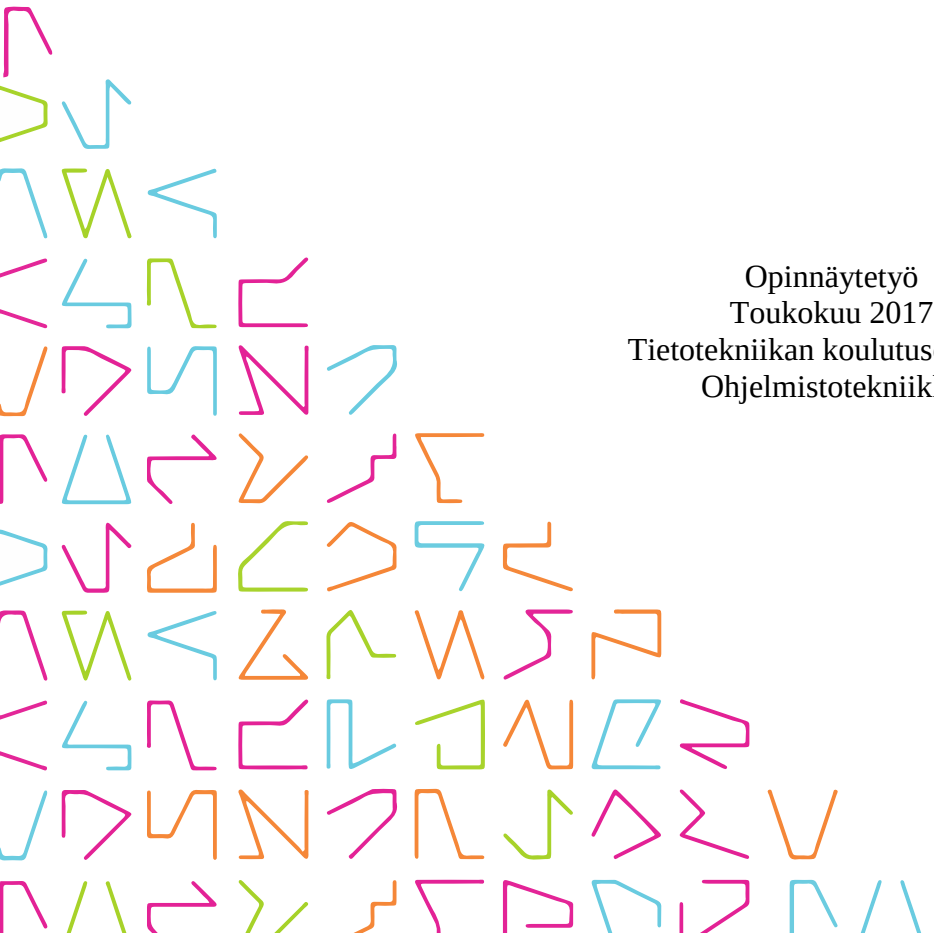
TAMPEREEN
AMMATTIKORKEAKOULU

OIVA-TIETOKANNAN KEHITYS JA DATAN VISUALISOINTI

Risto Aaltonen

Matias Rimmi



Opinnäytetyö
Toukokuu 2017
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

Aaltonen Risto & Rimmi Matias:
Oiva-tietokannan kehitys ja datan visualisointi

Opinnäytetyö 25 sivua, joista liitteitä 0 sivua
Toukokuu 2017

Opinnäytetyössä on ideoitu toimintoja ja etsitty mahdollisia kehitysehdotuksia jo olemassa olevaan, Tampereen Ammattikorkeakoulun käyttämään Oiva-palveluun, joka sisältää opinnäyte- ja opiskelijaharjoitteluraportteja. Opinnäytetyössä keskitytään Oiva-palvelun opinnäytetöiden listaamiseen ja tiedonhallintaan liittyvään toiminnallisuuteen. Osasta kehitysideoita on myös tuotettu prototyyppejä, joita tarkastellaan erikseen. Kehitysideoissa käydään läpi lähinnä palveluun sisältyvään tiedon esittämiseen ja muokkaukseen osallistuvia toimintoja. Jokaisen toiminnon kohdalla käsitellään erikseen myös löydettyjä ja ehdotettuja parannuksia eri toiminnollisuuksiin. Osassa tilanteista, projektin tuotoksia käsitellään tuotettujen prototyyppien kautta.

Opinnäytteessä todistetaan lisäparannuksien tarpeellisuus ja datan visualisoinnin hyödyt. Eri ongelmakohdat tuodaan esille ongelmatapausten kautta. Työn aikana on myös pyritty tuomaan esille eri näkökulmia havaittujen ongelmien ratkaisuun.

Työssä käytettyjä tekniikoita ovat Node.js ja D3.js–viitekehys, PostgreSQL-tietokanta ja ulkopuolinen REST-rajapinta. D3.js–viitekehyksen ohella, käsitellään datan visualisointiin liittyviä eri visualisointityyppejä ja -tekniikoita.

Asiasanat: html, css, javascript, d3, node.js, postgresql, react, datavisualisaatio

ABSTRACT

Tampere University of Applied Sciences
Degree Programme in ICT Engineering
Software Technology

Aaltonen Risto & Rimmi Matias:
Improvement of Oiva-database and data visualization

Bachelor's thesis 25 pages, appendices 0 pages
May 2017

The purpose of this thesis is to improve an existing database service called Oiva, that is used by Tampere University of Applied Sciences to store information on theses and student training reports. This thesis focuses on Oiva's listing of theses and data management functionality. Some of the improvement ideas are also made into prototypes, which will be examined as well. Improvement ideas are mainly based on the presentation and editing functionality Oiva and are presented for each functionality. The improvements are based on the previously mentioned prototypes.

The thesis also explains the need for these improvements and the benefits of data visualization. The problems are presented with actual problem cases and different points of view were also considered and presented.

The development technologies used in this thesis are Node.js and D3.js frameworks, PostgreSQL-database and the external REST-api. Some different data visualization types and techniques are also presented along with the D3.js framework.

Key words: html, css, javascript, d3, node.js, postgresql, react, data visualization

SISÄLLYS

LYHENTEET JA TERMIT

Big data	”iso data”, suurten tietomäärien keräys ja säilytys
asynkroninen	ajasta riippumaton, kommunikoinnin osapuolet eivät riipu toisistaan
REST-rajapinta	”Representational State Transfer”, eräs tapa järjestelmien väliseen kommunikaatioon internetin välillä
cross-origin	web-mekanismi resurssien jakamisesta eri sijaintiin, kuin mihin ensimmäinen resurssi lähetettiin

1 JOHDANTO

Opinnäytetyön pohjana pidetään Tampereen Ammattikorkeakoulun KINO-hankeen A-osan aikana suoritettua tutkimus – ja kehitystyötä. Projekti jakaantui useaan laajaan osuuteen, joissa opiskelijat ja opettajat saivat mahdollisuuden työskennellä. Eräässä tutkimusaihiossa lähtökohtana oli ideoida toimintoja ja etsiä mahdollisia parannusehdotuksia jo olemassa olevaan, Tampereen Ammattikorkeakoulun käyttämään Oiva-palveluun.

