään läpi grid_Orders-kohdassa.

```
//Renders the grid to browser
$("#jsGrid").jsGrid({
  autoload: true,

  controller: {
    loadData: function (filter) {
    return $.ajax({
      type: "GET",
      url: "../data.json",
      data: filter,
      dataType: "json"
    });
    }
  },
```

Esimerkkikoodi 29. jsGrid-funktio.

Funktion lopussa on erinäisiä asetuksia, joilla taulukkoa pystyy hienosäätämään. Asetuksen näkyy esimerkkikoodissa 30. Aikaisemmin tehty field-muuttuja sijoitetaan fields-kohtaan, jotta taulukon tiedot saadaan taulukko-ohjaimelle asti.

```
  //Used for grid setup
  width: "100%",
  height: "auto",
  pageSize: 10,
  pageButtonCount: 5,

  //Used for turning features on and off
  filtering: false,
  inserting: false,
  editing: false,
  sorting: true,
  paging: true,
  deleting:false,

  //Fields contains the layout of the grid
  //i.e. the column names, width, sorters, types and alignments
  fields: field

}); //End of $("#jsGrid").jsGrid
```

Esimerkkikoodi 30. jsGrid-funktion asetuksia.

Grid_Orders.js

Grid_Ordersin sisällä olevaa makeGrid-funktiota, joka löytyy esimerkkikoodista 31, kutsutaan samalla tavalla kuin aiemmin eli silloin, kun selaimessa painetaan ”Show table”-nappia. Funktion alussa haetaan taulukon tiedot gridConf.js-tiedostosta ja ne asetetaan field-muuttujalle jsGridin haluamassa muodossa.

```
//Makes the grid component for the browser using jsGrid
function makeGrid() {
    //Requires the grid config file to work
    require(['../gridConf'], function (config) {});

    var field = [];
    var gridConfig = data;
    var columns = {name:[], width:[],sorter:[], type:[]};
    var columnWidth = 175;

    //Sets values to column object from gridConfig
    for (var i = 0; i <= gridConfig[0].sorter; i++) {
        columns.name[i] = gridConfig[i].name;
        columns.sorter[i] = gridConfig[i].sorter;
        columns.width[i] = columnWidth;

        if (gridConfig[i].sorter == "string")
            columns.type[i] = "text";
        else
            columns.type[i] = "number";

    }

    //Sets values to fields array from column
    for (var i = 1; i <= gridConfig[0].sorter; i++) {
            field.push({
                name: columns.name[i], width: columns.width[i],
                sorter: columns.sorter[i],
                type: columns.type[i],
                align: "center"
            });
    }
```

Esimerkkikoodi 31. makeGrid-funktio.

$("#jsGrid").jsGrid piirtää edelleen taulukon selaimeen ja muutenkin toimii samalla tavalla kuin grid_Jsgridissä, mutta tässä siihen on lisätty suodatustoiminto. Suodatusasetukset lisätään controllerin loadDatan sisälle. Suodatusasetukset on esimerkkikoodissa 32.

```
loadData: function(filter) {
    var d = $.Deferred();

    // server-side filtering
    $.ajax({
        type: "GET",
        url: "../data.json",
        data: filter,
        dataType: "json"
    }).done(function(result) {

    // client-side filtering
    result = $.grep(result, function(item) {
        return //Filters for all of the columns
    });

    d.resolve(result);
})
```

Esimerkkikoodi 32. jsGrid-funktion suodatusasetukset.

Suodattimet pitää asettaa kaikille sarakkeille. Sarakkeen suodatusasetukset ovat hieman erilaiset riippuen siitä, että ovatko sarakkeen arvot tekstiä vai numeroita. Tekstille käytetään indexOf-käskyä ja numeroille suurempi/pienempi/yhtä suuri kuin -vertailuja. Tässä työssä numeroille käytettiin suurempi tai yhtä suuri kuin -vertailua. Koska on mahdollista, että suodattimelle annatuilla arvoilla ei löydy mitään, pitää palautukseen myös määrittää se. Asiakaspään suodatus on esimerkkikoodissa 33.

```
// client-side filtering
result = $.grep(result, function(item) {
    return
      (!filter.textColumn || // Returns empty
       item.textColumn.indexOf(filter.textColumn) > -1)
    &&
      (!filter.numberColumn || // Returns empty
       item.numberColumn >= filter.numberColumn)
});
```

Esimerkkikoodi 33. Asiakaspään suodatus

Funktion lopussa olevissa asetuksissa suodatus asetettiin päälle, kuten esimerkkikoodista 34 näkee.

```
 //Used for turning features on and off
  filtering: false,
  inserting: false,
  editing: false,
  sorting: true,
  paging: true,
  deleting:false,
```

Esimerkkikoodi 34. Ominaisuuksien päälläoloasetukset.

HTML-tiedostot

Esimerkkikoodissa 35 on HTML-tiedostojen head-kohdan ensimmäinen osa, missä määritellään sivun otsikko ja sivun käyttämät css-ulkoasutiedostot sekä käytettävä merkistöstandardi UTF-8.

```
<head>
  <title>Title</title>
  <link rel="stylesheet" type="text/css"
        href="stylesheets/style.css" />
  <link type="text/css" rel="stylesheet"
        href="modules/jsgrid-1.4.1/jsgrid.min.css" />
  <link type="text/css" rel="stylesheet"
        href="modules/jsgrid-1.4.1/jsgrid-theme.min.css" />
  <meta charset="UTF-8">
```

Esimerkkikoodi 35. Sivun otsikon, ulkoasun ja merkistöstandardin määrittely HTML-tiedostossa.

