Miika Ahonen

Web-pohjainen kyselysovellus

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikan tutkinto-ohjelma

Insinöörityö

14.4.2018

Metropolia

Tekijä Otsikko Sivumäärä Aika	Miika Ahonen Web-pohjainen kyselysovellus 42 sivua + 3 liitettä 14.4.2018
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	tieto- ja viestintätekniikka
Ammatillinen pääaine	mediatekniikka
Ohjaaja	Yliopettaja Kari Aaltonen

Insinöörityön tarkoitus oli kehittää asiakkaalle selaimessa toimiva järjestelmä, jolla pystytään luomaan kyselyitä, jakamaan kyselyitä helposti osallistujille ja muodostamaan kyselyn tuloksista helposti luettavia kuvaajia.

Järjestelmän lopullinen versio toteutettiin käyttämällä moderneja JavaScript-ratkaisuja kuten ECMAScript 6–8 -määrittelyjä, React-kirjastoa näkymien luontiin ja Koa-kehystä palvelinpuolen koodissa. Kyselyiden tulosten graafista esitystä varten luotiin kuvaajat käyttäen D3-JavaScript-visualisointikirjastoa.

Järjestelmän palvelinpuolta alettiin rakentaa Sails-kehyksellä, mutta sen kanssa ilmenneiden ongelmien vuoksi päädyttiin vaihtamaan kehys Koaan. Selainpuolen näkymät toteutettiin käyttäen JavaScriptiä ilman mitään kehystä. Palvelinpuolen kehyksen vaihdon yhteydessä päätettiin, että selainpuolen kehykseksi otetaan käyttöön React, koska se on hyvin yhteensopiva Koan kanssa. Reactin käyttö selainpuolella mahdollisti helpommin kehitettävät dynaamiset sivut, ja jatkokehitystä ajatellen koodista tuli helpommin ylläpidettävää, vaikka Reactin käytöstä ei ollut aikaisempaa kokemusta.

Järjestelmä saatiin valmiiksi kaikilla sovituilla ominaisuuksilla sekä muilla jälkikäteen toivotuilla toiminnoilla. Kehitysaikaa olisi voinut lyhentää, jos teknologiapino olisi alusta alkaen pysynyt samana.

Avainsanat	web-ohjelmointi, sovelluskehitys, JavaScript, D3, EJS, React, ECMAScript, Foundation, SCSS

Author	Miika Ahonen
Title	A Web-based survey application
Number of Pages	42 pages + 3 appendices
Date	14 April 2018
Degree	Bachelor of Engineering
Degree Programme	Information and Communication Technology
Professional Major	Media Technology
Instructor	Kari Aaltonen, Principal Lecturer

The purpose of this thesis was to develop a web-application for the customer where they could create surveys and easily share them with answerers. The application also needed to be able to create graphs from the survey results.

The final version of the application was developed using modern JavaScript solutions such as ECMAScript 6–8 -standards, React-library for creating views and Koa-framework for the back-end code. Graphs in the survey results page were built with D3-JavaScript visualization library.

The back-end framework was originally Sails, but problems with it made us change the framework to Koa later in the project. All front-end views were developed using JavaScript without frameworks. With the back-end framework change to Koa, some of the more dynamic front-end views were redone with React because of its compatibility with Koa. The change to React made the front-end code in the more dynamic views easier to develop and maintain in the future, even though I had no previous experience using it.

The application was finished with all the agreed features, as well as some extra features that were discussed in meetings during the development. The development time could have been reduced if the technology stack was not altered as much as it was during the project.

Keywords	Web-development, JavaScript, D3, EJS, React, ECMAScript, Foundation, SCSS

Sisällys

Liitteet

Lyhenteet ja käsitteet

CSS	Cascading Style Sheet. Merkintäkieli, joka määrittelee verkkosivun tyyliasun.
DOM	Document Object Model. Tapa, jolla HTML-sivut näytetään selaimessa. Siinä verkkosivu on dokumentti ja elementit ovat objekteja sen sisällä.
HTML	Hypertext Markup Language. Merkintäkieli, jolla usein luodaan verkkosivuja.
JavaScript	Skriptauskieli, jolla luodaan dynaamista sisältöä verkkosivuille.
JSON	JavaScript Object Notation. Yleinen tapa väittää ja säilyttää tietoa.
Node	Node.js. JavaScript-runtime-ympäristö, joka mahdollistaa JavaScript-koodin suorittamisen palvelimella.
NoSQL	Käsite, jolla kuvataan relaatiomallista poikkeavia tietokantoja.
Sass	Syntatically awesome stylesheets. CSS laajennus, jonka avulla tyylitiedostoissa voi käyttää muuttujia yms.
SCSS	Sassy CSS. Sassin uudempi versio, jossa on CSS:n tapainen syntaksi.
SVG	Scalable Vector Graphics. Tällä luodaan vektoripohjaista grafiikkaa verkkoon.
SQL	Structured Query Language. Kyselykieli, jota käytetään relaatiotietokantojen kanssa.

1 Johdanto

Insinöörityön tarkoituksena on kehittää tilaajalle selaimessa toimiva kyselyidenluontisovellus. Alun perin sovelluksen tarkoitus oli olla vain yhdenlaisten kyselyiden luontiin ja jakamiseen. Näissä kyselyissä oli 28 ennalta määrättyä kysymystä, joista sai valita halutut kysymykset kyselyyn. Jokaisella kysymyksellä on samat kolme kriteeriä: tärkeys, nykytilanne ja kiireellisyys. Näiden kyselyiden vastausten perusteella tulee saada selville, millä eri vastaajaryhmillä on toisistaan eroavia vastauksia, mistä voi päätellä, miten hyvin ryhmät pystyvät auttamaan toisiaan puutteellisten osa-alueidensa laadun kehityksessä. Myöhemmin projektissa kuitenkin päätettiin tehdä kyselyistä vapaampia, eli käyttäjät voivat luoda omat kysymykset ja kriteerit, mutta edellä mainittujen kysymysten tulee silti löytyä järjestelmästä.

Sovelluksella voi luoda käyttäjän, joka kuuluu käyttäjäryhmään, joko itse luotuun tai jonkun muun luomaan. Ennen kyselyn luontia täytyy luoda arviointiryhmä, joka koostuu käyttäjäryhmistä. Vain tämän arviointiryhmän alla olevien käyttäjäryhmien jäsenet pääsevät käsiksi kyselyyn. Kun kysely on luotu, sen voi jakaa joko pelkästään sovellukseen kirjautuneille käyttäjille tai jakaa linkillä esim. sähköpostitse. Jokaiselle käyttäjäryhmälle luodaan eri linkki, jotta vastaukset ryhmittyvät oikein tuloksia tarkasteltaessa.

Projektin tilaajia ovat Metropolia Ammattikorkeakoulun lehtori Katriina Schrey-Niemenmaa ja yliopettaja Markku Karhu. Katriina Schrey-Niemenmaa on jo 1990-luvulta asti järjestänyt näitä kyselyitä oppilaitosten kesken jakamalla vastaajille kyselyn Excel-muodossa levykkeillä. Sovelluksen on tarkoitus helpottaa näiden kyselyiden järjestämistä, niihin vastaamista ja tulosten koontia ja tarkastelua.

Insinöörityöraportissa käsitellään tämän projektin taustaa enemmän: mikä aikaisemmin mainittujen 28 kysymyksen tarkoitus on, miten dataa visualisoidaan yleisesti ja miten se toteutetaan JavaScriptillä, mitä teknologioita projektissa käytettiin ja miltä yksi esimerkkikäyttötapaus näyttää järjestelmää käytettäessä.

Projektista tehdään kaksi insinöörityöraporttia. Tässä raportissa keskitytään enemmän selainpuolen teknologiaan, datan visualisointiin ja käyttötapaukseen. Toinen insinöörityöraportti, jonka kirjoitti Tommi Pälviö, on kirjoitettu englanniksi, ja siinä keskitytään enemmän palvelinpuolen ratkaisuihin ja koodin testaukseen.

2 Projektin taustaa

2.1 Korkeakoulujen Laatu -malli

Korkeakoulujen Laatu (KOLA) on vuonna 1992 alkaneen hankkeen tulos, jonka on kehittänyt Tekniikan Akateemisten Liitto. Hankkeella kehitettiin diplomi-insinöörikoulutuksen koulutusohjelmatasoinen laatujärjestelmä, joka mukaili Malcom Baldridge -laatupalkintomallia. Laatujärjestelmän tueksi kehitettiin Excel-pohjaiset työkalut itsearvioinnille ja sidosryhmäarvioinnille, esim. opiskelijapalautteelle. Lisäksi luotiin malli vertaisarvioinnille ja konsensuskeskustelulle, joka yhdistää nämä kaikki asiat. [1; 2.]

2000-luvulla EVTEKin (Espoon-Vantaan teknillinen ammattikorkeakoulu) sisällä työkaluja modernisoitiin verkossa toimivan itsearviointilomakkeen myötä. Sitä hyödynnettiin kaikkien sen ajan tekniikan alan koulutusohjelmien arvioinnissa, ja sen avulla luotiin EVTEKin itsearviointiraportti. Myöhemmin KOLA-malli oli mukana erinäisissä kansainvälisissä ja EU-tasoisissa hankkeissa. Itsearviointi ja vertaisarviointi laajenivat ja täydentyivät EU:n ERASMUS+:n QAEMP-hankkeen myötä ns. cross-sparrauskonsepteiksi. [1.]

Cross-sparrauksella tarkoitetaan prosessia, jossa kaksi organisaatiota, esimerkiksi oppilaitosta, kokoontuvat yhteen ja parantavat haluamiensa alueiden laatua. Organisaatiot päättävät arvioitavat kriteerit yhdessä, minkä jälkeen toinen organisaatio arvioi toisen ryhmän rehellisesti. Kun toinen ryhmä on arvioitu, puolet vaihdetaan niin, että molemmat ryhmät ovat arvioineet ja tulleet arvioiduksi. Sparrausprosessi vaatii rehellisyyttä molemmilta osapuolilta, jotta prosessista saadaan parhaat mahdolliset tulokset. Sopivan sparrauskumppanin löytämistä varten on olemassa valikoima kysymyksiä, joista voi luoda kyselyn, jonka tulosten perusteella päätetään sopivat organisaatiot sparraukseen. Nämä kysymykset ovat ns. QAEMP-kysymyksiä (Quality Assurance and Enhancement Marketplace). [3.]

2.2 Quality Assurance and Enhancement Marketplace -kysymykset

Quality Assurance and Enhancement Marketplace eli QAEMP on laadunkehittämismalli, joka korostaa jakamista ja jatkuvaa laadun parantamista. Se ehdottaa käytännönläheistä tapaa laadun kehitykseen. Ehdotus perustuu cross-sparrauskonseptiin, joka tekee palautteesta konkreettista, kollaboratiivista ja puolueetonta. [3.]

QAEMP-kysymykset koostuvat 28 englanninkielisestä kysymyksestä, jotka on jaettu 10 eri alueeseen (taulukko 1). Jokaisella kysymyksellä on hyvin tarkkaan luotu perusteluteksti, joka antaa vastaajalle tarpeeksi tietoa, jotta hän voi arvioida kunkin aihealueen oikein. Jokainen alue arvioidaan asteikolla 0–5, ja jokaisella asteikon asteella on yksityiskohtainen kuvaus, joka auttaa vastaajaa valitsemaan oikean asteikon arvon. [4.]

Taulukko 1. QAEMP-kysymysten alueet ja kysymysten määrät [4].

Alue	**Kysymysten määrä**
Tutkinto-ohjelman filosofia	1
Tutkinto-ohjelman pohja	4
Oppiminen ja opetus	5
Arviointi ja palaute	2
Taitojen kehitys	4
Työllisyys	2
Tutkimustyö	1
Opiskelijan keskittyminen	4
Henkilökunnan kehitys	2
Arviointi	3

Kysymysten tarkoitus on auttaa tutkinto-ohjelmaa löytämään omat keskeisimmät kehittämiskohteensa ja sopivan sparrauskumppanin, jonka avulla voisi nopeuttaa kehittymistään. Kysymykset keskittyvät enemmän tutkinto-ohjelmassa toteutettaviin asioihin toisin kuin muissa arvioinneissa, joissa tarkastellaan korkeakoulutasoa ja sen hallinnointia. [1.]