Opinnäytetyössämme keskitytäänkin Oiva-palvelun opinnäytetöiden listaamiseen ja tiedonhallintaan liittyvän osaan. Osasta kehitysideoita on myös tuotettu prototyyppejä, joita tarkastellaan erikseen opinnäytetyön aikana. Kehitysideoissa käydäänkin läpi lähinnä palveluun sisältyvään tiedon esittämiseen ja muokkaukseen osallistuvia toimintoja. Jokaisen toiminnon kohdalla käsitellään erikseen myös löydettyjä ja ehdotettuja parannuksia eri toiminnollisuuksiin. Osassa tilanteista, projektin tuotoksia käsitellään tuotettujen prototyyppien kautta.

Opinnäytteen kautta pyritään myös todistamaan lisäparannuksien tarpeellisuus ja datan visualisoinnin hyödyt. Jotta ongelmakohdat voitaisiin havaita, tuli ne tuoda esille ongelmatapausten kautta. Jokaisen tapauksen kohdalla prototyypin tuottaminen havaittiin huomattavaksi apukeinoksi selvitysprosessissa. Työn aikana on myös pyritty tuomaan esille eri näkökulmia havaittujen ongelmien ratkaisuun.

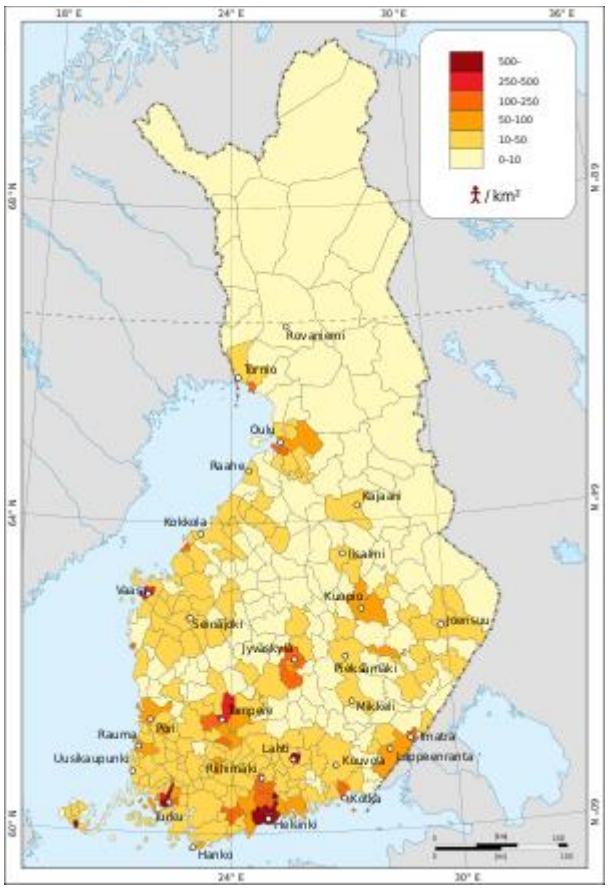
Tekniseltä kannalta opinnäytetyö käy läpi uusimpia web-teknologioita käytännössä. Työssä paneudutaan Javascript-sovelluksen kehittämiseen käyttäen Node.js ja D3.js–viitekehystä, PostgreSQL-tietokantaa ja ulkopuolista REST-rajapintaa. D3.js–viitekehyksen ohella, käsitellään datan visualisointiin liittyviä eri visualisointityyppejä ja -tekniikoita.

2 DATAN VISUALISOINTI

2.1 Datan visualisoinnin tärkeys

Tieto on tärkeätä ja sitä kerätään enemmän kuin koskaan. ”Big Data” on kuuma aihe ja suuri tietomäärä tuottaa paljon hyödyllistä informaatiota. Ongelmana on kuitenkin kaiken tiedon sisäistäminen ja analysointi. Tietokanta täynnä tietoa ei hyödytä ketään, jos siitä ei voi helposti saada kokonaiskuvaa. Pelkkä data ei sinällänsä omaa isoa tietoarvoa, vaan se piilottaa tiedon sisäänsä ja tieto itsessään löytyy isosta datamäärästä ilmenenevistä kokonaisuuksista. Tällaisissa tilanteissa voi olla suurtakin hyötyä datan visualisoinnista, joka tarkoittaa datan esittämistä graafisessa muodossa.

Esimerkiksi iso kasa väestötietoa ei itsessään ole kovin kuvaavaa tai helposti analysoitavaa dataa, mutta kun tämä tieto visualisoidaan kartaksi (kuva 1) huomataan nopeastikin mihin Suomen väestö on sijouttunut.



KUVA 1. Suomen väestötiheys (Oona Räisänen 2010)

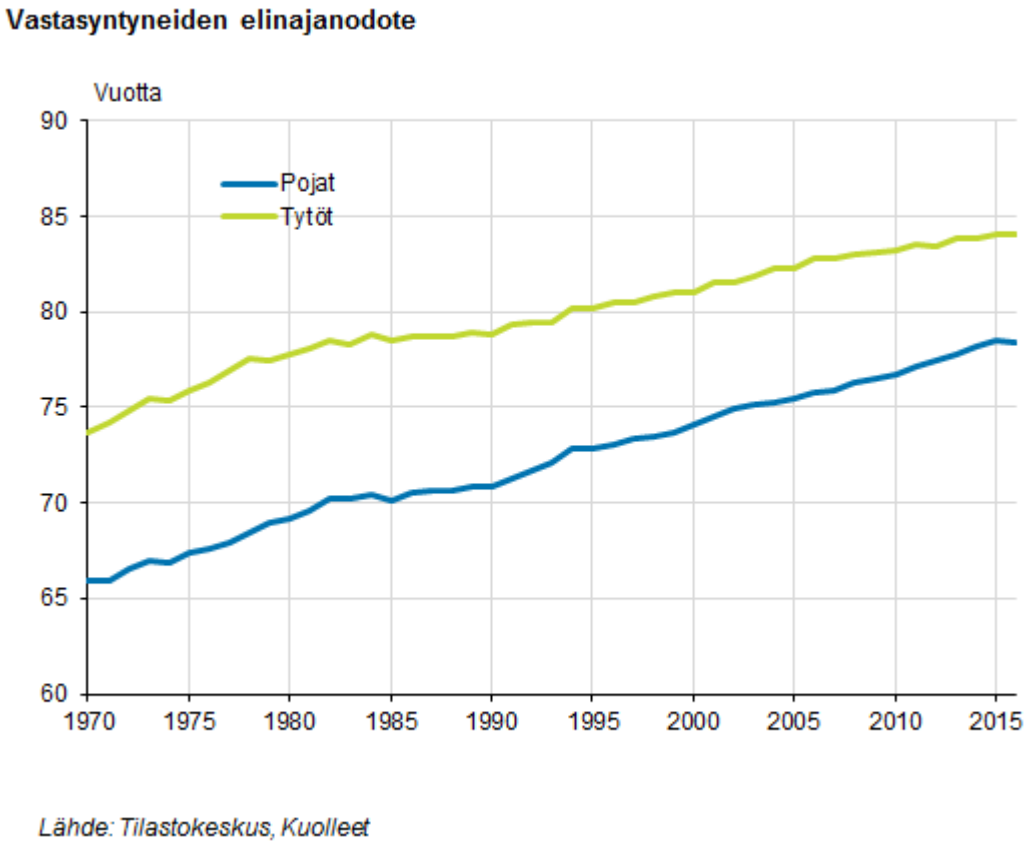
Visualisoinnin hyödyt eivät ole ainoastaan ison datamäärä analyssoinnissa vaan myös pienempi ja helposti ymmärrettävämpi data saadaan visualisoimalla hetkessä analysoitavaan muotoon. Esimerkkinä taulukko 1 sisältää vastasyntyneiden elinajanodotteen eron sukupuolin välillä.