Esimerkkikoodissa 36 oleva HTML-tiedoston script-osiossa määritellään, mitä moduuleita ja JavaScript-koodia käytetään. Jquery ja jsgrid-1.4.1 ovat jsGrid-taulukkomoduulin käyttöön tarvittuja moduuleita. Requirejs-moduulia tarvitaan, jotta JavaScript-tyyppistä require-käskyä voi käyttää asiakkaan puolen JavaScript-koodissa. Grid_Jsgrid.js on tiedosto, jossa haluttu taulukko määritellään ja piirretään.

```
  <script src="modules/jquery/dist/jquery.min.js"></script>
  <script src="modules/jsgrid-1.4.1/jsgrid.min.js"></script>
  <script data-main="gridConf"
          src="modules/requirejs/require.js"></script>
  <script type="text/javascript"
  src="javascripts/grid_Jsgrid.js" ></script>
</head>
```

Esimerkkikoodi 36. HTML-tiedoston head-kohdan script-osio.

Body-kohdassa määritellään, mitä itse verkkosivulla näkyy. Esimerkkikoodissa 37 oleva Index.html-tiedoston form-kohdan sisällä määritellään ja piirretään tekstinsyöttölaatikko ja nappulat taulukon nimen syöttämistä varten. ”Get data”-nappi hakee halutun taulukon tiedot tietokannasta. ”Reset” nappi tyhjentää tekstikentän. Tätä kohtaa ei tarvita orders.html tiedostossa, koska siinä vain haetaan ennalta määritelty taulukko. Kuvassa 5 näkyy, miltä Index-sivut näyttävät tässä vaiheessa.

```
<body>

  <form>
    Table name:<br>
    <input type="text" name="tablename"
           placeholder="Table name"><br>
    <input type="submit" value="Get data">
    <input type="reset">
  </form>
```

Esimerkkikoodi 37. Index.html-tiedoston form-osio.

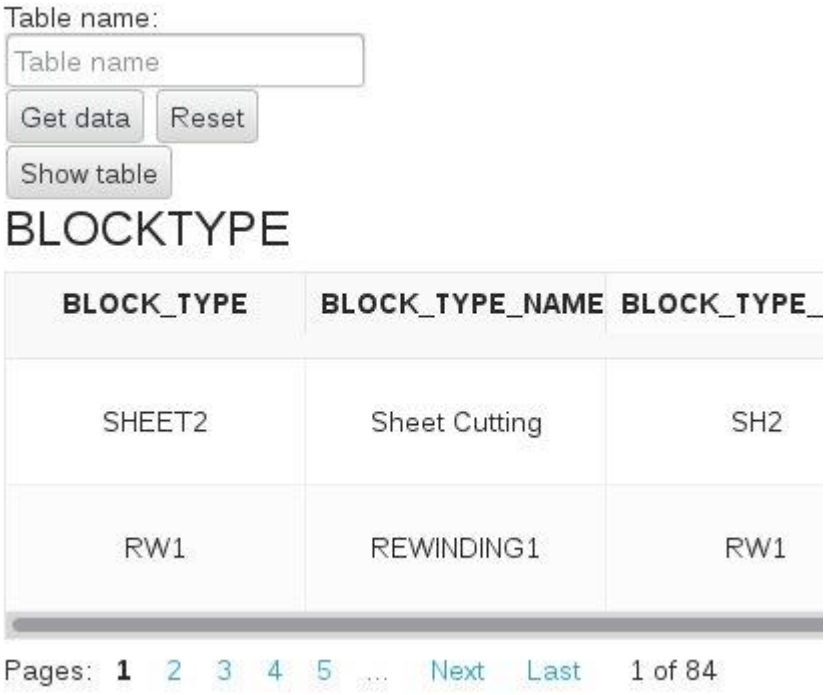

Kuva 5. Index-sivut formin käsittelyn jälkeen.

Esimerkkikoodissa 38 näkyvän container-kohdan sisälle tehdään nappula ”Show table”, jota painamalla kutsutaan Grid_jsGridissä olevaa makeGrid-funktiota, joka tuo näkyviin haetun taulukon. Kohtaan ” ” piirretään taulukko. Taulukko on määritelty grid_jsGrid-tiedostossa. Kuvissa 6, 7 ja 8 näkyy, miltä Index- ja Orders-verkkosivut näyttävät valmiina.

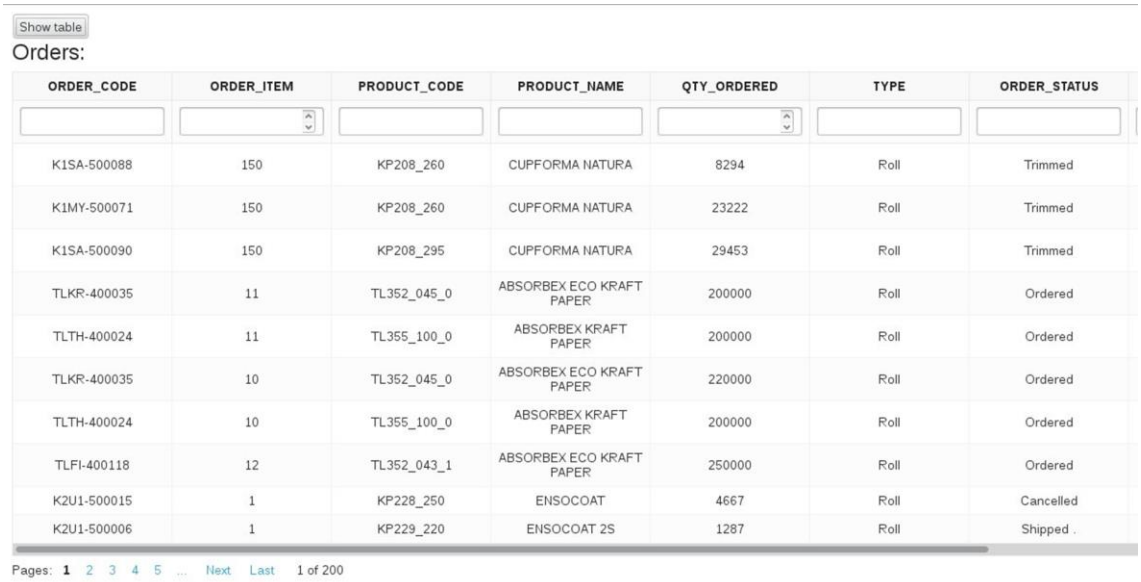
```
<div class="container">

  <button onclick="makeGrid()">Show table</button>

  <h1 id="h1">Table name</h1>

  <script>

    var header = window.location.search.substring(11)
    header = header.toLocaleUpperCase()
```

```
    document.getElementById("h1").innerHTML = header

  </script>

  <div id="jsGrid">
  </div>

</div>
```