2.3 Tarve verkossa toimivalle järjestelmälle

Kyselyitä järjestettiin pitkän aikaa menemällä korkeakouluihin paikan päälle sovittuun aikaan tiettyjen henkilöiden kanssa. Kyselyt järjestettiin niin, että jokaiselle oppilaitokselle annettiin levyke, joka sisälsi kyselyn Excel-muodossa. Kyselyihin vastattiin täyttämällä Excel-lomake ryhmissä, joiden sisällä ryhmän jäsenet käyttivät aikaa päättääkseen haluamansa vastauksen. Kun vastaukset olivat kerätty, ne koottiin manuaalisesti yhteen Excel-tiedostoon ja tehtiin tarvittavat arvioinnit sitä kautta. [1.]

Tämän prosessin helpottamiseksi vuonna 2009 Metropolian sisällä alettiin kehittää selaimessa toimivaa eKola-nimistä järjestelmää, jolla voi luoda kyselyitä käyttäen itse luotuja kysymyksiä. Järjestelmä osasi tehdä kyselyn tuloksista kuvaajia ja näyttää tilastoja vastauksista. Järjestelmän kehitys lakkasi ajan puutteen takia, eikä sitä ei voitu myöhemmin jatkaa, koska järjestelmän tekninen dokumentaatio oli olematon ja alkuperäiset tekijät eivät olleet enää kehittämässä järjestelmää. [1.]

Verkossa toimivan järjestelmän tarkoitus on tarjota samanlaiset hyödyt kuin digitalisoinnilla yleisestikin tarjotaan: kyselyiden tiedot pysyvät paremmin tallessa tietokannassa, josta voi tarvittaessa ottaa varmuuskopioita. Lisäksi, kunhan järjestelmää käytetään tarpeeksi, vanhojen kyselyiden tallessapito mahdollistaa uusien ja vanhojen kyselyiden tulosten vertailun. Näin voidaan esimerkiksi katsoa, miten tietty organisaatio on onnistunut kehittämään itseään ajan myötä. Tämänkaltainen vertailu on mahdollista, jos uudessa kyselyssä on samat kysymykset kuin vanhassa kyselyssä. [1.]

3 Datan visualisointi

3.1 Datan visualisointi yleisesti

Datan visualisoinnilla tarkoitetaan tiedon esittämistä kuvaajilla tai muulla helposti ymmärrettävällä graafisella tavalla. Visualisointia käytetään yleensä, kun käsitellään paljon numeerista dataa, joka ei ole ihmisluettavaa. Ennen internetiä datan visualisoinnilla viitattiin yleensä staattisiin kuvaajiin esimerkiksi kirjoissa tai lehdissä. Nykyteknologian avulla, kuten JavaScript, kuvaajista voi tehdä interaktiivisia, eli käyttäjä voi esimerkiksi tietokoneen hiirellä tarkastella kuvaajan pisteitä ja saada tarkempaa tietoa jokaisesta pisteestä.

3.2 Datan visualisointi JavaScriptillä

Dataa voi nykyään visualisoida monella tavalla esimerkiksi Microsoftin Excel-ohjelmalla, mutta jos haluaa kehittää omia kuvaajia tai jakaa visualisoinnit helposti maailmalle, paras vaihtoehto on käyttää jotain JavaScript-pohjaista visualisointikirjastoa, kuten D3 tai Chart.js. JavaScriptillä tehdyt visualisoinnit antavat myös mahdollisuuden animoituihin ja käyttäjän kanssa interaktiivisiin kuvaajiin. Muillakin ohjelmointikielillä on kirjastoja visualisointiin, mutta yksikään ei ole yhtä suosittu kuin JavaScript. Suosioon voi vaikuttaa se, että JavaScriptillä tehdyt ohjelmat ovat verkossa ja verkkosivut ovat helpoin tapa jakaa tietoa ympäri maailmaa.

JavaScriptillä on tehty visualisointeja jo vuodesta 2002 FusionCharts-nimisellä ohjelmistolla. FusionCharts luotiin, kun sen alkuperäinen kehittäjä Pallad Nadhani ei ollut tyytyväinen Excelin kuvaajaominaisuuksiin. [5.]

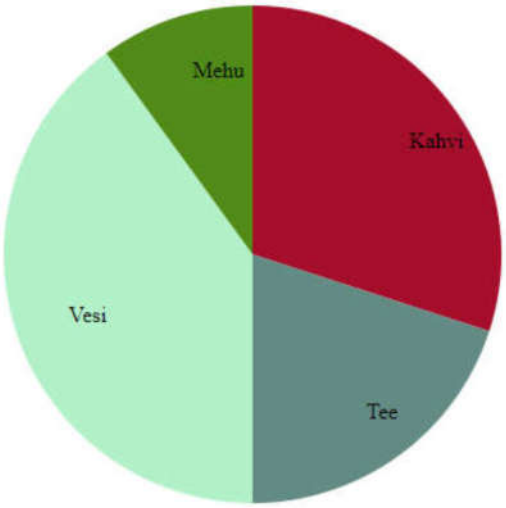
D3-JavaScript-visualisointikirjasto

D3 (Data-Driven Documents) on JavaScript-pohjaisista visualisointikirjastoista suosituin. Sillä on tätä kirjoitettaessa yli 73 000 tähteä GitHubissa, mikä tekee siitä suosituimman visualisointikirjaston. D3 on avoimen lähdekoodin kirjasto, jota on kehitetty vuodesta 2010 lähtien. Data-Driven Documents -termi viittaa siihen, että kirjasto käsitelee dokumentteja (HTML-dokumentteja) annetun datan mukaan näyttämään visualisointeja. D3

käyttää JavaScriptin lisäksi HTML:ää, CSS:ää ja SVG:tä (Scalable Vector Graphics) visualisointien esittämiseen. [6.]

Ensimmäinen versio D3:sta julkaistiin helmikuussa 2011. D3 on Protovis-nimisen kirjaston suora jälkeläinen, ja niillä on sama pääkehittäjä, Mike Bostock. Koska kirjastot ovat saman kehittäjän, niissä on myös paljon samankaltaisuuksia, mikä teki Protovisistä siirtymisen D3:een helpompaa. [7.]

D3 ei itsenään tarjoa valmiita visualisointeja, kuten monet muista suosituista JavaScript-pohjaisista datan visualisointikirjastoista, minkä takia yksinkertainenkin kuvaaja (kuva 1) vaatii melko paljon koodirivejä, kuten liitteessä 1 olevassa esimerkkikoodissa huomaa. Olemassa on useita melko suosittuja visualisointikirjastoja, jotka on rakennettu D3:lla ja jotka tarjoavat valmiita visualisointeja. Kirjaston suosio kuitenkin tulee sen muokattavuudesta, mikä valitettavasti tarkoittaa, että koodia tulee paljon yksinkertaisiinkin kuvaajiin. [8.]



Kuva 1. Liitteen 1 koodissa D3-kirjastolla luotu kuvaaja.

Koska D3 on niin vapaasti muokattavissa, sen käyttötapaukset eivät lopu vain datan visualisointiin, vaan sillä voi tehdä esimerkiksi yksinkertaisia pelejä, fysiikkasimulaatioita ja muuta interaktiivista sisältöä [10].

Koska D3 on tarkoitettu muokattavaksi omaan käyttöön, monet kehittäjät ovat tehneet sille hyödyllisiä liitännäisiä ja jakaneet ne kaikkien käytettäväksi. Näihin liitännäisiin

kuuluu valmiita kuvaajia, parempia aputekstejä kuvaajille ja monenlaisia uusia ominaisuuksia. [11.]

Vaikka D3 on kattavin JavaScript-kirjasto datan visualisointiin, kehittäjän kuitenkin kannattaa ensin tutkia muita vaihtoehtoja. Näin hän voi säästyä turhalta työltä. Jos kehittäjä ei kuitenkaan löydä kirjastoa, joka vastaa hänen tarpeitaan, on hyvä käyttää D3:a. [12.]

4 Selainkäyttöliittymässä käytetyt teknologiaratkaisut

4.1 Staattisten näkymien luonti EJS-sivupohjakielellä

EJS (Embedded JavaScript) on sivupohjakieli, joka yhdistää datan sivupohjan kanssa luodakseen HTML:ää [13]. Sivupohjien tarkoitus on vähentää toistuvaa koodia siten, että kehittäjä voi jakaa sivun pienempiin osiin, kuten sivun navigaatio ja alapalkki. EJS:ssä oletuksena kohdat, joissa suoritetaan JavaScript-koodia, merkitään `<% %>` -merkeillä. Jos sivulle tulostetaan dataa, kuten otsikko, se tulee kirjoittaa `<%= %>` -merkkien sisään. EJS kuitenkin antaa kehittäjän vaihtaa merkit, jos ne eivät ole mieluisia. [14.]

EJS:ää voi käyttää sekä selainpuolella että palvelinpuolella, mutta tässä projektissa sitä käytetään vain palvelinpuolella sivujen renderöintiin tietokannasta saadun datan kanssa (esimerkkikoodi 1).

```
--------Kontrolleri koa.js-kehyksellä-----------
static async profile(ctx, next) {
  // Haetaan käyttäjä tietokannasta user muuttujaan
  const user = await User.find(ctx.session.user.id);
  // Renderöidään näkymä EJS:llä antamalla sille EJS-pohjan sijainti ja data
  return ctx.view('user/profile.ejs', user);
}
--------EJS-tiedosto--------
<div>
  <label>Username</label>
  <p><%= user.username %></p>
  <label>Email</label>
  <p><%= user.email %></p>
  <ul>
  <% user.groups.forEach((group)=>{%>
     <li><%= group.name %></li>
  <%})%>
  </ul>
</div>

-------HTML, jonka EJS muodostaa-----
<div>
  <label>Username</label>
  <p>tester</p>
  <label>Email</label>
  <p>tst@example.com</p>
  <ul>
    <li>Group 1</li>
    <li>Group 2</li>
    <li>Group 3</li>
  </ul>
</div>
```

Esimerkkikoodi 1. Esimerkki siitä, miten palvelimelta saadaan muodostettua dynaamisesti luotua HTML:ää.

Esimerkkikoodin 1 ensimmäisessä osassa on esimerkki käyttäjän profiilisivun kontrolleri-sta. Siinä haetaan tietokannasta nykyisen käyttäjän profiili `user`-objektiin. Kun käyttäjän tiedot on haettu, renderöidään näkymä antamalla EJS-tiedoston sijainti ja `user`-objekti `ctx.view()`-funktiolle. Dataa voi antaa niin paljon kuin tarvitsee, mutta tässä esimerkissä annetaan vain käyttäjän tiedot.

Esimerkin toisessa kohdassa on lyhyt esimerkki EJS-syntaksista. Koodissa sivulle tulostetaan käyttäjän tunnus, sähköpostiosoite ja lista käyttäjän ryhmistä. Tässä esimerkissä käyttäjän tunnus on ”tester”, sähköpostiosoite on ”tst@example.com” ja käyttäjällä on kolme ryhmää: Group 1, 2 ja 3. EJS-tiedostossa pääsee käsiksi palvelimelta saatuun dataan käyttämällä samoja muuttujien nimiä, eli tässä esimerkissä kontrollerissa annetaan `user`-niminen objekti, ja siihen pääsee samalla nimellä käsiksi. Esimerkissä listataan ryhmät luomalla `forEach`-silmukka, jonka jokaisella kierroksella luodaan `<li>`-elementti, jonka sisälle tulostetaan ryhmän nimi.

EJS:n lisäksi suosittuihin sivupohjakieliin kuuluvat mm. Mustache, Handlebars ja Pug. Mustache on yksi tunnetuimmista, sillä se toimii JavaScriptin lisäksi muillakin ohjelmointikielillä, kuten PHP:llä. Handlebars on Mustachen seuraaja, ja kumpaakin voi käyttää samanaikaisesti vaihtelemalla niiden tunnistemerkkejä. Pug on sivupohjakieli, joka pyrkii saamaan koodin näyttämään mahdollisimman yksinkertaiselta ja luettavalta. Pug-syntaksi tekee HTML:n kirjoittamisesta tavallista nopeampaa, koska koodista jätetään kulmasulkeet kokonaan pois. Elementtien lapset määritellään koodin sisennyksillä, kuten Python-kielellä. [15.]

4.2 Dynaamisten sivujen luonti React-kirjastolla

React on Facebookin kehittämä avoimen lähdekoodin JavaScript-kirjasto, joka on pääasiassa tarkoitettu ns. SPA-sovelluksille (Single-Page Application) eli sovelluksille, joilla on käytännössä vain yksi ladattava sivu [16]. Vaikka tämä insinöörityöprojekti ei ole SPA-sovellus, Reactia voidaan silti hyödyntää sivuilla, joissa esimerkiksi tehdään tietokantaan kyselyitä siten, että sivun sisältö muuttuu ilman sivun uudelleenlatausta. Projektissa käytetään Reactia mm. kyselyn muokkaussivulla, ryhmän muokkaussivulla ja kyselyiden listaussivuilla.