TAULUKKO 1. Vastasyntyneiden elinajanodote 1970-2016 (Tilastokeskus 2016)

	Vuotta	
	Pojat	Tytöt
1970	65,9	73,7
1980	69,2	77,8
1990	70,9	78,9
2000	74,1	81,0
2010	76,7	83,2
2011	77,2	83,5
2012	77,5	83,4
2013	77,8	83,8
2014	78,2	83,9
2015	78,5	84,1
2016*	78,4	84,1

Taulukko on selkeä ja data on yksinkertaista. Vaaditaan kuitenkin taulukon eri solujen tutkimista ja yksittäisien lukujen vertailuja, jotta selviää mitä taulukko varsinaisesti todistaa.

Kuvaa 2 vilkaisemalla tulee kuitenkin jo selväksi, että vastasyntyneiden tyttöjen elinajanodote on korkeampi kuin poikien ja että elinajanodote on pysynyt nousussa. Myös eron määrä näkyy nopeasti ja mikään tieto ei ole visualisoinnin vaikutuksesta poistunut. Tieto on analysoitu hetkessä ja lopputulos on selkeä ja helposti esitettävä. Analyysiin ei kulunut turhaa aikaa ja voidaan siirtyä tutkimaan lopputuloksen syitä ja seurauksia.



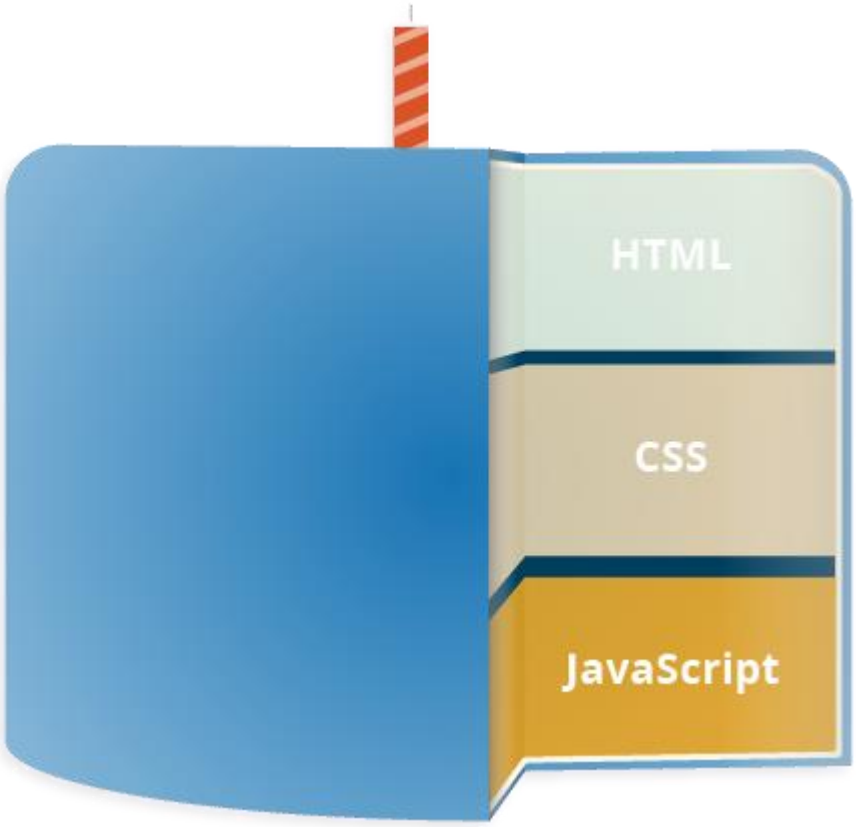
KUVA 2. Vastasyntyneiden elinajanodote (Tilastokeskus 2016)

Datan visualisoinnissa on oleellista datalle oikeanlaisen mallin käyttö, värien käyttö, sekä oikeiden asioiden esille tuominen. On myös hyvä muistaa, että kaikki data ei toimi hyvin visuaalisena ja liiallista visualisointiakin tulee välttää.

3 KÄYTETYT TEKNIIKAT

3.1 HTML

HTML (Hypertext Markup Language) on avoimesti standardoitu kuvauskieli (markup language), jolla luodaan verkkosivuja ja –sovelluksia. Se toimii yleensä pohjana ja rakenteena verkkosivuille ja se on yksi webkehityksen kulmakivistä CSS:än ja JavaScriptin ohella (kuva 3). (Mozilla 2017)



KUVA 3. Webkehityksen tärkeimmät tekniikat

3.2 CSS

CSS (Cascading Style Sheets) on toinen tärkeä webin tekniikka, joka antaa säännöt verkkosivun tyylille. Siinä määritellään esimerkiksi taustavärit ja fontit HTML sisällölle. (Mozilla 2017)

3.3 JavaScript

JavaScript on nykyajan käytetyimpiä ohjelmointikieliä. Se on yksi webin ydinkielistä HTML:än ja CSS:än ohella. Sitä käytetään lisäämään dynaamista toiminnallisuutta HTML-sivuihin. Kaikki modernit selaimet tukevat JavaScriptiä ilman erillisiä plug-inejä. On myös hyvä huomata, että Java ja JavaScript ovat täysin eri kieliä syntaksiltaan, sekä käyttötarkoitukseltaan eivätkä ne liity varsinaisesti toisiinsa. JavaScript ei ole myöskään ainoastaan webpohjaiseen käyttöön vaan sitä käytetään myös mm. pdf dokumenteissa ja työpöytä widgeteissä. (Mozilla 2017)

JavaScript tukee niin oliopohjaista, imperatiivista sekä funktionaalista ohjelmointityyliä. Se on kielenä korkeatasoista, dynaamista ja tulkattavaa. (Mozilla 2017)