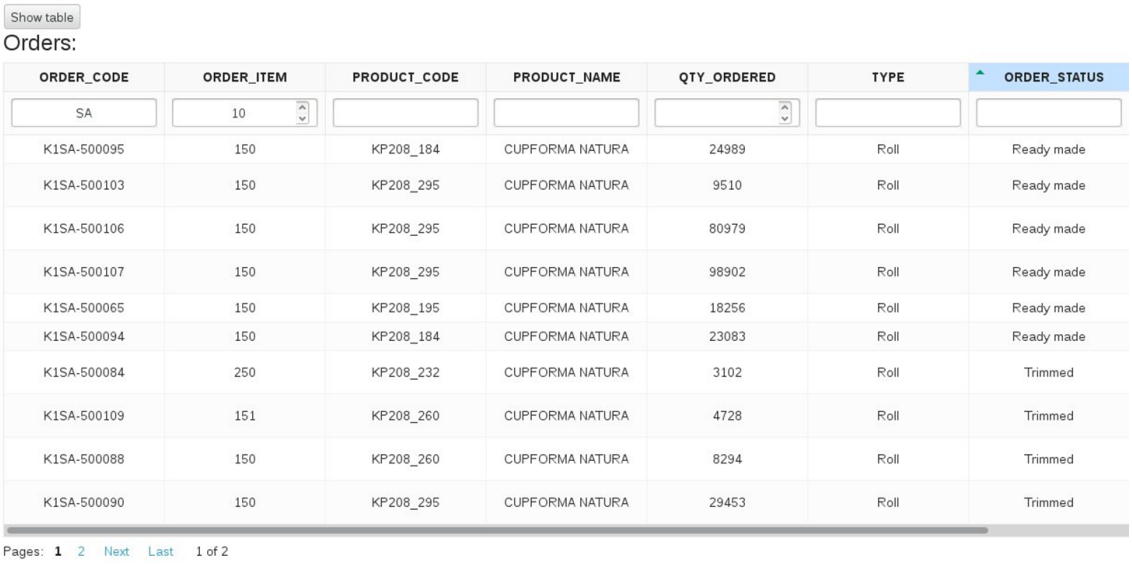
Esimerkkikoodi 38. HTML-tiedoston container-luokka.




Kuva 6. Index-sivut, kun Blocktype-niminen taulukko on haettu.



Kuva 7. Orders-sivut .




Kuva 8. Orders-sivut, kun käytössä on kaksi suodatinta ja järjestäminen tilaustilan mukaan.

# 5 Yhteenveto

Aloittaessani insinöörityön tekemistä en tiennyt Node.js:stä oikeastaan yhtään mitään, joten aloitin insinöörityön opettelemalla sen käyttöä. Huomasin pian, että onnekseni internetissa oli vaikka kuinka paljon kaikenlaista opetusmateriaalia Node.js:n käyttöön. Huomasin, että parhaiten opin katsomalla opetusvideoita YouTubesta ja tekemällä niissä olevia tehtäviä. Samalla opin lisää JavaScriptistä ja verkkosovelluksista ylipäätään.

Koska npm:n käyttö on niin olennainen osa Node.js:ää, aloin etsiä taulukkomoduuleja samalla, kun opettelin Node.js:n käyttöä, ja jatkoin etsimistä, kun tein esimerkkitoteutuksen demoversioita. Taulukkomoduuleja löytyikin hyvin paljon ja oli selvää, että minun olisi karsittava suurin osa moduuleista, jotta aikaa riittäisi tutkia moduuleja tarkemmin. Karsimisessa auttoivat muiden tekemät listaukset ja avoimen lähdekoodin käyttäjien yhteisöllisyys keskustelupalstojen muodossa. Jos taulukkomoduulilla oli GitHub-sivut, löytyivät niiden sivujen kommenteista hyvin nopeasti käyttäjien mielipiteet kyseisestä moduulista. Kun olin saanut rajattua mahdolliset taulukkomoduulit paremmin käsiteltävään määrään, jatkoin taulukkomoduulien karsimista, kunnes minulla oli enää kaksi vaihtoehtoa jäljellä. Näistä toinen oli kaupallinen GNU-lisenssiä käyttävä dhtmlxGrid ja toinen avoimen lähdekoodin MIT-lisenssiä käyttävä jsGrid. Valitsin lopulta jsGrid-moduulin, koska siinä oli paremmat lisenssiehdot ja valmis lokalisaatiotuki. Teknistä tukea jsGridille oli myös tarjolla ilmaiseksi sen GitHub-sivuilla, toisin kun dhtmlxGridille.