React perustuu ajatukseen, että DOM-manipulaatio on selaimelle raskas operaatio ja se täytyy minimoida. React ratkaisee tämän tarjoamalla kehittäjälle tietokoneen muistissa asuvan virtuaalisen DOMin varsinaisen selaimen DOMin sijaan. React osaa vertailla virtuaalisen ja varsinaisen DOMin eroja, ja kun eroavaisuuksia löytyy, se osaa päivittää vain eroavan kohdan DOMista mahdollisimman pienellä rasituksella selaimelle. [16.]

React on tehokkuutensa ansiosta nykyään yksi suosituimmista selainpuolen JavaScript-kirjastoista. Sitä käyttävät mm. videopalvelut Netflix ja HBO Nordic, chat-sovellus Slack ja verkkokaupat eBay ja Verkkokauppa.com. Muita suosittuja selainpuolen kirjastoja ovat mm. BackboneJS, KnockoutJS ja Googlen kehittämä AngularJS [17, s. 2].

Verkkosivuja kehittäessä sivut jaetaan yleensä helposti ymmärrettäviin osioihin, kuten navigaatio, sisältö, artikkeli, taulukko, jne. Reactissa nämä osiot ovat komponentteja. React-sivu on jo itsessään yksi iso komponentti, jonka alla on pienempiä komponentteja, joiden alla voi olla vielä enemmän pienempiä komponentteja. React antaa kehittäjälle melko vapaat kädet kehityksessä, joten komponenttien määrällä ei ole käytännössä mitään rajaa. [18, s. 69.]

Komponentti voi antaa lapsikomponenteilleen attribuutteja (props), johon lapsi pääsee käsiksi ”props”-objektin kautta. Data, jota yleensä annetaan lapsikomponenteille, on HTML-elementin id tai CSS-luokka, mutta kehittäjä päättää itse, millaista dataa haluaa välittää komponenttien välillä. Komponentteja voi tehdä kahdella eri tavalla, joko funktionaalisesti tai luokallisesti (esimerkkikoodi 2). Funktionaaliset komponentit on tarkoitettu ensisijaisesti pienille yksinkertaisille komponenteille, sillä niiden sisään ei voi kirjoittaa funktioita, toisin kuin luokallisiin komponentteihin. Funktionaalisia komponentteja voi myös kutsua tilattomiksi (stateless) komponenteiksi, sillä ne eivät salli komponentin tilan (state) muokkaamista.

```
// Ladataan tarvittavat tiedostot
import ReactDOM             from 'react-dom';
import React, { Component } from 'react';

// Esimerkki funktionaalisesta komponentista
function Greeting(props) {
  return(
    <div className="greeting-text">
      <h1>{props.text} World</h1>
      <a href={props.link}>Read more...</a>
    </div>
  );
}

// Esimerkki ES6 luokkakomponentista
```

```
class MainComponent extends Component {
  render() {
    return (
      <div id="greeting-container">
        <Greeting text="Hello" link="www.metropolia.fi"/>
        <Greeting text="Greetings" link="www.facebook.com"/>
        <Greeting text="Bye" link="www.google.com"/>
      </div>
    );
  }
}

// Renderöidään pääkomponentti 'react-root'-nimisen HTML-elementin sisään
ReactDOM.render(<MainComponent />, document.getElementById('react-root'));
```

Esimerkkikoodi 2. Komponentti tehty funktionaalisella tavalla ja käyttäen ES6-luokkaa.

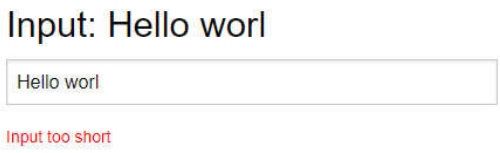
Esimerkkikoodissa 2 `<MainComponent/>` toimii pääkomponenttina, jonka lapsikomponentti on `<Greeting/>`. Pääkomponentti antaa lapsikomponenteilleen `link`- ja `text`-nimiset attribuutit. `text`-attribuutti määrittää kunkin lapsikomponentin `<h1>`-elementin tekstin, ja `link`-attribuutti määrittää `<a>`-elementin osoitteen. Esimerkistä voi huomata, kuinka paljon React-komponenteilla voi säästää toistettavaa koodia, kun paketoi koodia komponentteihin.

Yksi Reactin tärkeimmistä ominaisuuksista on komponentin tilan muokkaus. Koska komponentin props-attribuutit ovat aina muuttumatonta dataa, tarvitaan muokattava tila. Joka kerta kun komponentin tilaa muokataan, komponentin `render`-funktio suoritetaan automaattisesti uudelleen, mikä myös antaa lapsikomponenteilleen päivitetyt attribuutit [18, s.104]. Turhan `render`-funktion käytön välttämiseksi on suositeltavaa käyttää staattisia attribuutteja aina, kun on mahdollista [18, s. 104], mutta esimerkiksi lomakekomponenttien tilassa on hyvä säilyttää syötekenttien arvoja. Tämä helpottaa esimerkiksi syötteiden validoinnissa, jos tekstikentällä on jokin minimi- tai maksimipituus. (Esimerkkikoodi 3; kuva 2.)

```
// Luokkakomponentti jossa on 'greeting'-niminen tila-arvo
class FormComponent extends Component {
  constructor() {
    super();
    this.state = {
      greeting: ''
    }
  }
  handleInputChange({target}) {
    this.setState({greeting: target.value})
  }
  render() {
    let error = '';
    if(this.state.greeting.length < 11 ) error = <p>Input too short</p>;
    return (
      <div id="form-container">
        <form>
```

```
          <h3>Input: {this.state.greeting}</h3>
          <input type="text" onChange={this.handleInputChange.bind(this)} />
          {error}
        </form>
      </div>
    );
  }
}
```

Esimerkkikoodi 3. Reactilla tehty lomakekomponentti, jossa on yksi tekstikenttä ja tekstikentän yläpuolella on otsikko, joka päivittyy samalla, kun tekstikenttään kirjoittaa. Tekstikentällä on myös yksinkertainen validointi, joka tarkastaa, että teksti on vähintään 11 merkkiä pitkä.



Kuva 2. Esimerkkikoodissa 3 tehty lomake.

4.3 JavaScriptin nykyaikaistaminen ECMAScript 6+ -kielimäärittelyillä

ES on lyhenne ECMAScriptistä, joka on ECMA Internationalin standardoima skriptauskieli ja myös JavaScriptin perusta. ECMA International on järjestö, joka standardoi erilaisia tietotekniikkaan liittyviä asioita. Kaikki sen standardimääritelmät numeroidaan, kuten esimerkiksi ECMA-262, jota kutsutaan yleisesti nimellä ECMAScript. Se määrää käytännössä kaikki säännöt, yksityiskohdat ja suuntaviivat, joita kaikki skriptauskielet, kuten JavaScript, noudattavat [19]. ECMAScriptiä käytetään usein synonyyminä JavaScriptille, koska se on tunnetuin kieli, joka noudattaa sen määritelmiä.

ECMAScriptin 6. versio toi mukanaan suuria muutoksia ja paljon uusia ominaisuuksia, joita tässäkin projektissa käytetään. Tärkeitä uusia ominaisuuksia, joita tässä projektissa käytetään paljon ovat mm. nuolifunktiot, luokat, lupaukset ja uusi tapa luoda muuttujia. [20.]

Ennen ES6:ta JavaScriptillä ei virallisesti ollut luokkia kuten muilla olio-ohjelmointikielillä, mutta sille oli olemassa kirjastoja, kuten Classify tai JSClass. Nämä kirjastot mahdollistivat luokkien tapaisen toiminnallisuuden. ES6:n luokka on käytännössä vain funktio, joka yhdistelee JavaScriptin objekteja ja prototyyppejä niin, että koodista tulee helppolukuisempaa [21]. Tässä projektissa luokkia on pyritty käyttämään mahdollisimman paljon,

kuten esimerkiksi palvelinpuolella tietokantamalleissa ja kontrollereissa ja selainpuolella React-komponenteissa, D3-kuvaajissa ja liitteen 3 kielenkääntäjässä, koska luokkien käyttö tekee koodista siistimpää ja luettavampaa.

Liitteessä 3 on esimerkki `Translator`-luokasta, jossa on käytetty constructoria sekä staattista ja ei-staattista funktiota. Luokan `constructor()`-funktio ajetaan aina, kun luokasta luodaan uusi instanssi, kuten muissa olio-ohjelmointikielissä. Siellä on yleensä määritetty luokan omat ominaisuudet, kuten tässä tapauksessa oletuskieli, objekti, jossa käännökset ovat, ja muuttujien korvattava merkki. Staattiset funktiot, jotka merkitään avainsanalla `static`, ovat käytettävissä ilman instanssin luontia, esimerkiksi `Translator.getSupportedLocales()`. Ei-staattiset funktiot ovat käytettävissä vain, jos instanssi on luotu ensin: `const translator = new Translator()`, jonka jälkeen funktioon pääsee käsiksi kirjoittamalla `translator.changeLang('fi')`.

ES6:n uudet nuolifunktiot tekevät koodista tiiviimpää muuttamalla funktioiden luonnista `function`-avainsanan nuoleksi. Nykyään yksinkertaisen funktion voi luoda kirjoittamalla `(txt) => txt`, kun ennen samanlainen funktio luotiin kirjoittamalla `function(txt) { return txt }`. Jos nuolifunktion jälkeen ei merkitse aaltosulkeita, JavaScript ymmärtää, että suoraan nuolen jälkeinen koodi palautetaan, muuten vaaditaan aaltosulkeet ja `return`-avainsana. [20.]

Yksi ES6:n suurimpia muutoksia on muuttujien (variable) luominen. Ennen kaikki muuttujat luotiin avainsanalla `var`, nykyään kehittäjän tulee ajatella, onko kyseessä muuttumaton vai muuttuva muuttuja. Jos kehittäjä tietää, että muuttujan arvo tulee muuttumaan jossain vaiheessa, esimerkiksi silmukan iteraattori, hän käyttää `let`-avainsanaa, muuten käytetään `const`-avainsanaa. Käyttämällä `let`- ja `const`-avainsanoja koodista tulee myös luettavampaa, sillä lukija ymmärtää, että arvo on joko pysyvä tai muuttuva. ES6:ssa muuttujat ovat luettavissa vain oman blokkinsa (block) sisällä, eli jos luo muuttujan funktion sisällä, siihen ei pääse käsiksi funktion ulkopuolelta. Tämä helpottaa muuttujien nimeämisessä: esimerkiksi silmukoiden iteraattoreissa käytetään usein `i`-nimistä muuttujaa, mutta ennen kehittäjä ei voinut käyttää samaa muuttujan nimeä useassa paikassa. [20.]

Asynkronisen koodauksen helpottamiseksi ES6 tarjoaa lupaukset (Promise). Lupauksilla voidaan suorittaa koodia, jossa ei olla varmoja, koska data palautetaan, ja halutaan

jatkaa koodin suorittamista vasta, kun data on saatu, esim. tietokantapyynnöt. Ennen lupauksia vastaava koodi toteutettiin callback-funktioilla, jotka ovat ruman näköisiä ja joista tulee nopeasti epäluettavia, jos niitä on toistensa sisällä. Lupauksilla on kolme eri tilaa: odottava-, täytetty- ja hylätty-tila. Odottava tila on lupauksen oletustila, jossa lupausta ei ole vielä täytetty eikä hylätty. Täytetty tila tarkoittaa, että koodi on suoritettu onnistuneesti ilman virheitä. Jos lupausfunktion suorituksen yhteydessä ilmeni virheitä, lupauksen tila on hylätty. Täytetty lupaus suoritetaan `then()`-funktiossa, ja hylätty lupaus suoritetaan `catch()`-funktiossa, jotka ketjutetaan suoritettavan funktion perään (esimerkkikoodi 4). [22.]

```
function promiseExample(txt) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      if(txt.length < 3) return reject('Text must be over 3 characters');
      return resolve(txt);
    }, 1000);
  });
}

promiseExample('Hello World')
.then((res)=>{
  console.log(res); // Tulostaa Hello World konsoliin sekunnin viiveellä
}).catch((err)=>{
  console.error(err);
});
```

Esimerkkikoodi 4. Lupausfunktio, joka palauttaa hylätyn tai täytetyn lupauksen yhden sekunnin viiveellä.