Korkeatasoiset ohjelmointikielet helpottavat monia asioita matalan tason kieliin verrattuna ja on jotain matalan kielten ja puhuttujen kielten väliltä. Käytännössä korkeatasoisissa kielissä automatisoidaan esimerkiksi muistinhallintaa ja esimerkiksi osoittimia ei käytetä. (Dictionary 2011)

Dynaamisuus taas viittaa siihen, että osa sen toiminnallisuudesta toteutetaan sitä ajettaessa. Toisin kuin staattisissa kielissä, joissa nämä toiminnallisuudet toteutettaisiin koodia käännettäessä. (Mozilla 2017)

Tulkattavat kielet tulkataan koneelle ohjelman suorituksen aikana. Käännettävä kieli taas tulkattaisiin ennen ohjelman suorittamista. (Mozilla 2017)

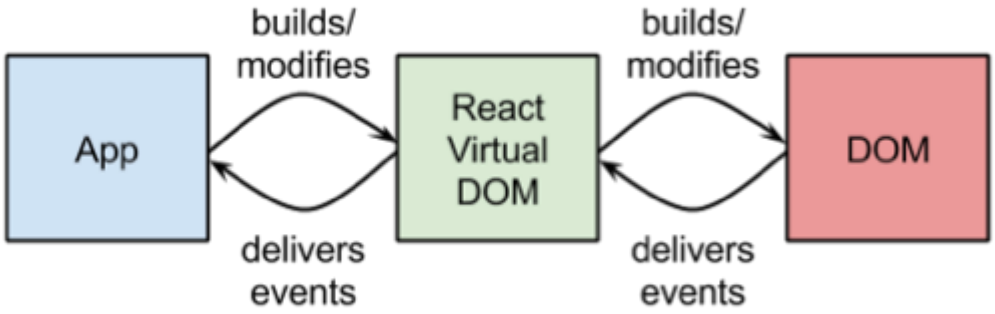
Käytännössä JavaScritpillä voidaan esimerkiksi tallettaa arvoja muuttujiin, ajaa koodia vastauksena tapahtumiin kuten nappien painamiseen ja paljon muuta. Sillä voi myös käyttää erilaisia ohjelmointirajapintoja. Näihin kuuluvat esimerkiksi selaimen omat rajapinnat kuten audio-, video-, sijainti- ja muut hyödylliset rajapinnat. Myös kolmannen osapuolen rajapintoja voi käyttää, mutta on hyvä huomata, että selaimet eivät näitä oletuksena tue. (Mozilla 2017)

3.4 ReactJS

ReactJS on Facebookin ylläpitämä avoimen lähdekoodin JavaSript-kirjasto. Sillä voi toteuttaa websovelluksien käyttöliittymiä, jotka käyttävät muuttuvaa dataa päivittämättä sivua. Sen tarkoituksen on olla nopeaa, yksinkertaista sekä skaalautuvaa. (React 2017)

Reactissa määritellään komponentit ja niistä rakennetaan käyttöliittymiä ja jokaisella komponentilla on oma lokaali tilansa, joka välittää tietoa alaspäin. Komponentin tilan muuttuessa se renderöidään uudelleen koskematta muuhun, mikä mahdollistaa datan muuttumisen päivittämättä sivua. Komponentti voi koostua HTML-elementeistä, sekä muista React-komponenteista. (React 2017)

React tarjoaa myös virtuaalisen DOM:in (Document Object Model), jonka avulla hitaamman HTML-DOM:in päivitys onnistuu helpommin ja sovelluksen toiminta nopeutuu (KUVA 4). (React 2017)



KUVA 4. Virtuaalisen DOM:in toiminta (Dzone 2017)

3.5 Node.js

Node.js on asynkroninen avoimen lähdekoodin JavaScript ajonaikainen ympäristö, jota käytetään JavaScriptin ajamiseen palvelinpuolella. Yleensä JavaScriptiä käytetään pääasiassa asiakaspuolella, mutta Node.js:än avulla ajetaan scriptejä palvelinpuolella. Sillä tuotetaan dynaamista websisältöä ennen kuin sivu lähetetään käyttäjän selaimeen. (Node 2017)

Palvelin- ja verkkotoiminnallisuuden lisäksi Node.js tarjoaa myös eri toiminnallisuutta tarjoavia moduuleja, jotka tarjoavat esim. tiedostojärjestelmäliitäntöjä, datavirtoja ja salausfunktioita. Node.js tarjoaa myös npm (node package manager) paketinhallinnan, jolla voi hakea ja jakaa rekisteristä löytyviä JavaScript-moduuleja ja se hallitsee myös sovelluksen riippuvuuksia. Sillä voidaan yhtä komentoa käyttämällä asentaa kaikki projektin riippuvuudet ”package.json”-tiedostoa käyttämällä. (Node 2017)

3.6 D3.js

D3.js (Data Driven Documents) on Javascript-pohjainen kirjasto datan visualisoimista varten. D3-kirjaston avulla tietoa voidaan muokata käyttäen tavallisimpia web-kehitysstandardeja, kuten HTML, SVG ja CSS. Kirjasto sisältääkin suuren määrän erilaisia visualisointiin liittyviä komponentteja ja moduuleja, kuten histogrammeja, paneeleja ja akseleita (D3 2017).

Kirjaston avulla tehtävät muokkaukset tapahtuvat muokkaamalla rakenteisen dokumentin dokumenttioliomallia (Document Object Model). Käyttötilanteessa data sidotaankin dokumenttioliomalliin, jonka jälkeen D3-kirjaston kautta määritellyt muutokset näkyvät kyseisellä dokumentilla (D3 2017).

Käytännössä D3–koodi perustuu valintoihin (selections), joilla voidaan päästä kiinni mihinkä tahansa dokumenttioliomallin solmuihin. Tavallisessa käyttötapauksessa kutsutaan aluksi d3-olion instanssia, josta voidaan kutsua haluttua valintaa. Solmuihin päästään käsiksi esimerkiksi valintaan asetetulla Id:llä, atribuutin arvolla, luokalla tai vaikkapa HTML-tägillä (kuva 5).

Solmuja voidaan muokata useilla eri tavoilla, muun muassa muuttamalla tyylejä ja määrittelyjä, lisäämällä ja poistamalla, lajittelemalla ja muuttamalla sisältöä. Dynaamisina ominaisuuksina voidaan valintojen sisällä kutsua funktioita ja sitoa dataa. Animoiduissa siirtymissä voidaan taas käyttää kirjaston omaa transition-funktioita (D3 2017).

KUVA 5. Dokumentin taustan muuttaminen harmaaksi, 3 sekunnin viiveellä

3.7 PostgreSQL

PostgreSQL on Berkeleyn Tietojenkäsittelytieteen laitoksen, postgres-projektin koodista syntynyt, oliorelaatiotietokannan hallintajärjestelmä. Järjestelmä perustuu avoimeen lähdekoodiin ja tukee suurta osaa SQL-standardin alaisista toiminnallisuuksista. Lisäksi ohjelmisto tarjoaa liudan toimintoja, joita muista tietokantajärjestelmistä ei projektin aloitushetkellä löytynyt (PostgreSQL 2017).