Lisenssien sopivuus oli erittäin tärkeää, koska ilman oikeanlaista lisenssiä ei vaaditun esimerkkisovelluksen tekeminen olisi mahdollista. Tutkiessani taulukkomoduuleja huomasin, että melkein kaikki käyttivät joko MIT-lisenssiä tai GNU-lisenssiä. Oli siis helppo valita, mitä lisenssejä tulisi tutkia tarkemmin. Tarkempi tutkimus paljasti, että molemmat näistä lisensseistä mahdollistaisivat esimerkkisovelluksen tekemisen, mutta GNU-lisenssi vaatii, että kaikkien sitä käyttävien ohjelmien pitäisi olla avointa lähdekoodia. Tämä voisi kuitenkin haitata esimerkkisovelluksen mahdollista käyttöä osana kaupallista ohjelmaa.

Valittuani sopivan taulukkomoduulin aloin tehdä esimerkkisovellusta. Vaikka esimerkkisovelluksen tekemiseen menikin enemmän aikaa, kuin oli suunniteltu, selvisi sen perusteella, että Node.js ja Oracle Database soveltuivat hyvin esimerkkisovelluksen tekemiseen. Esimerkkitoteutuksena pystyttiin tekemään ohjelma, joka haki ja näytti tietokannan tiedot selaimiin soveltuvassa muodossa käyttämällä hyväksi Node.js:n moduulikirjastoa.

Node.js:n moduulikirjastoa hetken käytettyäni huomasin hakevani sieltä moduulia ratkaisuksi, kun eteeni tuli hankala ongelma, jota en meinannut saada ratkaistua. Monesti kokonainen moduuli oli kuitenkin liian iso työkalu ongelmaani ja päädyin vain käyttämään vain tarvitsemiani osia. Tämä ei olisi ollut mahdollista ilman avoimen lähdekoodin tuomaa vapautta.

Oraclen tietokanta ja etenkin sen käyttöön tehty ajuri osoittautuivat hyväksi ratkaisuksi. Oraclen tekemän ajurin avulla tietokantahakujen tekeminen Node.js:stä osoittautui helpoksi, kunhan ajurin toimintaan tutustui.

Kuten tutkimukset antoivat olettaa, modulaarinen ohjelmointityyli sopi esimerkkitoteutuksen tekemiseen. Modulaarisuuden ansioista esimerkkitoteutuksen demoversioissa pystyi kokeilemaan eri taulukkomoduuleja ilman suuria muutoksia ohjelmaan. Myös esimerkkitoteutuksen taulukkomoduulin pystyisi vaihtamaan tarvittaessa. Modulaarisuuden ansioista ohjelman koodi pysyi luettavana eikä siihen päässyt muodostumaan sisäkkäisten takaisinkutsufunktioiden ketjua.

Epäsynkronisen siirrännän etuja ei pystytty todentamaan, koska esimerkkitoteutus salli vain yhden samanaikaisen käyttäjän. Epäsynkronisen siirrännän toimivuus palvelinohjelmissa on kuitenkin ollut jo pitkään tiedossa. Esimerkkitoteutukseen on mahdollista lisätä tuki monelle samanaikaiselle käyttäjälle, jos sitä myöhemmin tarvitaan. En kokenut sitä tarpeelliseksi toteutetusta tehdessäni. Epäsynkroninen siirräntä aiheutti aluksi ongelmia, kun ohjelma ei suorittunutkaan ylhäältä alaspäin. Ymmärrettyäni, miten takaisinkutsufunktiot toimivat, ei epäsynkronisuudesta koitunut suurempaa harmia.

Valittu taulukkomoduuli jsGrid osoittautui toimivaksi, joskin sen kaikkia ominaisuuksia ei saatu otettua käyttöön annetussa ajassa. Suodatustoiminnon saaminen toimimaan dynaamisesti eri taulukkojen kanssa osoittautui odotettua vaikeammaksi. Uskoisin sen olevan mahdollista, jos sen ratkaisemiseen käyttäisi tarpeeksi aikaa. Sain suodatuksen kuitenkin toimimaan staattisessa taulukossa sen jälkeen, kun ymmärsin, miten suodatuksen filter-palautus toimii.

JSON- esitysmuoto oli maineensa veroinen, ja sen käyttäminen oli helppoa ja sulavaa. Taulukoissa olevat ylimääräiset välilyönnit, rivien vaihdot ja lainausmerkit aiheuttivat JSON-jäsentimelleni (parser) ongelmia, mutta niiden paljastuttua syyllisiksi rikkinäiseen JSON-tietoon ne oli helppo poistaa ennen tiedon syöttämistä JSON-tiedostoon.

Olisin halunnut vielä lisätä esimerkkitoteutukseen pääsivun, josta olisi ollut mahdollista siirtyä eri taulukoihin. Muutenkin sovelluksen ulkoasua voisi parantaa huomattavasti. Aikaa esimerkkitoteutuksen tekemiseen oli kuitenkin rajallinen määrä, joten päätin keskittyä pelkästään toiminnallisuuksien tekemiseen. Mielestäni esimerkkitoteutuksessa voisi vielä kehittää tietokantahakua paremmaksi. Tällä hetkellä haut on melko lailla kovakoodattu molemmille hauille. Haluaisin, että käyttäjä voisi itse määritellä, mitä haetaan ja mistä taulusta. Kun tämä ominaisuus olisi käytössä, voisi käyttäjä käyttää koko tietokantaa hyväkseen. Tällaisessa haussa on kuitenkin olemassa tietoturvariski, koska syöttämällä oikeanlaisia SQL-käskyjä voisi käyttäjä periaatteessa tehdä tietokannassa mitä haluaa. Tämän estämiseen on kuitenkin keinoja, joten tietoturvariski ei estäisi ominaisuuden lisäämistä.