Esimerkkikoodissa 4 oleva funktio tarkistaa, että sille annettu `txt`-parametri on vähintään kolme merkkiä pitkä. Jos annettu teksti on alle kolme merkkiä, funktio palauttaa hylätyn lupauksen `reject()`-funktiolla, muuten palautetaan täytetty lupaus `resolve()`-funktiolla. Funktiota suorittaessa parametrinä annetaan teksti ”Hello World”, ja koska teksti on yli kolme merkkiä pitkä, lupaus täyttyy, joten koodin suoritus jatkuu `then()`-funktion sisällä, jossa tulostetaan kyseinen teksti konsoliin. Jos teksti olisi ollut vain kaksi merkkiä, koodi olisi jatkunut `catch()`-funktion sisällä, jossa tulostetaan `reject()`-funktiossa määritelty teksti konsoliin virheilmoituksena.

ECMAScriptin uusimmassa versiossa, eli ES8:ssa, tuli parannuksia asynkronisiin funktioihin async- ja await-toimintojen kautta. Tässä projektissa async-funktioita käytetään paljon palvelinpuolella, kun haetaan dataa tietokannasta ja koodin seuraavat rivit ovat riippuvaisia tietokannasta saadusta datasta. Kun async-funktiota suoritetaan, se palauttaa lupauksen. Kun se palauttaa jonkin arvon, lupaus täyttyy, tai jos se heittää virheen,

lupaus hylätään. Käytännössä async-funktioissa olevat await-toiminnot pysäyttävät koodin suorituksen kunnes suoritettava funktio palauttaa arvon tai heittää virheen (esimerkkikoodi 5). [23.]

```
async function asyncExample() {
  const hello = await promiseExample('Hello');
  const world = await promiseExample('World');
  console.log(`${hello} ${world}`);
}
asyncExample(); // Tulostaa Hello World 2 sekunnin viiveellä
```

Esimerkkikoodi 5. Esimerkki async-funktiosta ja await-toiminnoista

Esimerkkikoodissa 5 on async-funktio, jossa suoritetaan esimerkkikoodissa 4 luotua `promiseExample()`-funktiota, joka palauttaa sille annetun tekstin tai heittää virheen sekunnin viivellä. Esimerkissä asetetaan sanat ”Hello” ja ”World” omiin muuttujiin käyttämällä await-toimintoa. `hello`- ja `world`-muuttujat saavat arvonsa sekunnin viiveellä toisistaan, eli funktiossa oleva konsoliin tulostus suoritetaan kahden sekunnin jälkeen, kun molemmat arvot on asetettu. Jos saman koodin suorittaa ilman await-toimintoa JavaScript yrittää tulostaa pelkät odottavassa tilassa olevat lupausoliot konsoliin heti funktiota kutsuessa. Esimerkissä käytetään myös ES6:ssa tullutta uutta tapaa sisällyttää JavaScript-koodia tekstin sisään, kunhan teksti on ``-heittomerkkien sisällä ja ajettava koodi suoritetaan `${}`-merkkien sisällä.

Koska vanhemmat selaimet eivät tue ES6:n syntaksia ja uusia ominaisuuksia, koodi tulee koota ES5:n syntaksiin. Yksi JavaScript-liitännäinen, joka ratkaisee tämän ongelman, on Babel. Se osaa muuntaa kaikki ES6:ssa ja sen jälkeen tulleet ominaisuudet ES5:n syntaksiin menettämättä mitään toiminnallisuuksia (esimerkkikoodi 6). [24.]

```
// ES6
function greet(greeting = 'Hello') {
     return greeting;
}
let arr = [1,2,4,6,8].filter((num)=>num>4);
const plus = `10 + 10 = ${10+10}`;

/* Babelilla koottu ES5 syntaksiin
'use strict';

function def() {
     var greeting = arguments.length > 0 && arguments[0] !== undefined ?
                    arguments[0] : 'Hello';

     return greeting;
}
var arr = [1, 2, 4, 6, 8].filter(function (num) {
     return num > 4;
});
var plus = '10 + 10 = ' + (10 + 10);
```

```
*/
```

Esimerkkikoodi 6. Esimerkkejä oletusarvoisesta funktiosta, nuolifunktiosta, const/let-muuttujista ja tekstin sekaan upotetusta koodista. Esimerkissä alkuperäinen ES6- ja Babelilla koottu ES5-koodi.

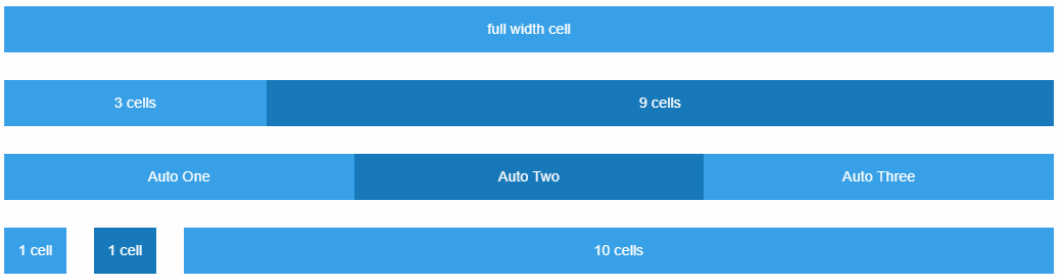
4.4 Käyttöliittymän tyylin yhtenäistäminen Foundation 6 -kehyksellä

Foundation on Zurbin kehittämä selainpuolen kehys, joka tarjoaa kehittäjälle valmiita HTML-komponentteja ja CSS-tyylejä. Se vähentää kehittäjän työtä huomattavasti tarjoamalla valmiit tyylit eri komponenteille, kuten napeille, tekstikentille ja taulukoille. Foundation tarjoaa myös uusia komponentteja, joita tässäkin projektissa käytetään, esimerkiksi modaalit, navigaatiopalkin ja accordion-komponentin. Foundationin komponentit on tehty responsiivisiksi, eli komponentit reagoivat käyttäjän syötteeseen. Tämän huomaa esimerkiksi pienentämällä selainikkunan kokoa: komponentit pienenevät tai vaihtavat paikkaa ikkunan pienentyessä. [25.]

Foundationin käyttöliittymät rakennetaan käyttäen ruudukkomallia (grid). Ruudukkomallissa yksi ruudukko on aina enintään 12 saraketta (column) tai solua (cell) leveä, joka helpottaa käyttöliittymän ulkoasun yhtenäisyyden pitämisessä. Foundation 6 tuo mukanaan XY-ruudukon, joka tulee korvaamaan vanhan ruudukkomallin. Vanha ruudukko on toteutettu käyttäen CSS:n ”float”-arvoja, ja se toimii edelleen, mutta on rajoitettu vain vaakatasoisiin riveihin. Uusi ruudukko on toteutettu modernimmalla flexboxilla, jonka yhteydessä sarakkeet muuttuivat soluiksi. Flexbox-ruudukko mahdollistaa uusia toimintoja, kuten solujen automaattisen leveyden ja pystysuuntaisen ruudukon (esimerkkikoodi 7; kuva 3). [26.]

```
<div class="grid-x">
  <div class="cell">full width cell</div>
</div>
<div class="grid-x">
  <div class="small-3 cell">3 cells</div>
  <div class="small-9 cell">9 cells</div>
</div>
<div class="grid-x">
  <div class="cell auto">Auto One</div>
  <div class="cell auto">Auto Two</div>
  <div class="cell auto">Auto Three</div>
</div>
<div class="grid-x grid-margin-x">
  <div class="small-1 cell">1 cell</div>
  <div class="small-1 cell">1 cell</div>
  <div class="small-10 cell">10 cells</div>
</div>
```

Esimerkkikoodi 7. Erilaisia Foundation 6:n XY-ruudukkoesimerkkejä.



Kuva 3. Esimerkkikoodissa 6 luodut ruudukkoesimerkit.

Ruudukkomallissa solujen koot annetaan HTML-elementeille CSS-luokkina. Esimerkiksi solu, jolla on luokka `small-2`, vie kaikenkokoisilla ruuduilla ruudukosta kuudesosan. Solu, jolla on luokat `small-6` ja `medium-3`, vie pienellä näytöllä puolet ruudukon leveydestä, ja keskisuurella näytöllä solu vie vain neljänneksen. Koska Foundation on ns. ”Mobiili ensin” -kehys, solut joilla on vain `small`-alkuinen kokoluokka, määrittävät koon kaikenkokoisille ruuduille. Jos solulla on vain esim. `large-6`-luokka, Foundationin mukaan solu täyttää vain suurilla näytöillä puolet ruudukosta ja sitä pienemmillä ruuduilla solu vie oletusarvoisesti täyden ruudukon leveyden.

Foundation on yksi suosituimmista selainpuolen kehyksistä ja sitä käyttävät mm. Disney, HP, eBay, Adobe ja Amazon. [27.]

4.5 Tyylitiedostojen parantaminen SCSS-esiprosessorilla

CSS:n esiprosessori on ohjelma, joka generoi validia CSS:ää käyttäen omaa syntaksiaan. Esiprosessorit yleensä tarjoavat ominaisuuksia, joita pelkkä CSS ei sisällä. Näillä ominaisuuksilla tyylitiedostoista tulee helpommin luettavia ja ylläpidettäviä. Suosittuja esiprosessoreita ovat Sass, SCSS, LESS ja Stylus. CSS:n esiprosessoreita varten palvelimella täytyy olla asennettuna jokin koodin kääntäjä, sen mukaan, mitä esiprosessoria käyttää. Kääntäjä muuntaa käytetyn syntaksin normaaliksi CSS:ksi. [27.]

Tässä projektissa käytetty esiprosessori on SCSS. Se on Sassin uudempi versio, jonka syntaksi muistuttaa enemmän CSS:n omaa syntaksia. Sen syntaksi eroaa Sassista siten, että Sass ei käytä aaltosulkeita eikä puolipisteitä. SCSS:llä on ominaisuuksia, joita

CSS:llä ei ole, kuten muuttujat, mixinit, pesintä (nesting) ja värien ja muiden arvojen käsittelyyn tarkoitettuja funktioita (esimerkkikoodi 8). [28.]

```
$default-color: #020111;
$default-height: 40px;

@mixin box-shadow($top, $left, $blur, $spread, $color) {
  -webkit-box-shadow: $top $left $blur $spread $color;
  -moz-box-shadow: $top $left $blur $spread $color;
  box-shadow: $top $left $blur $spread $color;
}

#test {
  background: $default-color;
  .container {
    min-height: $default-height;
    @include box-shadow(2px, 2px, 5px, 0px, black);
    p {
      font-size: 15px;
    }
  }
  .large-container {
    background: lighten($default-color, 10%);
    min-height: $default-height * 2;
    @include box-shadow(10px, 10px, 10px, 0, black);
  }
}
```

Esimerkkikoodi 8. Esimerkkejä eri SCSS-toiminnoista.

Esimerkkikoodissa 8 luodaan ensin kaksi muuttujaa nimeltään: ”default-color” ja ”default-height”. SCSS-muuttujat luodaan kirjoittamalla dollarimerkki, jonka perään kirjoitetaan muuttujan nimi ja kaksoispisteen jälkeen muuttujan arvo.

Muuttujien jälkeen esimerkissä luodaan SCSS mixin, jonka tarkoitus on vähentää toistettavaa koodia. Tässä esimerkissä luodaan oma mixin `box-shadow`-säännölle, joka yleensä vie kolme riviä, jos yhteensopivuuslisät ovat mukana. Mixineissä voi myös käyttää `each`-silmukoita ja `if / else`-operaattoreita dynaamisuuden lisäämiseen.

Mixinin luonnin jälkeen näkyy, miten SCSS:n pesintä toimii. Kaikki säännöt ovat `#test`-elementin sisällä, jolle on annettu taustaväriksi `default-color`-muuttujan arvo. Säännöt, jotka on luotu `#test`-elementin sisällä, eivät päde sen ulkopuolisiin elementteihin, eli jos sovelluksessa käyttää `.container`-luokkaa `#test`-elementin ulkopuolella, esimerkin säännöt eivät päde. Mixinejä kutsutaan `@include`-avainsanalla ja mixinin nimellä. Kaikki mahdolliset arvot annetaan parametreinä sulkujen sisään.

Esimerkin `.large-container`-luokassa käytetään `lighten()`-funktiota, jolle annetaan väri ja prosenttiarvo värin vaalentamiselle. Saman luokan sisällä on myös esimerkki, miten laskuoperaattorit toimivat, kuten tässä tapauksessa kerrotaan `$default-height`-muuttujan arvo kahdella.