Eräs halutuista toiminnallisuuksista oli niin kutsuttu Full Text Search-haku. Muista järjestelmälle yksilöllisiä toiminnallisuuksia ovat muun muassa, tiedon transaktioiden muuttumattomuus, NoSQL-ja JSON-tietotyyppituki, tuki palvelinpuolen kielille ja materialisoidut näkymät. Avoimen lisenssin ansioista Postgres-tietokantaa voidaan käyttää maksuttomasti niin yksityisissä, että kaupallisissa projekteissa (PostgreSQL 2017).

PostgreSQL-järjestelmässä käytetään asiakas/palvelin-arkkitehtuuria. Jokainen Postgres-istunto (kuva 6) käsittää aina vähintään yhden palvelinprosessin sekä asiakassovelluksen. Palvelinprosessi hallinnoi tietokantaan tallennettua tietoa, hyväksyy asiakassovelluksien yhteydet tietokantaan, sekä suorittaa tietokannan muokkaukset asiakassovelluksien puolesta. Tietokantapalvelimen pääohjelmaa kutsutaan postgresiksi (PostgreSQL 2017).

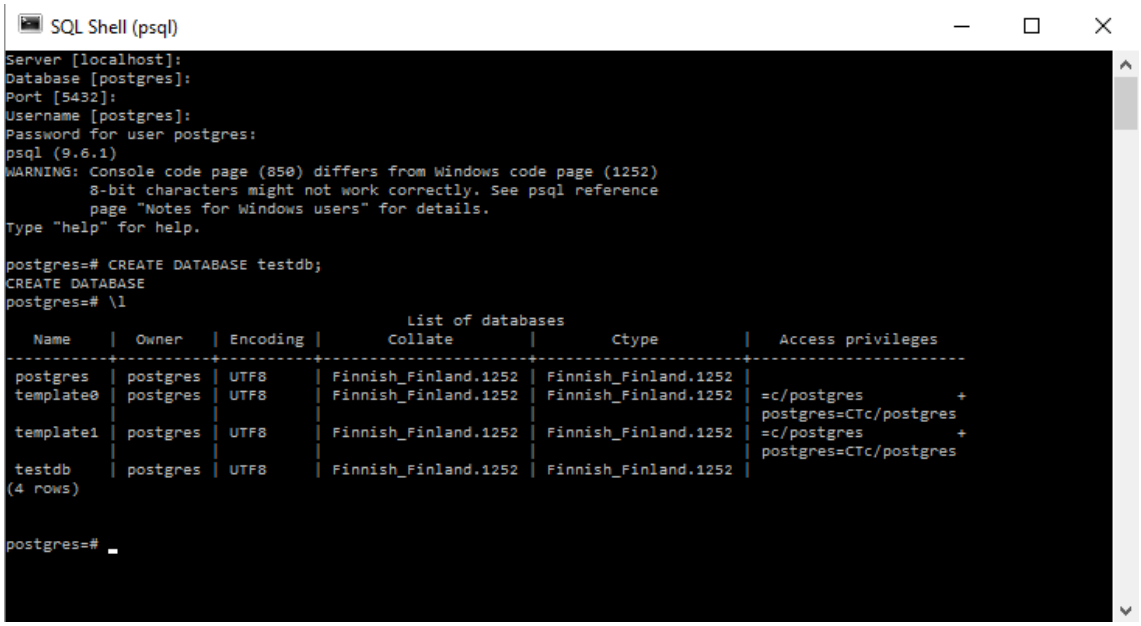
Asiakassovelluksien kautta suoritetaan muokkauksia ja toimenpiteitä tietokannassa. Asiakkaana toimivina sovelluksina voidaan pitää muun muassa web-sivuja tai toisia palvelimia. PostgreSQL löytyykin osasta palvelinohjelmistoja vakiona (PostgreSQL 2017).

PostgreSQL-järjestelmä kykenee käsittelemään useita asiakasyhteyksiä samanaikaisesti. Tätä varten käynnistetään uusi prosessi jokaista yhteyttä kohden. Prosessit keskustelevat toistensa kanssa riippumatta pääohjelmasta. Ohjelma pysyykin jatkuvasti käynnissä, prosesseja taas käynnistellään ja sammutellaan tarpeen mukaan (PostgreSQL 2017).

KUVA 6. Postgres-palvelimeen yhdistäminen komentorivisovelluksesta

PostgreSQL-järjestelmässä käytetään relaatiomalliin perustuvaa tietokannan hallintajärjestelmää (RDBMS). Toisin sanoen järjestelmässä tiedot tallennetaan relaatioihin (kuva 7). Relaatiot eroavat perinteisistä tauluista muun muassa duplikaattien tallentamisessa, mutta muilta osin lähes samankaltaisia. Postgresissä relaatioita muun muassa kutsutaan tauluiksi (PostgreSQL 2017).

Jokainen relaatio sisältää nimettyjä rivejä, jotka sisältävät samoin nimettyjä sarakkeita tietotyyppeineen. Sarakkeet järjestetään riveihin, kun taas rivit saattavat olla relaatiossa missä tahansa järjestyksessä. Taulukot järjestetään tietokantoihin. Usea Postgres-järjestelmän hallinnoima tietokanta taas muodostaa tietokantaklusterin (PostgreSQL 2017).



KUVA 7. Postgres-relaation luonti ja kutsuminen komentorivisovelluksella

4 KINO-HANKE

4.1 Hankkeen lähtökohdat

Kansainvälisen Innovaatio-osaamisen kehittämishanke KINO näytteli merkittävää osaa opinnäyteprojektissa. KINO-hanke oli jaettu viiteen eri sisällölliseen työpakettiin. Yhdessä paketissa käsiteltiinkin innovaatioaihioiden löytämisen ja jalostamisen menetelmiä.

Työpaketissa käsitellään laajemmin innovaatioaihioiden prosessia, huomioimalla toimintatavat ja mahdollisuudet innovaatioaihioiden löytämisessä sekä niiden jalostamisessa kaupallisesti hyödynnettäväksi. Innovaatioaihioiden löytämisessä keskitymme kolmen eri lähteen tarkasteluun;

- aikaisemmat tutkimustulokset, joita ei ole vielä hyödynnetty,
- tuotteiden ja palveluiden käyttäjäkokemukset
- Big Datan mahdollisuudet

(KINO-hanke, 2016)

Etenkin tuotteiden ja palveluiden käyttäjäkokemukset, sekä Big Datan mahdollisuudet soveltuivat Oiva-projektiin liittyvien parannuksien liittämiseen osaksi innovaatioaihiota. Lisäksi luominen mahdolliselle ekosysteemille, jossa korkeakoulut ja yritykset saadaan yhdessä löytämään uusia innovaatioaihioita, onnistui tätä kautta.