Mielestäni insinöörityö oli onnistunut, ja lopputuloksena saatiin esimerkkitoteutus, joka oli tehty käyttäen työn tutkimusosasta saatuja tietoja. Työn valmistuminen ajallaan ei onnistunut, ja kun sille varattu aika loppui ja pystyin tekemään insinöörityötä vasta päivätyön jälkeen, hidastui tekeminen entisestään. Mielestäni yksi syy ajan loppumiseen oli se, että käytin liikaa aikaa dynaamisen taulukon suodattimen tekemiseen, kuitenkaan siinä onnistumatta. Opin insinöörityötä tehdessäni uuden ohjelmointikielen, JavaScriptin ja opin myös HTML:n perusteita. Pääsin myös kokeilemaan niitä käytännössä. Avoimen lähdekoodin ohjelman tekeminen moduuleilla Node.js-ympäristössä osoittautui erittäin mielenkiintoiseksi kokemukseksi, ja uskon sen olevan tärkeä taito ammattini tulevaisuutta ajatellen.

## Lähteet


1 What is open source. 2017. Verkkodokumentti. Red Hat, Inc. <https://opensource.com/resources/what-open-source> Luettu 3.6.2017

2 Nasa. 2017. Verkkodokumentti. GitHub, Inc. <https://github.com/nasa>. Luettu 4.6.2017

3 Coverity Scan: 2013 Open Source Report. 2014. Coverity, Inc.

4 Raymon, Eric Steven. 1999. The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. O'Reilly Media.

5 High performance Node.js applications with Oracle Database. 2017. Verkkodokumentti. Oracle. <http://www.oracle.com/technetwork/database/database-technologies/scripting-languages/node_js/index.html#close>. Luettu 17.6.2017

6 Google open source projects. 2017. Verkkodokumentti. Google. <https://opensource.google.com>. Luettu 17.6.2017

7 Node.js. 2017. Verkkodokumentti. Node.js Foundation. <https://nodejs.org>. Luettu 18.6.2017

8 What is V8. 2017. Verkkodokumentti. GitHub, Inc. <https://github.com/v8/v8/wiki>. Luettu 1.7.2017

9 Teixeira, Pedro. 2012. Professional Node.js: Building Javascript Based Scalable Software. John Wiley & Sons.

10 What is npm. 2017. Verkkodokumentti. npm, Inc. <https://docs.npmjs.com/getting-started/what-is-npm>. Luettu 1.7.2017

11 McAffer, Jeff ym. 2010. Eclipse Rich Client Platform. Addison-Wesley Professional.

12 Overview of Blocking vs Non-Blocking. 2017. Verkkodokumentti. Node.js Foundation. <https://nodejs.org/en/docs/guides/blocking-vs-non-blocking/>. Luettu 9.7.2017

13 Basics of libuv. 2017. Verkkodokumentti. libuv contributors. <http://docs.libuv.org/en/v1.x/guide/basics.html>. Luettu 9.7.2017

14 Design overview. 2017. Verkkodokumentti. libuv contributors. <http://docs.libuv.org/en/v1.x/design.html>. Luettu 15.10.2017

15 A Relational Database Overview. 2017. Verkkodokumentti. Oracle. <https://docs.oracle.com/javase/tutorial/jdbc/overview/database.html>. Luettu 22.10.2017.

16 Introduction to the Oracle Database. 2017. Verkkodokumentti. Oracle. <https://docs.oracle.com/cd/B19306_01/server.102/b14220/intro.htm>. Luettu 22.10.2017.

17 Lampola, Kim. 2015. R&D Manager. ABB Oy. Haastattelu 7.12.2015.

18 Introducing JSON. 2017. Verkkodokumentti. JSON. <http://www.json.org/>. Luettu 29.10.2017.

19 JSON: The Fat-Free Alternative to XML. 2017. Verkkodokumentti. JSON. <http://www.json.org/xml.html>. Luettu 29.10.2017.

20 ECMAScript Language Specification, ECMA-262. 8th edition. 2017. Ecma International.

21 Denial of Service and Unsafe Object Creation Vulnerability in JSON. 2013. Verkkodokumentti. Ruby. <https://www.ruby-lang.org/en/news/2013/02/22/json-dos-cve-2013-0269/>. Luettu 13.8.2017.

22 Microsoft .NET Framework JSON Content Processing Denial of Service Vulnerability. 2013. Verkkodokumentti. Cisco. <https://tools.cisco.com/security/center/viewAlert.x?alertId=31048>. Luettu 13.8.2017.

23 ISO 9241-11:1998. Ergonomic requirements for office work with visual display terminals (VDTs) -- Part 11: Guidance on usability. 2008. International Organization for Standardization.

24 The MIT License. 2017. Verkkodokumentti. Open Source Initiative. <https://opensource.org/licenses/MIT>. Luettu 11.8.2016.

25 Top Open Source Licenses. 2017. Verkkodokumentti. Black Duck. <https://www.blackducksoftware.com/top-open-source-licenses>. Luettu 10.10.2017.

26 The GNU General Public License. 2017. Verkkodokumentti. Free Software Foundation, Inc. <https://www.gnu.org/licenses/>. Luettu 11.8.2016.