Koska selaimet eivät pysty käsittelemään SCSS:n ominaisuuksia, se täytyy kääntää CSS:ksi (esimerkkikoodi 9). SCSS:n ja Sassin kääntämistä varten tarvitaan jokin kääntäjäohjelma, kuten Compass.

```
#test {
  background: #020111;
}
#test .container {
  min-height: 40px;
  -webkit-box-shadow: 2px 2px 5px 0px black;
  -moz-box-shadow: 2px 2px 5px 0px black;
  box-shadow: 2px 2px 5px 0px black;
}
#test .container p {
  font-size: 15px;
}
#test .large-container {
  background: #080441;
  min-height: 80px;
  -webkit-box-shadow: 10px 10px 10px 0 black;
  -moz-box-shadow: 10px 10px 10px 0 black;
  box-shadow: 10px 10px 10px 0 black;
}
```

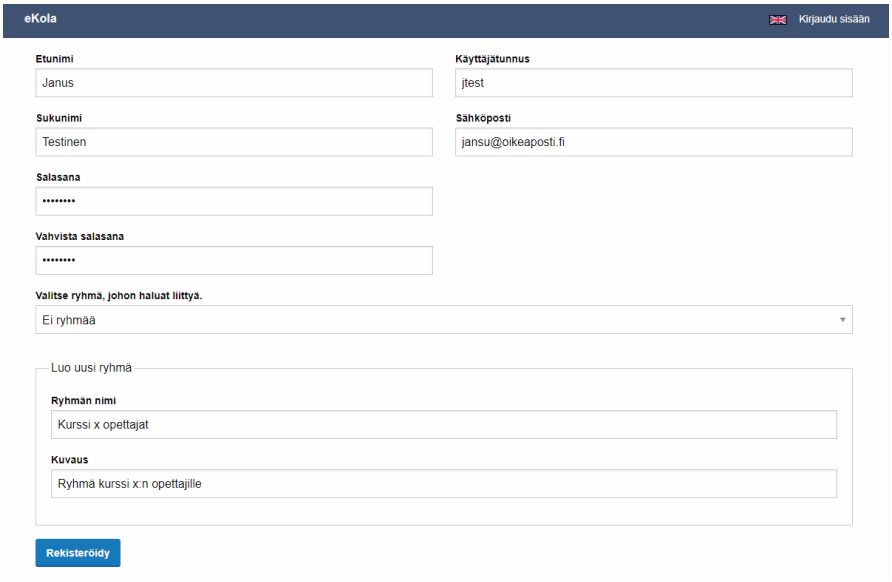
Esimerkkikoodi 9. Esimerkkikoodin 8 SCSS-koodi käännetty CSS:ksi.

5 Insinöörityössä tehdyn järjestelmän esimerkkikäyttötapaus

Tässä luvussa käydään läpi insinöörityössä tehdyn järjestelmän eri ominaisuudet ja se, miten käyttäjän tulisi niitä käyttää. Tässä esimerkkikäyttötapauksessa käyttäjä on Kurssin X opettaja. Hän luo opiskelijoilleen kyselyn, jossa heitä pyydetään arvioimaan kurssitovereiden projektit ja itsearvioimaan oma projekti. Kyselyn päätyttyä opettaja tarkastelee, miten opiskelijat arvioivat toistensa projekteja, ja päättää tulosten perusteella opiskelijoilleen loppuarvosanan. Liitteessä 2 on käyttötapauksesta tehty UML-kaavio, josta näkee prosessin yksinkertaisemmin.

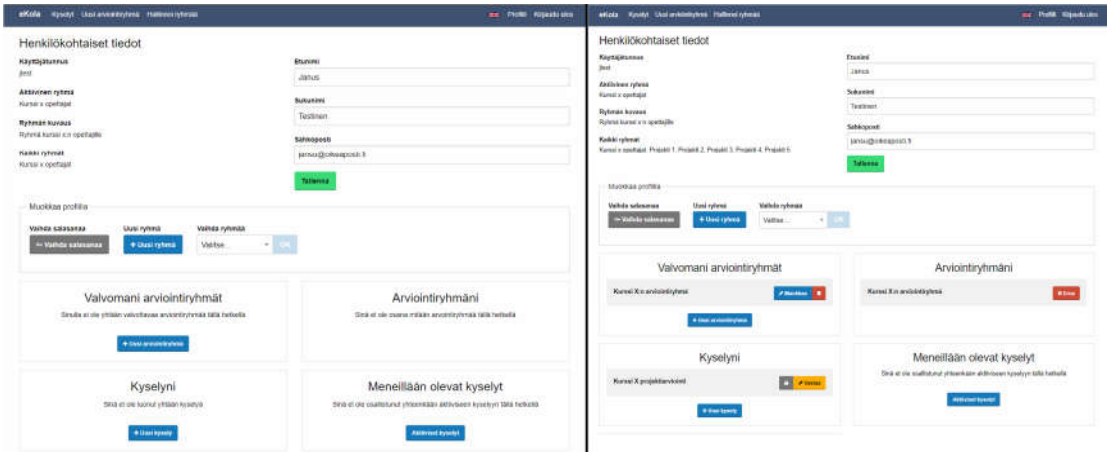
5.1 Profiilin luonti ja muokkaus

Jos käyttäjällä on aikomus luoda kyselyitä tai tarkastella kyselyiden tuloksia, hänen tulee rekisteröityä järjestelmään (kuva 4). Järjestelmään pääsee rekisteröitymään kirjautumissivun kautta painamalla rekisteröintilinkkiä. Uutta profiilia luodessa käyttäjän täytyy antaa oma etu- ja sukunimi, käyttäjätunnus, jota ei ole vielä käytössä, ja sähköpostiosoite. Sähköpostiosoitteen tulee olla oikea, sillä jos salasana on unohtunut, tähän osoitteeseen lähetetään salasanan vaihtolinkki. Kun kaikki perustiedot on annettu, täytyy vielä valita ryhmä, johon käyttäjä kuuluu. Ryhmän tarkoitus on kyselyn tuloksia tarkastellessa erottaa vastaukset toisistaan, jotta ryhmien vastauksia on helpompi verrata. Jos ryhmää ei löydy sivun pudotusvalikosta, sitä ei ole vielä luotu. Uuden ryhmän voi luoda kirjoittamalla ryhmälle halutun nimen oikeaan kenttään, ja halutessaan ryhmälle voi antaa kuvauksen.



Kuva 4. Järjestelmän rekisteröintisivu esimerkkitiedoilla täytettynä.

Käyttäjä ohjataan kirjautumisen jälkeen omalle profiilisivulle (kuva 5), jossa hän voi muokata henkilötietojaan, vaihtaa salasanaa tai vaihtaa aktiivista ryhmää. Profiilisivulta löytyy linkkejä uuden arviointiryhmän tai kyselyn luontiin. Kun käyttäjä on luonut vähintään yhden kyselyn tai arviointiryhmän, ne ilmestyvät käyttäjän profiilisivulle, kuten kuvan 5 oikealla puolella näkyy.



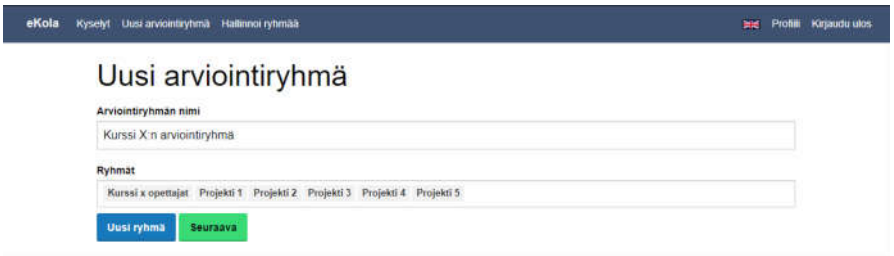
Kuva 5. Järjestelmän profiilisivu. Vasemmalla on uuden käyttäjän profiili, oikealla on profiili, jolla on luotu kysely ja arviointiryhmä.

5.2 Arviointi- ja käyttäjäryhmän luonti ja hallinnointi

Tässä käyttötapauksessa käyttäjä luo jokaiselle kurssin projektiryhmälle oman ryhmän järjestelmään. On myös mahdollista, että jokainen kurssin opiskelija luo itselleen profiilin ja liittää sen omaan ryhmään rekisteröitymisvaiheessa, mutta tämä ei kuitenkaan ole pakollista. Ryhmän voi luoda useassa eri näkymässä, kuten profiilisivulla tai uutta arviointiryhmää luodessa. Opettajakäyttäjä klikkaa uusi ryhmä -nappia ja syöttää ryhmän nimen ja halutessaan kuvauksen ryhmälle klikkauksen yhteydessä avattuun ikkunaan (kuva 6).

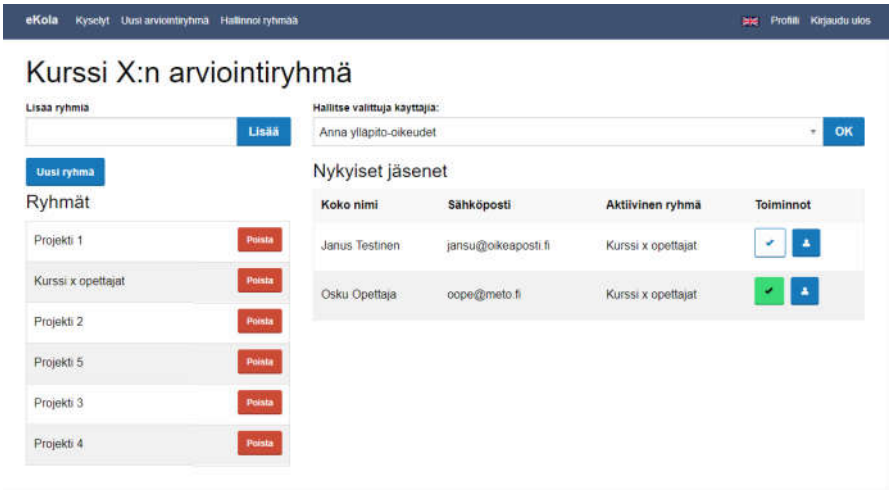
Kuva 6. Ryhmän luonti -ikkuna esimerkkitiedoilla täytettynä.

Ryhmien luonnin jälkeen opettaja luo arviointiryhmän, johon hän liittää kaikki projektiryhmät, jotka kuuluvat tämän esimerkin kurssiin (kuva 7). Arviointiryhmää luodessa tulee syöttää arviointiryhmälle nimi ja käyttäjäryhmät, joiden vastauksia halutaan lopuksi verrata keskenään. Ryhmät-kenttään voi syöttää useamman ryhmän nimen samaan aikaan, ja kenttä ehdottaa ryhmiä samalla, kun käyttäjä kirjoittaa.



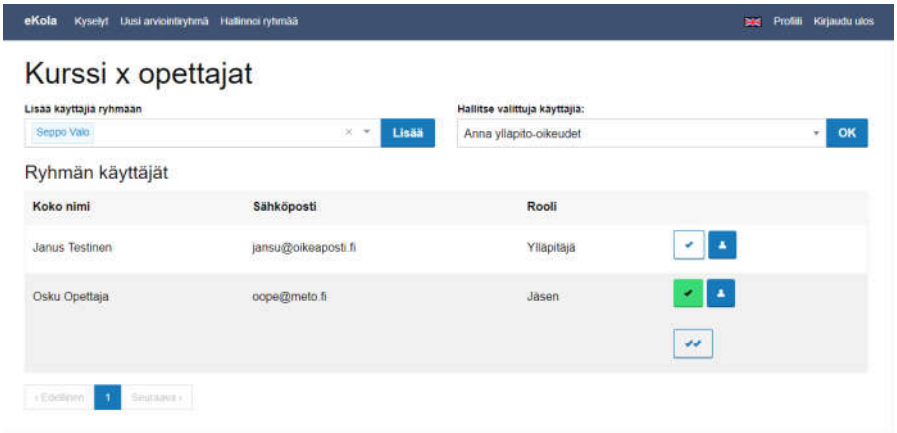
Kuva 7. Arviointiryhmän luontisivu, jossa on syötetty ryhmän nimi ja käyttäjäryhmät.

Kun arviointiryhmä on luotu, käyttäjä ohjataan arviointiryhmän hallinnointisivulle (kuva 8), jossa hän voi muokata arviointiryhmän käyttäjien oikeuksia, poistaa ja lisätä ryhmiä, luoda uusia ryhmiä ja sulkea käyttäjien vastausistuntoja. Tässä esimerkissä annetaan toiselle käyttäjälle oikeudet hallita arviointiryhmää. Käyttäjä valitsee halutut käyttäjät klikkaamalla tarkistusmerkkiä käyttäjän rivillä niin, että se muuttuu vihreäksi. Kun halutut käyttäjät on valittu, käyttäjä valitsee toimenpiteeksi ylläpito-oikeuksien antamisen ja painaa OK-nappia.



Kuva 8. Arviointiryhmän hallinnointisivu. Kuvassa näkyy käyttäjä Osku Opettaja valittuna ja toteutettavana toimenpiteenä on valittu ylläpito-oikeuksien antaminen.