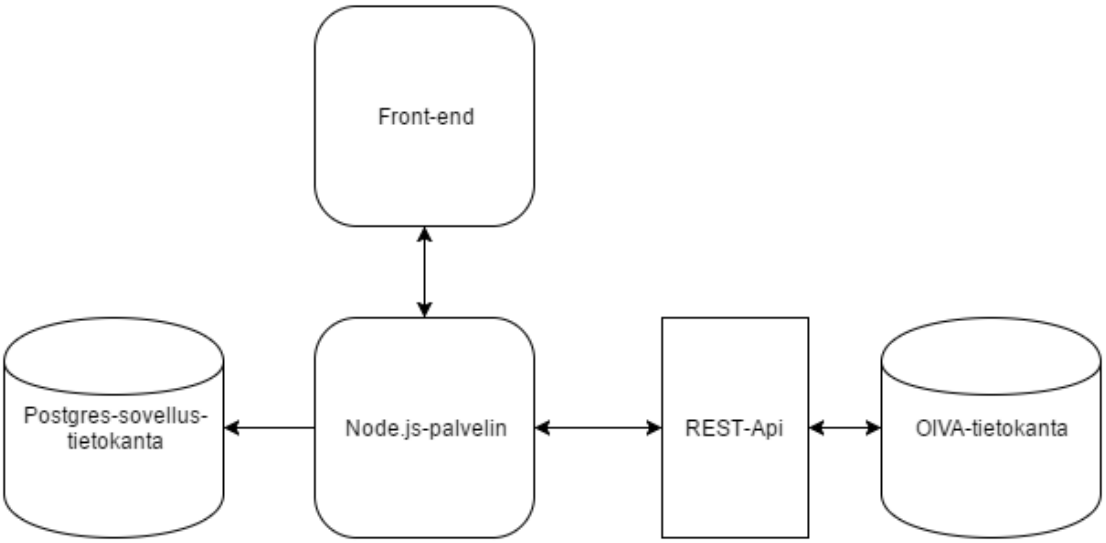
4.2 Kehitysaihion prototyypit

Projektin toteutusta suunniteltiin useaan otteeseen. Parhaan mahdollisen tuloksen varmistamiseksi, otettiin suunnittelupalavereihin mukaan myös alkuperäisen OIVA-tuotteen suunnitellut arkkitehti. Kun havaittujen lisäparannusten suunnitteluvaihe saatiin päätökseen, hajautettiin projekti pienempiin osakokonaisuuksiin. Näitä olivat toteutettavan sovelluksen tietokannan integroiminen Oiva-palvelun REST-rajapintaa vasten, sekä halutun datavisualisoinnin muodostaminen jollakin D3-kirjaston toteutuksista.

4.3 Oiva-palvelun integrointi

Oiva-palvelun tietokannasta oli jo ennakkoon tuotettu REST-rajapinta aikaisempaa käyttötarkoitusta varten. Tämä sopikin hyvin uuden projektin ja suunnittelun lähtökohdaksi. Oiva-palvelun tietokantapalvelimelle asetettua kuormaa haluttiin kuitenkin, optimaalisen toiminnallisuuden ja käyttönopeuden takia rajoittaa. Ratkaisussa onkin siis pyritty löytämään paras mahdollinen vaihtoehto suorille tietokantakyselyille.

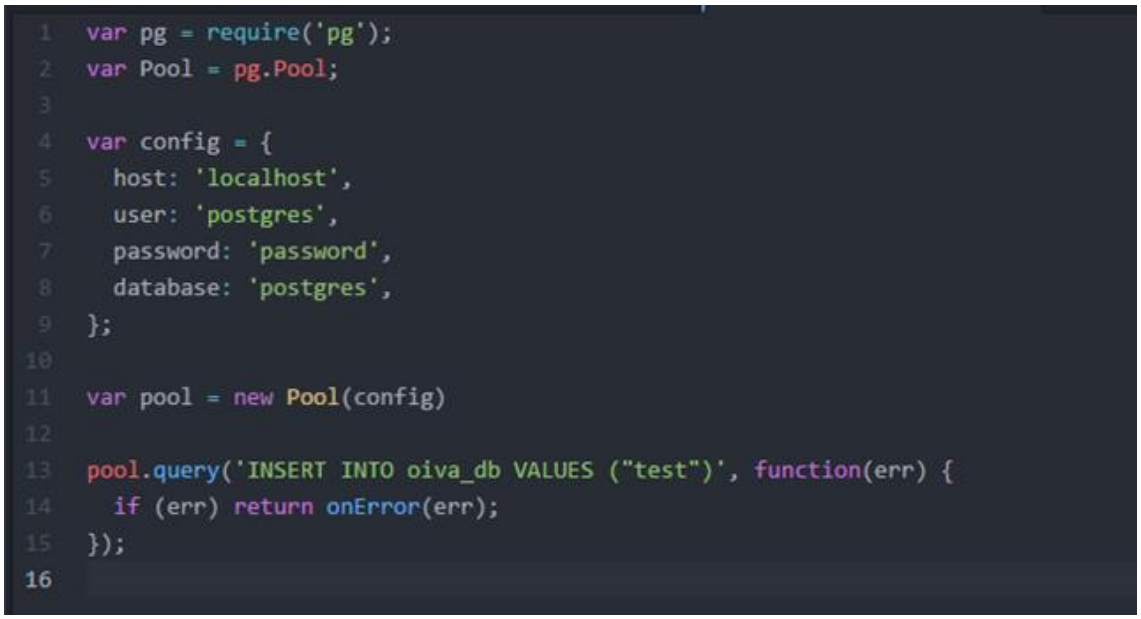
Sovellusten väliin luotiin niin kutsuttu staging-palvelin. Sen tarkoituksena on toimia kutsujen välittäjänä D3-kirjaston toteutuksien ja tietokannan välillä, sekä suorittaa ajoitettuja kyselyitä Oiva-tietokannalla, REST-rajapinnan välityksellä (kuva 8). Ajoitettujen kyselyiden avulla voidaan tietokannalle suorittaa migraatiota jatkuvalla syötöllä. Kyselyiden väli päätettiin ajoittaa sopivaksi mahdollisen rajapinnan kuvauksen mukaisesti, sekä tietokannan kuormaa rajoittaen.



KUVA 8. Vuokaavio sovelluksen tietokannan integroinnista REST-rajapintaa vasten

Staging-palvelin toteutettiin käyttäen Javascript-pohjaista Node.js-viitekehystä. Node.js-kehitykselle ominaiseen kehitystapaan lähdettiin valitsemaan mahdollisia Postgres-tietokannan käyttöä ja REST-rajapinnan kutsumista helpottavia npm-paketteja. Valinnassa päädyttiin käyttämään node-postgres-npm-pakettia. Paketti lisättiin projektiin käyttäen npm install pg – käskyä. Koodissa pakettia käytetään sijoittamalla ensin require(”pg”)-funktion paluuarvo valittuun muuttujaan. Require-funktio toimii moduulien lataajana. Pg-olion luokkien kautta voi nyt kutsua moduulin eri toiminnollisuuksia.