**Server.js**

```
var express = require('express');
var app = express();
var db = require('./dbtoJSON');
var dbOrders = require('./getOrders');
var fs = require('fs');

var tableName = '';
var dataJSON = '';

//Makes a empty grid config file
var gridConfFile = './public/gridConf.js';
var text = 'data = [{"name":"numberOfColumns","sorter":0}]' ;
fs.writeFile(gridConfFile , text, function (err) {
    if (err) return console.log(err);
});

//Runs when client calls the '/' page
//Gets the data from DB
//Puts the data to './public/data.json'
app.get('/', function (req, res) {
    console.log("Client called")

    //Gets the table name from query box in clients browser
    tableName = req.query.tablename;

    //Gets the data from DB
    //Puts the data to './public/data.json'
    if (tableName != undefined) {
        tableName = tableName.toLocaleUpperCase();
        console.log("Get table: " + tableName);

        db.getJSON(tableName, function (err, data) {
            var fileName = './public/data.json';
            dataJSON = data;

            //Writes the data to './public/data.json'
            fs.writeFile(fileName, dataJSON, function (err) {
                if (err) return console.log(err);
                console.log('Data is in ' + fileName);
            });
        });
    }

    //Sends the index.html to client
    res.sendFile('public/index.html', {root: __dirname });
}); //End of app.get

//Show Orders page
app.get('/orders', function (req, res) {
```

```
    console.log("Client called")

    //Gets the data from DB
    //Puts the data to './public/data.json'

    console.log("Show orders");

    dbOrders.getJSON(function (err, data) {
        var fileName = './public/data.json';
        dataJSON = data;

        //Writes the data to './public/data.json'
        fs.writeFile(fileName, dataJSON, function (err) {
            if (err) return console.log(err);
            console.log('Data is in ' + fileName);
        });
    });

    //Sends the orders.html to client
    res.sendFile('public/orders.html', {root: __dirname });
}); //End of app.get

//Folder name of the public app
app.use(express.static(__dirname + '/public/'));

//Sets the server to listen to port 3000
var port = 3000;    //http://localhost:3000
app.listen(port, function () {
    console.log('Server on localhost:' + port);
});
```

**DBtoJSON.js**

```
var oracledb = require('oracledb');
var dbConfig = require('./dbconfig.js');
var fs = require('fs');

module.exports.getJSON = getJSON;

//Gets the grid layout and data from DB
//Puts the grid layout to './public/gridConf.js'
//Returns data as JSON format string
function getJSON(tName, callback) {

  var JSONData = '';
  var tableName = tName;
  var rows = '';
  var data = '';

  var rowsToGet = 1000;
  var getAll = false;

  //Gets the grid layout and data from DB
  //Puts the grid layout to './public/gridConf.js'
  //Returns data as JSON format string
  getTable(tableName, function (err, columnName,
           numberOfColumns, numberOfRows, gridConf, connection){

    table = columnName;
    tableLength = numberOfColumns;
    rows = numberOfRows;

    doRelease(connection);

    var fileName = './public/gridConf.js';
    data = 'data = ';
    data += JSON.stringify(gridConf);
    data += ';';

    //Writes the grid layout to './public/gridConf.js'
    fs.writeFile(fileName, data, function (err) {
      if (err) return console.log(err);
      console.log('Grid config is in ' + fileName);
    });

    //Gets the data if the table has more than 0 rows
    //i.e. it exists
    if(rows > 0) {
      if (getAll)
        rowsToGet = rows;
      else if (rowsToGet > rows)
        rowsToGet = rows;
```

```
      //Sets the maximum rows that the query can return
      //to rowsToGet
      if (rowsToGet > 0)
        oracledb.maxRows = rowsToGet;

      console.log("Rows to get: " + rowsToGet);
      console.log("Rows in DB: " + rows);

      //Gets the data from DB
      //Returns the data in JSON format string
      getData(rowsToGet, tableName, table, tableLength,
              function (err, jsonString, connection) {
        //Removes all row changes and
        //puts the data to JSONData string
        JSONData = jsonString.replace(new RegExp('\r\n|\n|\r',
                   'gm'), null);
        doRelease(connection);
        callback(null, JSONData);
      }); //End of getData
    }
  }); //End of getTable
}

//Gets the table layout
//Returns column names and grid configuration array
//and the number of columns and rows
function getTable(tableName, callback) {

  var numberOfColumns = 0;
  var numberOfRows = 0;
  var columnNames = [];
  var gridConf = [];

  //Gets column names and grid configuration array
  //and the number of columns and rows
  oracledb.getConnection(
    {
    user          : dbConfig.user,
    password      : dbConfig.password,
    connectString : dbConfig.connectString
    },
    function(err, connection) {
    if (err) {
      console.log(err);
      throw err;
    }
```

```
//Gets the number of columns in table to numberOfColumns
connection.execute(
  'SELECT COUNT(column_id) ' +
  'FROM user_tab_columns ' +
  'WHERE table_name = \'' + tableName + '\'',
  {},
  function(err, results) {
    if (err) {
    console.log(err);
    }

    if (results != undefined) {
    numberOfColumns = results.rows[0][0];
    }
  }
);


//Gets the number of rows in table to numberOfRows
connection.execute(
  'SELECT COUNT(*) ' +
  'FROM ' + tableName,
  {},
  function (err, results) {
    if (err) {
    console.log(err);
    }

    if (results != undefined) {
    numberOfRows = results.rows[0][0];
    }
  }
);

//Gets the column names and types in table to gridConf array
connection.execute(
  'SELECT column_name,data_type ' +
  'FROM user_tab_columns ' +
  'WHERE table_name = \'' + tableName + '\'',
  {},
  function (err, results) {
    if (err) {
    console.log(err);
    }

    //Puts the number of columns to
    //the first value of the gridConf
    gridConf[0] = {
    name: 'numberOfColumns',
    sorter: numberOfColumns
    };
```

```
        //Puts the name and type to gridConf and columnNames
        if(results != undefined) {
        var type = '';
        for (var i = 0; i < numberOfColumns; i++) {
          if (results.rows[i][1] == 'NUMBER') {
            type = 'number';
          }
          else {
            type = 'string';
          }

          gridConf[i + 1] = {
            name: results.rows[i][0],
            sorter: type
          };

          columnNames[i] = {
            name: results.rows[i][0],
            sorter: type
          };
        }
        }
        else{
            console.log("ERROR: Results undefined!")
        }

        callback(null, columnNames, numberOfColumns,
                 numberOfRows, gridConf, connection);
      }
    );  //End of connection.execute
    }
  )   //End of oracledb.getConnection
}

//Gets data from DB
//Returns a JSON format string of the data
function getData(nroOfRows, tableName, table,
                 tableLength, callback) {

  var variable= [];
  variable= "[";
  var values = "";
  var name = "";
  var value = "";
  var str = "";
```

```
//Builds and makes a query to DB and
//returns the data in JSON format string
oracledb.getConnection(
  {
  user          : dbConfig.user,
  password      : dbConfig.password,
  connectString : dbConfig.connectString
  },
  function(err, connection) {
  if (err) {
    console.log(err);
    throw err;
  }

  //Build query using column names
  var query = 'SELECT ';

  //Loops through the column names
  for(var i = 0; i < tableLength; i++){
    query += table[i]["name"];

    if(tableLength!= i +1){
      query += ', ';
    }
  }
  query += ' FROM ' + tableName;
  //query = SELECT columnName1, columnName2 FROM tableName

  //Makes the query to DB
  connection.execute(
    query,
    {},
    function(err, results) {
      if (err) {
      console.log(err);
      throw err;
      }
```

```
        //Loops through the query result
        //and puts it in JSON format string
        // [{"name1":"value1","name2":"value2"},{...}]
        for(var j = 0; j < nroOfRows; j++) {
        values = "{";
        for (var i = 0; i < tableLength; i++) {
          name = table[i]["name"];
          value = results.rows[j][i];

          //Remove extra spaces and " from result string
          if (value != null){
            str = results.rows[j][i].toString();
            str = str.replace(/"/g, null);
            str = str.replace(/\s+/g," ");
            value = str;
          }

          values += "\"" + name + "\":\"" + value + "\"";
          //"name":"value"

          //Adds a comma if not last value else ends the object
          if ((i+1) < tableLength)
            values +=  ",";
          else
            values +=  "}";
        }

        variable += values;
        //[{"name1":"value1","name2":"value2"},{...}

        //Adds a comma if not last object
        if(j+1 < nroOfRows)
          variable += ",";
        }
        //Ends the JSON string
        variable += "]";
        //[{"name1":"value1","name2":"value2"}],{...}]

        callback(null, variable, connection);
      }
    );  //End of connection.execute
    }
  );  //End of oracledb.getConnection
}
```

```
//Releases the connection to DB
function doRelease(connection)
{
  connection.release(
    function(err) {
    if (err) {
      console.error(err.message);
    }
    console.log('Connection released.');
    });
}
```