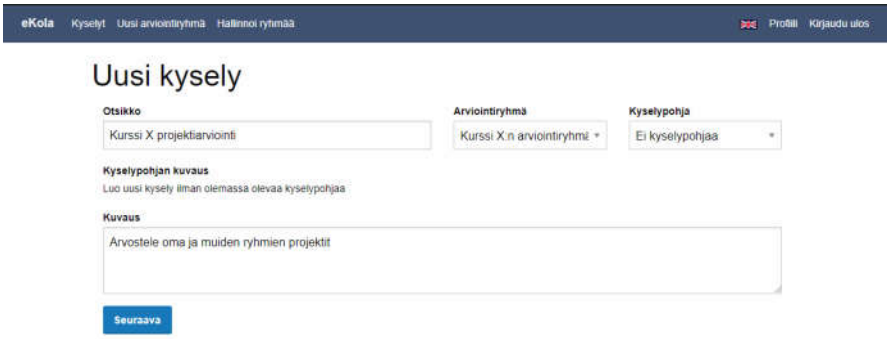
Käyttäjä voi myös hallita omaa aktiivista käyttäjäryhmää menemällä navigaatiopalkista ryhmän hallinnointisivulle (kuva 9). Ryhmän hallinnointisivulla käyttäjä voi lisätä ja poistaa käyttäjiä ryhmästä, antaa käyttäjille ylläpito-oikeudet ryhmään ja lisätä ja poistaa ryhmän käyttäjiä arviointiryhmiin. Tässä esimerkissä annetaan toiselle opettajalle ylläpito-oikeudet ryhmään, jotta hänkin voi lisätä tai poistaa käyttäjiä. Hallittavat käyttäjät tulee valita klikkaamalla tarkistusmerkkiä käyttäjän rivillä niin, että se muuttuu vihreäksi. Kun halutut käyttäjät on valittu, käyttäjä voi valita pudotusvalikosta suoritettavan toimenpiteen, joka tässä tapauksessa on ylläpito-oikeuksien antaminen. Ryhmän hallintasivu on hieman erilainen riippuen käyttäjän oikeuksista, eli jos käyttäjällä ei ole ylläpito-oikeuksia ryhmään, hän voi ainoastaan tarkastella ryhmän jäsenten käyttäjätietoja.



Kuva 9. Ryhmän hallinnointisivu. Kuvassa on valittu käyttäjä Osku Opettaja ja toimenpiteeksi on valittu ylläpito-oikeuksien antaminen. Lisäksi ryhmään ollaan lisäämässä käyttäjä nimeltään Seppo Valo.

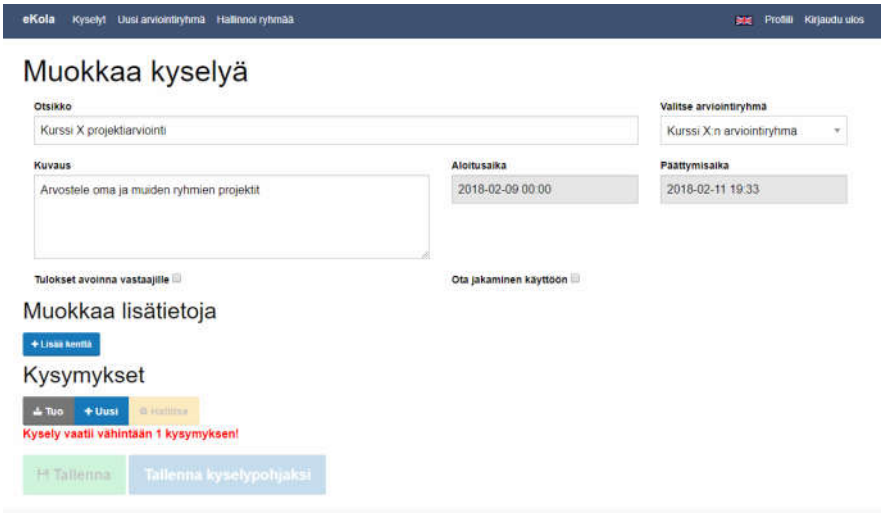
5.3 Kyselyn luonti ja jakaminen

Kun käyttäjällä on arviointiryhmä ja siihen on liitetty halutut ryhmät, hän voi luoda uuden kyselyn. Uuden kyselyn luontisivulla (kuva 10) käyttäjän täytyy antaa kyselylle nimi, arviointiryhmä ja vapaaehtoisesti kuvaus kyselylle. Halutessaan käyttäjä voi käyttää kyselypohjaa, joka luo uuden kyselyn etukäteen tehdyillä kysymyksillä. Kuka tahansa voi luoda kyselypohjia kyselyn muokkaussivulla. Kun tiedot on annettu, voi siirtyä seuraavalle sivulle.



Kuva 10. Kyselyn luontisivu. Sivulla on annettu kyselylle nimi, arviointiryhmä ja kuvaus. Kyselypohjaa ei käytetä.

Kun kysely on luotu, käyttäjä ohjataan juuri luodun kyselyn muokkaussivulle (kuva 11). Tällä sivulla kyselylle luodaan kysymykset, muokataan aloitus- ja päättymisaikoja ja luodaan linkit, joilla vastaajat pääsevät vastaamaan kyselyyn ilman kirjautumista. Tässä esimerkissä luodaan kysely, jossa on sama kysymys jokaiselle kurssin projektille. Kysymyksissä pyydetään opiskelijaa arvioimaan kunkin ryhmän projektin tekninen toteutus ja projektin esitys asteikolla 0–5. Lisäksi kyselyssä kysytään opiskelijan nimeä ja opiskelijanumeroa lisätietokysymyksillä.

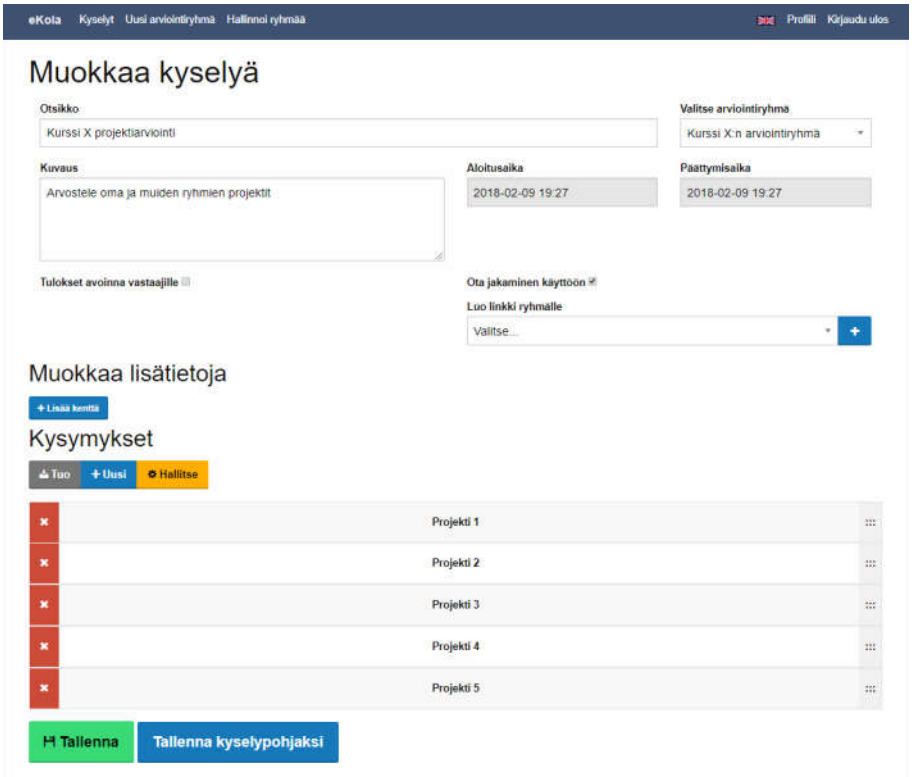


Kuva 11. Uuden kyselyn muokkaussivu. Kyselyllä on tässä vaiheessa vain perustiedot.

Lisätietokysymyksiä voi lisätä klikkaamalla lisätietojen muokkauskohdan alla olevaa nappia. Lisätietokentässä kyselyn luoja voi kysyä vastaajalta mitä haluaa, kuten ikää, nimeä, sukupuolta jne.

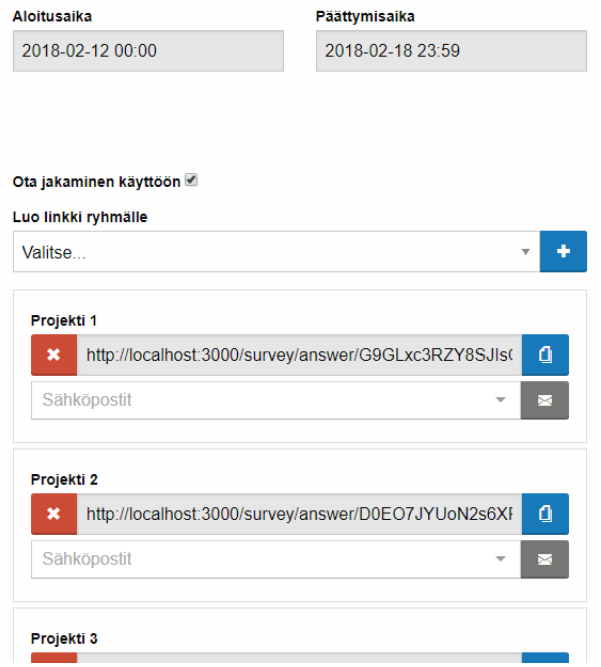
Seuraavaksi kyselyn luoja voi lisätä uuden kysymyksen klikkaamalla kysymyskohdan alla olevaa *Uusi*-nappia. Sitä klikkaamalla sivulle ilmestyy uusi kysymyslaatikko, jossa kysytään kysymykselle otsikkoa, kuvausta ja kriteereitä. Otsikko ja vähintään yksi kriteeri ovat pakollisia tietoja. Kriteereitä lisätään kysymyksiin klikkaamalla *Lisää*-nappia kysymyslaatikon sisällä. Tämä lisää kysymyslaatikon kaltaisen laatikon kysymyksen sisälle. Kriteerilaatikossa kysytään otsikkoa, kuvausta, kriteerin tyyppiä ja tyypin asetuksia. Kriteereillä on kahta erilaista tyyppiä: asteikko ja monivalinta. Monivalintakriteerissä voi luoda useita eri vaihtoehtoja, joista vastaaja voi valita vain yhden. Asteikkokriteerissä luoja määrittää minimi- ja maksimiarvon, joiden väliltä vastaaja valitsee sopivan arvon.

Tässä esimerkissä luodaan jokaiselle kurssin projektille yksi kysymys ja jokaiseen kysymykseen luodaan kaksi asteikkokriteeriä: tekninen toteutus ja esitys, joiden minimiarvot ovat 0 ja maksimiarvot ovat 5. (Kuva 12.)



Kuva 12. Valmis kysely. Jokaiselle projektille on luotu oma kysymys.