Postgres-tietokantayhteydet toimivat istuntojen välityksellä. Istuntojen luominen hoidetaankin tässä tapauksessa luomalla niin kutsutta pool-yhteys, jonka avulla eri tietokantaoperaatioita voidaan suorittaa. Pool-yhteys luodaan sijoittamalla pg-moduuli-luokan sisäkkäinen luokka Pool muuttujaan Pool. Tämän jälkeen määritetään halutut konfiguraatiot tietokantayhteyttä varten ja sidotaan ne config-olioon. Pool-yhteyttä päästäänkin seuraavaksi käyttämään, kun luodaan uusi Pool-luokan instanssi pool, jolle annettaan parametrina config-olio. pool-luokalle on määritetty useita tietokantaoperaatiofunktioita. Funktiolla query voidaan suorittaa lisäyksiä tietokantaan. Query-funktiolle annetaan parametreina SQL-lauseke. Vaihtoehtoisesti voidaan query-funktion sisällä suorittaa myös virheenkäsittelyä (kuva 9).



```
var pg = require('pg');
var Pool = pg.Pool;

var config = {
  host: 'localhost',
  user: 'postgres',
  password: 'password',
  database: 'postgres',
};

var pool = new Pool(config)

pool.query('INSERT INTO oiva_db VALUES ("test")', function(err) {
  if (err) return onError(err);
});
```

KUVA 9. Testiarvon lisääminen Postgres-tietokantaan Npm-paketin avulla

Seuraavana toteutettiin rajapinnan kommunikaatio Node.js-palvelimen kanssa. Tämä onnistuisi luomalla http-palvelin-instanssi ja lähettämällä sen kautta http-kutsuja. Käyttötarkoitusta varten löydettiin npm-paketti http. Seuraavaksi ladattiinkin moduuli samannimiseen muuttujaan, jotta sen luokkia voitiin käyttää jatkossa.

Http-palvelimen instanssi luodaan http-luokan createServer-funktion kautta. Oheisessa käyttötapauksessa instanssi sijoitetaan server-muuttujaan mahdollista jatkokäyttöä varten. CreateServer-funktio sijoitetaan lisäksi anonyymi-funktio, joka saa parametreinaan kysely- ja vastaus-oliot. Näiden kautta voidaan http-kutsuja välittää ja http-vastauksia vastaanottaa palvelininstanssin kautta.

Seuraavaksi työssä määriteltiin rajapintojen osoitteet. Yhteys muodostettiin Oiva-palvelun REST-rajapintaan, jonka yksilöimistä varten muodostettiin rajapinta-avain. Lisäksi tietoliikenneyhteyksiä muokattiin niin, että vain tietyistä osoitteista, kuten Tamkin verkoista, pääsisi kutsumaan rajapintaa. Huomattiin, että haluttua testaamista varten tuli hyväksyä cross-origin HTTP – kyselyiden tekeminen. Tämä johtui suoraan selaimessa pyörivien resurssien käyttämisestä. Näin kuitenkin saatiin mahdollistettua mahdollinen prototyypin testaus selainpuolella ilman Postgres-kantayhteyttä.

Cross-origin-kyselyt saatiin menemään läpi lisäämällä saatuun http-vastaukseen ylätunnisteteitä. Vastaukseen tuli lisätä alkuperään liittyvä ylätunniste, johon tuli asetta paikallisesti ajettavan selainpuolen osoite sekä portti. Seuraavaksi lisättiin kyselymetodeihin liittyvä ylätunniste, jonka kautta voitiin määrittää selainpuolelta tehtävien kyselyiden rajoitukset. Viimeisenä lisättiin ylätunniste, joka hyväksyisi kyselyltä palautuvan ylätunnisteen tyypin.

Vastaustenhallinnan kohdalla suoritettiin yksinkertaiset lisäykset. Kyselyn palautuessa tarkistetaan vastauksen statuskoodi ja error-muuttuja. Jos statuskoodi on arvoltaan 200 ja vastaus ei palauta virhettä, voidaan vastaus lähettää selainpuolelle. Jos kuitenkin havaitaan virhe, palautetaan selainpuolella ylätunniste 500 ja teksti ”An error occured” (kuva 10).

```
var server = http.createServer(function(req, res) {

  var api = 'http://localhost:3000';
  var oivaApi = '[illegible]';

  var onError = function(err) {
    console.log(err.message, err.stack)
    res.writeHead(500, {'content-type': 'text/plain'});
    res.end('An error occurred');
  };

  // request oiva
  request(oivaApi, function (error, response, body) {
    if (!error && response.statusCode == 200) {
      res.setHeader('Access-Control-Allow-Origin', api);
      res.setHeader('Access-Control-Allow-Methods', 'GET,PUT,POST,DELETE');
      res.setHeader('Access-Control-Allow-Headers', 'Content-Type');
      res.end(body);
    }
  })
});
```

KUVA 10. Oiva-rajapinnan kutsuminen http-palvelin-instanssin avulla

4.4 Sanapilvet

Yksi datan visualisaatiossa käytetyistä D3-toteutuksista oli sanapilvi eli Wordcloud-toteutus (kuva 11). Työssä selainpuolen visualisaatio päätettiinkin hoitaa valmiilla, Jason Daviesin d3-cloud-visuaalisaation toteutuspohjalla. Sanapilvi-toteutuksen suunniteltiin vastaanottavan parsittua dataa Postgres-tietokannasta ja muodostavan mahdollisista suosituista avainsanoista sanapilvi. Lisäarvoa työlle pyrittiin tuomaan mahdollisten trendien seuraamisesta ja ennustamista, sanapilvimäisen visualisaation avulla.

KUVA 11. Jason Daviesin wordcloud-toteutus

Toteutusta vartena tarvittiin niin kutsuttua html-kanvaasia, johon visualisaatio piirrettiin. Tämä saatiin otettua käyttöön lisäämällä canvas-npm-moduuli Canvas-olioon. Wordcloud D3-oliolle ei valmista npm-pakettia ollut, joten se tuli lisätä projektiin erikseen. Moduuli onkin lisätty käsin projektin ylätason indeksitiedostoon ja siihen päästään käsiksi sijoittamalla koko ylätaso cloud-olioon. Kyseiselle oliolle aletaankin D3-kirjastolle tyypillisesti ketjuttamaan funktioita.

Ensiksi määritellään html-elementin koko, syöttämällä cloud-olion size-funktiolle parametrina taulukko, joka sisältää elementin leveyden ja pituuden pikseleinä. Tämän jälkeen luodaan kyseisellä elementille html-kanvaasi, jolle tuleva visualisaatio voidaan piirtää. Kanvaasi muodostetaan syöttämällä uusi Canvas-olion instanssi cloud-olion canvas-

funktiolle. Seuraavaksi syötetään visualisoitava data piirrettävään sanapilveen. Tämä onnistuu sijoittamalla taulukko sanoja cloud-olion words-funktiolle. Työssä toteutetussa testitapauksessa syötettävät arvot kovakoodattiin taulukkoon words, joka annettiin parametrinä cloud-olion words-funktiolle.