**grid_jsGrid.js**

```
//Makes the grid component for the browser using jsGrid
function makeGrid() {
  //Requires the grid config file to work
  require(['../gridConf'], function (config) {});

  var field = [];
  var gridConfig = data;
  var columns = {name:[], width:[],sorter:[]};
  var columnWidth = 100;

  //Sets values to column object from gridConfig
  for (var i = 0; i <= gridConfig[0].sorter; i++) {
      columns.name[i] = gridConfig[i].name;
      columns.sorter[i] = gridConfig[i].sorter;
      columns.width[i] = columnWidth;
  }

  //Sets values to fields array from gridConf
  for (var i = 1; i <= numberOfColumns; i++) {
      field.push({
          name: columns.name[i],
          width: columns.width[i],
          sorter: columns.sorter[i],
          type: function () {
            if (column.sorter[i]== "string") return "text";
                          else return "number";
          },
          align: "center"
      });
  }

  //Renders the grid to browser
  $("#jsGrid").jsGrid({
    autoload: true,

    controller: {
      loadData: function (filter) {
        return $.ajax({
          type: "GET",
          url: "../data.json",
          data: filter,
          dataType: "json"
        });
      }
    },
```

```
    //Used for grid setup
    width: "100%",
    height: "auto",
    pageSize: 10,
    pageButtonCount: 5,

    //Used for turning features on and off
    filtering: false,
    inserting: false,
    editing: false,
    sorting: true,
    paging: true,
    deleting:false,

    //Fields contains the layout of the grid i.e.
    //the column names, width, sorters, types and alignments
    fields: field

  }); //End of $("#jsGrid").jsGrid
}  //End of makeGrid
```

**Index.html**

```
<!DOCTYPE html>
<html>
  <head>
    <title>DB to Browser</title>
    <link rel="stylesheet" type="text/css"
          href="stylesheets/style.css" />
    <link type="text/css" rel="stylesheet"
          href="modules/jsgrid-1.4.1/jsgrid.min.css" />
    <link type="text/css" rel="stylesheet"
          href="modules/jsgrid-1.4.1/jsgrid-theme.min.css" />
    <meta charset="UTF-8">

    <script src="modules/jquery/dist/jquery.min.js"></script>
    <script src="modules/jsgrid-1.4.1/jsgrid.min.js"></script>

    <script data-main="gridConf"
            src="modules/requirejs/require.js"></script>
    <script type="text/javascript"
            src="javascripts/grid_Jsgrid.js" ></script>
  </head>
  <body>
    <form>
      Table name:<br>
      <input type="text" name="tablename"
             placeholder="Table name"><br>
      <input type="submit" value="Get data">
      <input type="reset">
    </form>

    <div class="container">
      <button onclick="makeGrid()">Show table</button>

      <h1 id="h1">Table name</h1>

      <script>
        var header = window.location.search.substring(11)
        header = header.toLocaleUpperCase()
        document.getElementById("h1").innerHTML = header
      </script>

      <div id="jsGrid">  </div>
    </div>

  </body>
</html>
```