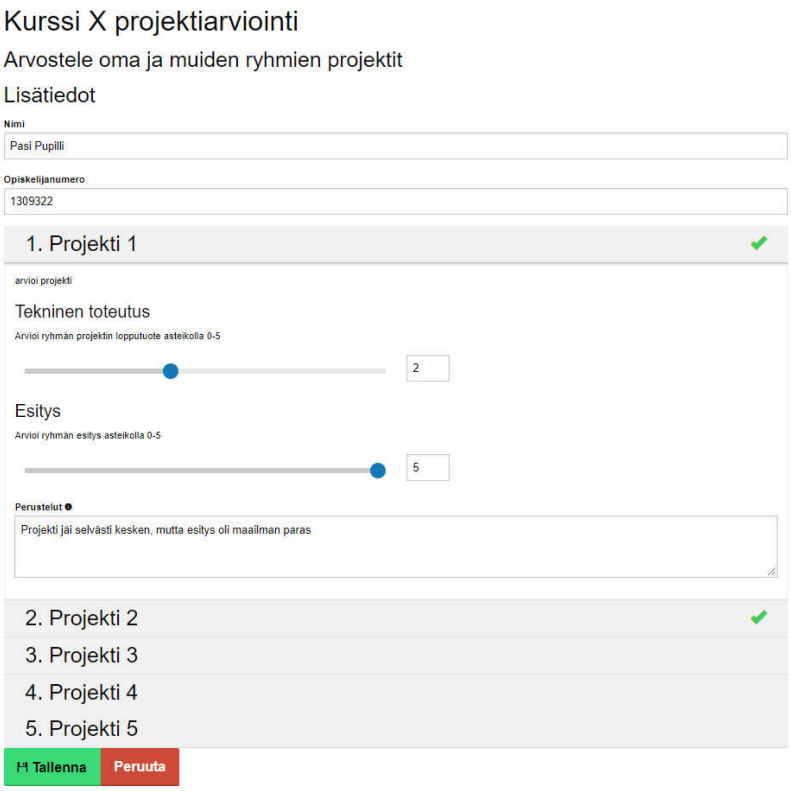
Kun kysymykset on luotu ja kysely pitää avata ja jakaa vastaajille, ensin asetetaan kyselylle aloitus ja päättymisaika (kuva 13). Vastaajat eivät pääse kyselyyn, jos aloitusaika on tulevaisuudessa tai jos päättymisaika on menneisyydessä. Tässä esimerkissä vastaajille annetaan viikko aikaa vastata kyselyyn. Aloitus- ja päättymisaikakenttiä klikatessa aukeaa kalenteri, josta valitaan päivämäärä ja kellonaika, jolloin kysely alkaa tai päättyy. Kun ajat on asetettu, kysely voidaan jakaa vastaajille. Kyselyn voi jakaa linkeillä, kun ensin painaa *Ota jakaminen käyttöön* -valintaruutua. Sen alapuolelle ilmestyy pudotusvalikko, jossa on listattuna kyselylle valitun arviointiryhmän eri käyttäjäryhmät. Jokaiselle käyttäjäryhmälle voi luoda oman linkin, jotta vastaukset erottuvat tuloksia tarkastellessa. Linkki luodaan valitsemalla ryhmä pudotusvalikosta ja painamalla plusmerkillistä nappia valikon vieressä. Linkin voi lähettää sähköpostitse suoraan tältä sivulta syöttämällä kaikkien vastaajien sähköpostiosoitteet sähköpostikenttään. Kun kaikki on valmista, kysely tulee tallentaa klikkaamalla tallennusnappia sivun alareunassa. Kyselyn voi tallentaa keskeneräisenäkin, mutta ei jos kyselystä löytyy jotain suurempia virheitä, kuten samoja kysymyksiä useaan kertaan tai jos kyselyllä ei ole lainkaan kysymyksiä.



Kuva 13. Kyselylle on asetettu viikon verran vastausaikaa, ja kyselyyn on luotu jaettavat linkit vastaajille.

5.4 Kyselyyn vastaaminen

Kun kyselyn linkit on jaettu ja aloituspäivä on menneisyydessä, vastaajat pääsevät vastaussivulle (kuva 14). Vastaamissivulla näkyvät kyselyn otsikko, kuvaus, lisätietokysymykset ja varsinaiset kysymykset. Sivu tallentuu automaattisesti 30 sekunnin välein, jos lomakkeessa on tapahtunut muutoksia. Jos kyselyyn on tullut kirjautumatta linkin kautta, painamalla peruuta-nappia vastaus poistetaan. Kun kysymyksen pakollisiin kohtiin on vastattu, kysymyksen otsikon viereen ilmestyy vihreä tarkistusmerkki ilmaisemaan, että kysymykseen on vastattu. Kysymyksiä voi laajentaa ja pienentää klikkaamalla niiden otsikoita. Tässä esimerkissä opiskelijat arvostelevat kaikki projektit, mukaan lukien omansa. Jokaiseen kysymykseen voi myös jättää perustelun, jossa kerrotaan, miksi tuli valittua annetut arvot. Kun kaikkiin kysymyksiin on vastattu, voi lopuksi painaa tallennusnappia ja poistua sivulta. Kysymykset ovat tallessa selaimen istunnossa, joten jos käyttäjä ei ole sulkenut selainta, hän pääsee linkin kautta vielä muokkaamaan vastauksiaan poistumisen jälkeen.



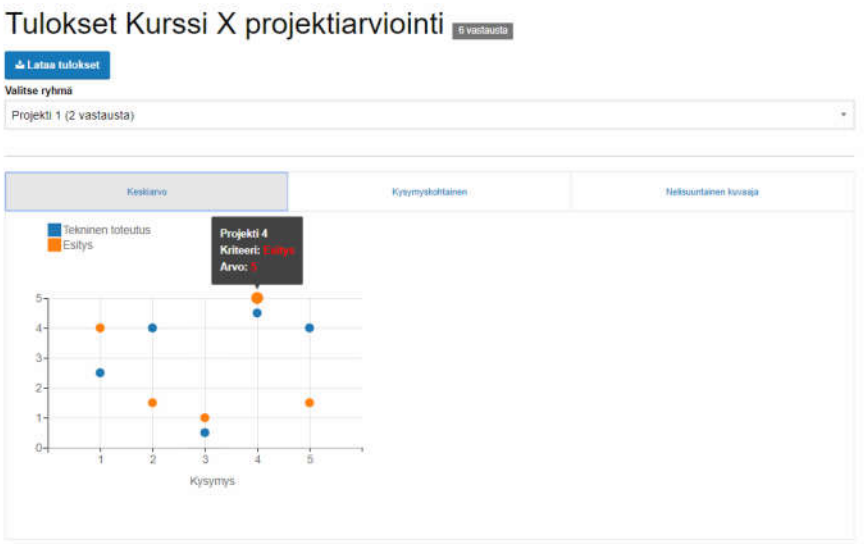
Kuva 14. Kyselyn vastaamissivu. Kuvassa opiskelija on antanut lisätiedot ja vastannut kahteen kysymykseen.

5.5 Kyselyn tulosten tarkastelu

Kun kyselyn aika on mennyt umpeen tai kyselyn luoja on manuaalisesti sulkenut kyselyn, hän voi tarkastella kyselyn tuloksia. Tulossivulle pääsee joko omasta profiilista tai sulkeutuneiden kyselyiden listasta. Tulossivulla voi verrata eri ryhmien vastauksia ja tehdä erilaisia johtopäätöksiä kuvaajien perusteella. Sivulta saa myös ladattua tulokset Excel-taulukkona, jos haluaa tarkastella yksittäisiä vastauksia. Kyselyn lisätietokysymykset ovat näkyvillä vain ladatuissa taulukkotiedostoissa, sillä tulossivulla ei voi tarkastella yksittäisiä vastauksia.

Tässä esimerkkitapauksessa opettaja aikoo käyttää opiskelijoiden arvioita auttamaan kurssiarvosanojen päättämisessä. Kuvassa 15 tarkastellaan Projekti 1 -ryhmän vastausten keskiarvoja, ja kuvaajasta näkee nopeasti, että ryhmän mielestä Projekti 4 -ryhmä oli kurssin paras. Viemällä hiiren yksittäisen pisteen päälle näkee tarkempia tietoja

kyseisestä pisteestä. Tarkasteltavaa ryhmää voi vaihtaa pudotusvalikosta kuvaajien yläpuolella. Kuvaajat päivittyvät automaattisesti, kun pudotusvalikon aktiivinen kohde vaihtuu.



Kuva 15. Tulossivu. Kuvassa tarkastellaan Projekti 1 -ryhmän vastauksia ja aktiivisena kuvaajana on keskiarvokuvaaja.

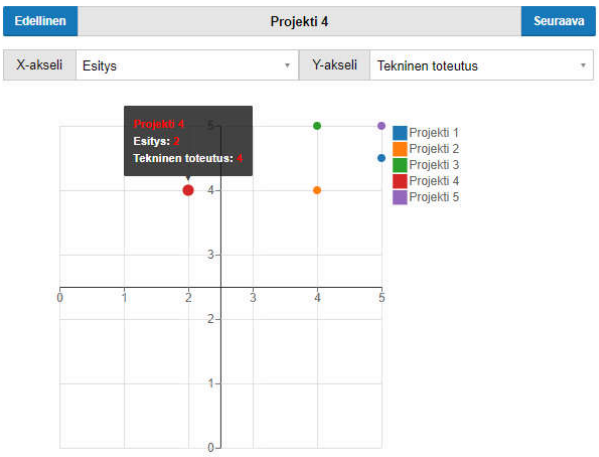
Kysymyskohtaisissa kuvaajissa (kuva 16) voi katsella tarkemmin, kuinka moni vastaaja antoi saman arvon tai kysymyksen keskiarvovastauksia. Kuvaajien alapuolella on näkyvillä myös vastaajien antamat perustelut kysymyskohtaisesti.

Kuva 16. Tulossivu. Valittuna kuvaajana on kysymyskohtaiset kuvaajat. Aktiivisena kysymyksenä on 4. kysymys. Ylempi kuvaaja näyttää, kuinka monta vastausta kullakin arvolla on. Alempi kuvaaja on yksinkertainen keskiarvokuvaaja.

Nelisuuntaisesta kuvaajasta (kuva 17) näkee kysymyskohtaisesti jokaisen ryhmän keskiarvovastauksen. Nelisuuntaisessa kuvaajassa voi vaihtaa y- ja x-akseleita haluamakseen, jos kriteereitä on vähintään kaksi. Jos kriteereitä on vain yksi, kuvaajaa ei piirretä.

Kuvassa 17 on valittuna Projekti 4 -ryhmän arviointikysymys, josta näkee mitä mieltä kaikki ryhmät ovat valitun ryhmän projektista. Toisin kuin kuvan 15 keskiarvokuvaajassa, kukin kuvaajan pisteistä viittaa eri ryhmään. Pisteiden värit on selitetty kuvaajan oikealla puolella.

Nelisuuntaisessa kuvaajassa voi nopeasti katsoa, miten hyvin kummatkin valitut kriteerit on arvioitu, katsomalla mihin lohkoon pisteet sijoittuvat. Esimerkiksi kuvan 17 kuvaajassa melkein kaikki pisteet ovat yläoikeassa lohkossa, mikä tarkoittaa sitä, että ryhmät ovat arvioineet Projekti 4 -ryhmän esityksen ja teknisen toteutuksen hyvin. Jos pisteet keskittyisivät alavasempaan lohkoon, voidaan päätellä, että projektin kriteerit on arvioitu huonosti.



Kuva 17. Tulossivu. Valittuna kuvaajana on nelisuuntainen kuvaaja. Kuvaajasta näkyy jokaisen ryhmän keskiarvovastaus kysymyskohtaisesti.

6 Kyselysovelluksen toteutus

6.1 Ensimmäisen version kehitys

Insinöörityössä kehitetyn järjestelmän suunnittelu alkoi vuoden 2017 alussa pidetyssä kokouksessa. Kokouksessa katsottiin vanhan julkaisemattoman järjestelmän kuntoa ja pohdittiin, onko mahdollista jatkaa järjestelmän kehitystä. Kokouksen jälkeen tutustuttiin enemmän vanhan järjestelmän koodiin ja tehtiin päätös, että olisi parempi tehdä järjestelmä kokonaan uusiksi, nykyaikaisemmalla teknologialla. Vanhan järjestelmä oli toteutettu PHP:llä vuonna 2009 ilman mitään ohjelmointikehystä, ja osa koodista oli ikänsä vuoksi toimimatonta.

Kun päätös järjestelmän uusimisesta oli tehty, pidettiin kokous, jossa päätettiin uuden järjestelmän minimivaatimukset ja luotiin projektisuunnitelma. Päätettiin, että järjestelmällä tulisi pystyä luomaan, muokkaamaan ja poistamaan kyselyitä, kysymyksiä, kysymyksen kriteereitä, organisaatioita ja arviointiryhmiä. Järjestelmän piti olla minimivaatimuksineen käytettävissä jo kesän jälkeen, eli kehitykselle annettiin aikaa reilu kolme kuukautta.

Alkuperäisessä projektisuunnitelmassa määriteltiin, että:

- kyselyä muokatessa käyttäjän tulisi pystyä muokkaamaan sen nimeä, kuvausta, vastausaikaa, organisaatiota ja kysymyksiä
- kysymystä muokatessa tulisi pystyä muokkaamaan sen otsikkoa, kuvausta, kategoriaa ja kriteereitä
- kriteeriä muokatessa tulisi pystyä muokkaamaan sen otsikkoa, kuvausta, tyyppiä ja tyypin asetuksia
- organisaatiota muokatessa tulisi pystyä muokkaamaan sen nimeä ja kuvausta
- arviointiryhmää muokatessa tulisi pystyä muokkaamaan sen nimeä, kuvausta, kyselyitä ja osallistuvia organisaatioita.

Lisäksi oli määritelty, että kyselyn tuloksista piirretään kuvaaja, josta voi verrata eri organisaatioiden vastauksia keskenään. Alkuperäisen suunnitelman mukaan järjestelmän tuli toimia vain ennalta määritettyjen 28 QAEMP-kysymyksen kanssa, joten kuvaaja oli suunniteltu toimimaan parhaiten sellaisten kysymysten kanssa, joilla on kolme kriteeriä, joissa kaikissa on yhtä monta vaihtoehtoa tai sama arvoasteikko.