Seuraavaksi muotoiltiin visualisaation tyylejä. Padding-funktiolla saatettiin muokata kanvaasin reunoja antamalla funktiolle jokin numeroarvo parametrina. Rotate-funktiolla saatiin visualisaatio kääntymään. Font-funktiolla taas voitiin visualisaatiolle asettaa mukautettu fontti. Lopuksi asetetaan niin kutsuttu end-kuuntelija, joka odottaa, kunnes visualisaation asettelu on saanut loppuun kaikkien sanojen sijoittelun. Kun nyt sijoittelun on valmis, voidaan visualisaatio laukaista start-funktiolla. Funktio muokkaa sanojen koon suurimmasta pienimpään ja pitää huolen, etteivät sanojen reunat osu päällekäin (kuva 12).

```
var Canvas = require("canvas");

var cloud = require("../");

var words = ["IOT","IOT","IOT","-datan","datatietojen","visualisointi","Visuali
"Virtuaalinen","todellisuus","ja","lisääntynyt","todellisuus","B","+","Markkino
"Ei-haihtunut","muisti","B","-","Vaikka","muutama","teknologiailmoitus","ei","t
"Cyber","Physical","Systems","(CPS)","B","+","CPS","ja","erityisesti","teolline
"Tietotieteet","Tietotieteen","edistyminen","johti","uuden","toimialan","asemaa
"Capability-pohjainen","turvallisuus","C","Ehkä","jätimme","eniten","tämän","te
"Advanced","Machine","Learning"];

cloud().size([960, 500])
    .canvas(function() { return new Canvas(1, 1); })
    .words(words)
    .padding(5)
    .rotate(function() { return ~~(Math.random() * 2) * 90; })
    .font("Impact")
    .on("end", end)
    .start();
```

KUVA 12. D3-esimerkkikoodi visualisaation muodostamisesta

5 POHDINTA

Projektin osanottajien mielestä lopputulemaa voidaan pitää onnistuneena. Vaikkakaan projektista ei syntynyt konkreettista toimivaa lopputuotetta, saatiin kasaan prototyyppien kautta mahdollisessa jatkokehityksessä käytettävää tai jalostettavaa tietoa. Ottaen huomioon, että työn päällimmäinen tarkoitus oli luoda ja löytää uusia innovaatiokehyksiä ja –ekosysteemejä, voidaan työn alkuperäisen tarkoituksen nähdä toteutuneen. Hyvin tuotetun raportoinnin ja dokumentoinnin pohjalta mahdollinen projektiin liittyvä jatkokehitys saadaankin jatkossa nopeasti käyntiin.

Projektin tuloksena saatiin kerättyä tietoa mahdollisista Oiva-palvelun tiedon tallentamiseen liittyvistä ongelmakohdista. Havaittuja ongelmia on käsitelty tiedon jäsentämisen ja yhtenäistämisen näkökulmasta, lähinnä datavisualisaatiota silmällä pitäen. Lopuksi jokaiseen ongelmakohtaan esitetään mahdollinen ratkaisu ja teoreettinen parannusehdotus. Palvelun nykyisen toiminnallisuuden ongelmakohtiin ei lopputuloksessa ole tarkoitus ottaa kantaa.

Muita tuloksia olivat työskentelyn ohessa tuotetut prototyyppikoodit. Näiden kohdalla valmistettua projektikoodeja tuotettiin kaksi kappaletta. Toinen prototyyppikoodeista koodeista käsitteli Oiva-palvelun integrointia ulkoiseen sovellukseen ja toinen taas D3-kirjaston sanapilvitoteutuksen käyttöönottoa Oiva-palvelusta jäsennetyn tiedon datavisualisoinnissa. Molempien prototyyppikoodien valmistus käydään läpi yksityiskohtaisesti opinnäytetyöraportin aikana.

Työskentely projektissa onnistui projektin jäsen mielestä luonnikkaasti. Suurimmat työskentelyyn vaikuttuvat ongelmat liittyivät lähinnä omaan motivaatioon. Projektin aikataulutus koettiin ongelmallisena ja ajan voidaan sanoa loppuneen kesken. Toisaalta tämä jättää mahdollisuuden seuraavalle jatkokehitykselle ja aiheen työhön, vaikka jollekin seuraavalle opiskelijalle.

Projektin aikana ohjausta ja ohjeistusta saatiin niin opettajilta, kun työelämän edustajilta. Projektin aikana käytiin palavereja useaan otteeseen. Osaa palavereja käytiin myös vastuuyrityksen kanssa. Tämän avulla työn spesifikaation hahmottaminen ja lopputuotteen suunnittelu helpottui huomattavasti.

Toteutuneen pohjalta työhön jäi vielä jonkin verran varaa jatkokehitykselle. Seuraavaksi voitaisiinkin pohtia, kuinka mahdolliset parannusehdotukset toteutettaisiin Oiva-palveluun. Prototyyppikehitykseen voitaisiin lisätä yksityiskohtaisesti prototyyppi jokaisesta parannusehdotuksesta. Tällöin tulisi todennäköisesti ottaa huomioon myös niin sanotun testiympäristön perustaminen. Työssä testaamiseen käytettiin Oiva-palvelun testirajapintaa. Jatkossa tulisikin ehkä miettiä, kuinka mahdollinen selainpuolten muutosten testaaminen ja testiympäristön pystytys toteutetaan.

LÄHTEET

Kansainvälisen Innovaatio-osaamisen kehittäminen Tampereen korkeakouluissa projektisuunnitelma, KINO-hanke 2016.

React.js Essentials. Dzone 2017. Luettu 4.4.2017.
https://dzone.com/refcardz/reactjs-essentials

Suomen väestörakenne. Tilastokeskus 2016. Luettu 12.4.2017.
http://www.tilastokeskus.fi/tup/suoluk/suoluk_vaesto.html

Mozilla Developer Network 2017. Luettu 3.4.2017.

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

https://mdn.mozillademos.org/files/13502/cake.png

React. Facebook 2017. Luettu 4.4.2017.

https://facebook.github.io/react/

Node. NodeJS Foundation 2017. Luettu 5.4.2017.

https://nodejs.org/en/

D3. Data-Driven Documents 2017. Luettu 5.4.2017.

https://github.com/d3/d3/wiki

PostgreSQL. The PostgreSQL Global Development Group. Luettu 5.4.2017

https://www.postgresql.org/docs/9.6/static/index.html

High-level language definition. Dictionary 2011. Luettu 12.4.2017.

http://www.dictionary.com/browse/high-level-language

Suomen väestötiheyskuva. Oona Räisänen 2010.

https://upload.wikimedia.org/wikipedia/commons/thumb/0/0e/Population_map_of_Finland.svg/300px-Population_map_of_Finland.svg.png