## Taulukkomoduulien ominaisuudet

| Name | Description | License | Browser support | Data format | Ease of implementation | Development status | Latest version |
|---|---|---|---|---|---|---|---|
| Backgrid.js | Backgrid.js is a set of components for building semantic and easily stylable data grid widgets. | MIT | Desktop & Mobile: IE 8+, Chrome 4+, Safari 4+, Firefox 4+, Opera 9+ | JSON | Easy | Feature complete | 0.3.5 (21.1.2014) |
| jsGrid | jsGrid is a lightweight client-side data grid control based on jQuery. It supports basic grid operations like inserting, filtering, editing, deleting, paging, and sorting. | MIT | Desktop: Chrome, Safari, Firefox, Opera 15+,IE 8+<br>Mobile: Safari, Chrome, IE10 | JSON | Easy | Done, new features possible | v1.4.1 (21.2.2016) |
| Angular UI Grid | A data grid for AngularJS. Native AngularJS implementation, no jQuery. Performs well with large data sets. | MIT | Desktop: IE9+(marginal), Chrome, Firefox, Opera, Safari 5+<br>Mobile: Safari 5+, Android 4+ | CVV or JSON | Easy | Done, new features coming | 3.2.1 (24.6.2016) |
| React Data Grid | Excel-like grid component built with React | MIT | Tested Desktop: Chrome, Firefox, IE 11 | JSON | Easy | Done, new features possible | v1.0.9 (7.7.2016) |
| SlickGrid | SlickGrid is a JavaScript grid/spreadsheet component. | MIT | Desktop: All except IE6 and Opera | JSON | Difficult | Development for original has stopped, alternative adding new features | v2.2.6 (3.2.2016) |
| Open JS Grid v2 | The No-Work-Involved Datagrid | MIT | Tested Desktop: Chrome, Firefox, IE 11 | mySQL | Easy | Done, no new features coming | v2.1.5 (21.11.2013) |
| Grid | A highly scaleable grid component written in javascript | MIT | Tested Desktop: Chrome, Firefox, IE 11 | Own datamodel, data format doesn't matter | Difficult | Fully functional, but still in beta | v1.4.0 (7.10.2015) |
| dhtmlxGrid | Ajax Powered JavaScript DataGrid Component Editable JavaScript Table | GNU General Public License, version 2 | IE, Chrome, Firefox, Opera, Safari | XML, JSON, CSV, JavaScript array, HTML table | Easy | Commercial product | v5.0 |

| Name | Styling | Formatting | Data filtering | Sorting | Pageable | Documentation | Get data with multiple searches |
|---|---|---|---|---|---|---|---|
| Backgrid.js | With CSS | Cells, Columns, Rows | Yes | Yes | Yes | Good | Yes with JSON |
| jsGrid | With CSS | Rows | Yes | Yes | Yes | Good | Yes with JSON |
| Angular UI Grid | With CSS | Cells | Yes | Yes | In alpha stage | Good | Yes with JSON |
| React Data Grid | With CSS | Cells | Yes | Yes | No | Good | Yes with JSON |
| SlickGrid | With CSS | Cells | Yes | Yes | No | Good | Yes with JSON |
| Open JS Grid v2 | With CSS | Cells, Rows | Yes | Yes | Yes | OK | Possible |
| Grid | With Decorators | Cells, Columns, Rows | Not yet | Not yet | Not yet | Weak | Not yet |
| dhtmlxGrid | With CSS or skins | Cells, Columns, Rows | Yes | Yes | Yes | Good | YES |

| Name | GitHub | Localization | Data Pivoting | Read/write | Comments | Link |
|---|---|---|---|---|---|---|
| Backgrid.js | Active | No | No | Read/write | Highly modular and customizable. Lightweight. Drag&drop row reordering. | http://backgridjs.com/ |
| jsGrid | Active | Yes | No | Read/write | Flexible and allows to customize its appearance and components. Drag&drop row reordering. | http://js-grid.com |
| Angular UI Grid | Active | Yes | No | Read/write | Plugins for new features. Tree view in beta. Data export in CSV or pdf. | http://ui-grid.info/ |
| React Data Grid | Active | No | No | Read/write | Works with hundreds of thousands rows | http://adazzle.github.io/react-data-grid/index.html# |
| SlickGrid | Not active | No | No | Read/write | | https://github.com/6pac/SlickGrid/wiki https://github.com/mleibman/SlickGrid/wiki |
| Open JS Grid v2 | Not active | No | No | Read/write | Can do complex mysql things like where, join, concat, group, having and more | http://square-bracket.com/openjs https://github.com/optikalefx/OpenJS-Grid |
| Grid | Not active | No | No | Read/write | | https://github.com/relateiq/grid |
| dhtmlxGrid | No | No | No | Read/write | Free version uses GNU licence, so the whole program is open source with it. Grouping of cells. Highly customizable. Math formulas for cells. Export to Excel and PDF. Treeview. | http://dhtmlx.com/docs/products/dhtmlxGrid/ |