Kun vaadittavat ominaisuudet olivat tiedossa, kehittäjäryhmän kesken päätettiin käytettävät teknologiat ja vastuualueet. Järjestelmää oli kehittämässä kolme henkilöä, ja vastuualueet jakautuivat niin, että yksi oli vastuussa palvelinpuolen koodista, yksi selainpuolen koodista ja yksi oli projektipäällikkö. Minun vastuualueeni oli selainpuolen koodi, mutta koodasin silti palvelinpuolen toimintoja, aina kun oli tarvetta.

Palvelinpuolen teknologiaksi päätettiin Sails.js, joka on Node-kehys, koska projektipäälliköllä oli aikaisempaa kokemusta siitä ja hän suositteli sen käyttöä. Tiedon tallentamiseen päätettiin, että tarvitaan molempia, SQL- ja NoSQL-tietokantoja. NoSQL-tietokantaa tarvitaan vain kysymysten vastausten tallentamiseen, koska kysymykset ovat mahdollisesti hyvin erilaisia rakenteellisesti. Selainnäkymien renderöintiä varten valittiin EJS, koska minulla oli siitä aikaisempaa kokemusta ja näkymien luonti oli vastuullani. Selainpuolen koodi tehtiin JavaScriptillä ilman mitään kehystä, käyttäen ES6:n toimintoja. Kuvaajien piirtämiseksi valittiin D3, koska tämän projektin kyselyitä varten oli rakennettava omanlainen kuvaaja. Selainpuolen tyylikehykseksi valittiin Foundation, koska kaikilla meistä oli siitä aikaisempaa kokemusta. Ryhmän keskinäinen viestintä tapahtui Slackillä, ja työtehtävät merkittiin Trello-tauluun.

Projektin ensimmäiset viikot menivät opetellessa teknologiapinon käyttöä, koska osa teknologioista oli vielä tuntemattomia. Näiden viikkojen aikana luotiin alustavia näkymiä ja funktioita oleellisimmille järjestelmän toiminnoille, kuten

- kyselyn luonti, muokkaus, tarkastelu, poisto
- kyselyyn vastaaminen, vastauksen poisto
- kysymysten lisääminen kyselyyn
- kysymysten luonti, muokkaus ja poisto
- arviointiryhmien luonti ja poisto
- organisaatioiden luonti ja poisto.

Kun järjestelmä oli palvelinpuolella funktionaalisesti käytettävissä, siirryttiin kuvaajan rakentamiseen D3:lla. Ensimmäinen versio kuvaajasta valmistui noin viikossa, mutta sen parantelu ja ohjelmointivirheiden korjaus jatkui läpi koko projektin. Käyttöliittymää kehittäessä pidin tärkeänä, että järjestelmä toimii kaikenkokoisilla näytöillä, koska projektissa käytettiin Foundation-kehystä, joka on suunniteltu ”mobiili ensin” -menetelmälle. Selainpuolen toiminnallisuuksia kehittäessä halusin, että mahdollisimman moni toiminto toimii ilman sivun uudelleen latausta, koska uskon sen olevan käytettävyyden kannalta mukavampaa.

Suurin osa selainpuolen JavaScript-koodista luotiin käyttäen ES6:n luokkia, joka tekee koodista helppolukuisempaa ja helpommin ylläpidettävää. Luokkien käyttö myös estää kehittäjää vahingossa käyttämästä väärää funktiota väärässä paikassa. Selainpuolen JavaScript-tiedostot oli jaoteltu niin, että jokaisella näkymällä, joka tarvitsee funktionaalisuutta, on oma tiedosto.

Projektisuunnitelmassa oli ehdotettu, että pidettäisiin kokouksia n. kahden viikon välein, mutta todellisuudessa kokouksia oli kerran kuukaudessa. Kokouksissa esiteltiin järjestelmän senhetkinen tila ja otettiin vastaan palautetta ja keksittiin uusia toiminnallisuuksia. Kokouksissa tuli usein huomattua ohjelmointivirheitä ja käytettävyydeltään huonoja toteutuksia, jotka korjattiin aina kokouksien jälkeen. Kokouksissa pyydettiin joskus suuriakin muutoksia järjestelmän toimintoihin, kuten kyselyn jako kirjautumattomille käyttäjille linkillä. Koska jotkin uudet toiminnot tulivat melko myöhäisessä vaiheessa projektin kehityksessä, niiden toteutus ei aina ollut paras mahdollinen. Kokouksissa korostettiin, että kyselyn luonti ja siihen vastaaminen ovat järjestelmän tärkeimmät näkymät, joten niistä tulee tehdä niin helppokäyttöisiä kuin mahdollista.

Haasteellisin sivu kehittää selainpuolella oli kyselyn muokkaussivu, koska siinä on paljon eri toimintoja, kuten kysymyspankki, kyselyn tallentaminen, kyselypohjan tallentaminen ja kysymysten uudelleenjärjestäminen. Suurin haaste kuitenkin oli kyselylomakkeen tallennus tietokantaan. Koska kyselyyn voi lisätä kysymyksiä, joilla on erityyppisiä kriteereitä, lomaketta ei voi tallentaa sellaisenaan, vaan sitä varten kehitettiin oma komponentti, joka muuntaa lomakkeen ymmärrettäväksi JSONiksi. Tämän komponentin kehittämiseen meni melko paljon aikaa, koska siitä löytyi jatkuvasti pieniä ohjelmointivirheitä. Lisäksi komponentti vaati, että HTML-elementit nimetään tietyllä tavalla, mikä myös hankaloitti koodaamista.

Järjestelmä saatiin ajallaan käytettäväksi, ja luvatut vähimmäisvaatimukset olivat mukana. Vähimmäisvaatimuksien lisäksi tehtiin paljon ominaisuuksia, joita ei ollut määritelty projektisuunnitelmassa, kuten kyselyn jakaminen linkillä, omien kysymysten luonti ja vastauksista koottu Excel-tiedosto.

6.2 Teknologiapinon vaihto

Ensimmäisen version kehityksen aikana ryhmällä oli jatkuvasti ongelmia Sails-kehyksen kanssa, minkä vuoksi jouduttiin tekemään hieman epäselviä ratkaisuja ongelmien kiertämiseksi. Kun ajateltiin järjestelmän tulevaisuutta ja jatkokehitystä, nämä huonot ratkaisut kasautuisivat ja lopulta järjestelmän ylläpidosta tulisi erittäin vaikeaa. Ratkaisuksi päätettiin vaihtaa palvelinpuolen kehys Sailsista Koa-nimiseen kehykseen. Koa on tällä hetkellä yksi suosituimpia Node-kehyksiä, ja se käyttää ES8:n async-funktioita, joka tekee palvelinpuolen koodista helppolukuisempaa ja nykyaikaisempaa. Koan lisäksi päätettiin, että käännetään selainpuolen dynaamiset sivut, kuten kyselyn muokkaus-, kyselyiden listaus- ja ryhmien muokkaussivut Reactille. React valittiin sen suosion takia ja koska ratkaisu, jolla dynaamiset sivut luotiin aikaisemmin, ei tulisi enää toimimaan uudella teknologiapinolla.

Sillä aikaa kun projektipäällikkö käytti aikaa uuden teknologiapinon pohjan konfigurointiin ja optimointiin, opettelin käyttämään Reactia. Opetteluun käytin erilaisia verkko-ohjeita ja videoita, mutta parhaiten opin käyttämään sitä, kun päätin aloittaa järjestelmän monimutkaisimman näkymän eli kyselyn muokkaussivun kääntämisen. Kun uuden teknologiapinon pohja oli valmis, alettiin kääntää vanhan järjestelmän palvelinpuolen koodia uudelle järjestelmälle. Koodin kääntämisessä kesti noin kaksi viikkoa, koska alkuperäinen koodi oli hyvin suunniteltu ja osa koodista oli kopioitavissa pienillä muutoksilla. Selainpuolen staattiset näkymät, kuten kirjautumis-, rekisteröinti- ja vastaussivut pystyttiin ottamaan käyttöön suoraan ilman muutoksia.

React-sivujen kehityksessä kesti eniten aikaa, etenkin monimutkaisen kyselyn muokkaussivun kanssa. React-sivujen kehityksessä auttoi se, että alkuperäset JavaScript-koodit käyttivät ES6:n luokkia, koska React-komponentit saa myös rakennettua ES6-luokilla. Reactin käyttöönotto myös poisti paljon toistettavaa koodia. Reactin tilanhallinta mahdollisti lomakkeiden automaattisen validoinnin, eli esimerkiksi jos käyttäjä lisää kyselyyn samannimisen kysymyksen kahdesti, sivu heti ilmoittaa virheellisestä tiedosta. Ensimmäisessä versiossa virheentarkastus tapahtui pelkästään palvelinpuolella, eli käyttäjän täytyi ladata sivu uudestaan saadakseen ilmoituksen virheellisistä tiedoista. Tämä myös vähentää palvelimen rasitusta, koska sivun tallennusnappia ei voi painaa, jos lomakkeessa on virheitä. Reactin tilanhallinnan ansiosta ei tarvinnut enää käyttää ensimmäisen version komponenttia, joka muuntaa lomakkeen tietynlaiseksi JSONiksi,

vaan tiedot päivittyvät koko ajan React-komponentin tilaan, josta tiedot saadaan lähetettyä palvelimelle oikeanlaisena JSONina.

6.3 Uuden version kehitys

Koska järjestelmästä oli jo käytettävä versio olemassa, uuden version julkaisulla ei ollut yhtä kova kiire kuin ensimmäisellä. Uuden version kehityksessä oli tavoitteena saada järjestelmä toimimaan ja näyttämään käyttäjälle samalta kuin ensimmäinen versio, mutta kehittäjälle järjestelmä on helppolukuisempi ja paremmin dokumentoitu. Uusi versio saatiin samaan kuntoon ensimmäisen version kanssa melko nopeasti.

Uuden version kehityksen aikana pidettiin yhä säännöllisesti kokouksia, joissa aina löytyi korjattavaa, muokattavaa tai tarvetta uusille ominaisuuksille. Kokouksissa huomasi aina, jos jokin ominaisuus ei ole helposti ymmärrettävä, kun asiakas opetteli järjestelmän käyttöä. Asia, joka toistui monessa kokouksessa, oli termien väärinymmärrys: asiakas ei aina käsittänyt, mikä on organisaation (Organization), arviointiryhmän (Evaluation) ja kyselyn (Survey) ero, vaikka asiat oli kehitetty yhdessä tehdyn projektisuunnitelman mukaan. Myöhemmin päädyttiin uudelleen nimeämään organisaatio-termi käyttäjäryhmäksi (Group).

Suurin muutos alkuperäisestä suunnitelmasta oli fokuksen siirtyminen linkkien kautta jaettaviin kyselyihin kirjautuneiden käyttäjien sijasta. Linkkiominaisuus tuli ensimmäiseen versioon erittäin myöhään, joten se jäi hieman keskeneräiseksi toteutukseksi. Ensimmäisellä versiolla kyselyyn oli mahdollista luoda vastauslinkki vain omalle aktiiviselle organisaatiolle, mutta uudessa versiossa käyttäjä voi luoda oman linkin jokaiselle arviointiryhmän organisaatiolle tai yhden yhteisen linkin, jossa vastaaja valitsee, mihin organisaatioon kuuluu.

Projektin loppupuolella asiakkaat halusivat järjestelmästä monikielisen, mutta aluksi tuettaisiin vain englantia ja suomea. Olin aloittanut tämän toiminnon kehittämisen jo ennen kuin sitä varsinaisesti pyydettiin, koska se oli projektisuunnitelmassa mahdollinen jatkokehityskohde. En käyttänyt valmista kääntäjäliitännäistä, koska en löytänyt sellaista, joka toimii kuten haluan, joten päätin kehittää oman. Kehitin kääntäjäkomponentin niin, että siihen on helppo lisätä uusia kieliä, jos niille tulee tarvetta. Kääntäjää kehittäessä tutkin, miten muut kielen kääntäjät toimivat, ja yhdistelin asioita, joista pidin omaan kääntäjään.

Huomasin, että järjestelmän monessa paikassa tekstillä tulisi olla käännökset niin monikko- kuin yksikkömuodossa, joten lisäsin tämän toiminnallisuuden kääntäjään. (Liite 3.)

Kääntäjä toimii niin, että ensin luodaan uusi Translator-instanssi, esimerkiksi `const translator = new Translator('fi')`. Kääntäjäinstanssia luodessa sen oletuskielen voi vaihtaa laittamalla halutun kielen lyhenteen sulkujen sisään, muuten kieli on oletuksena englanti. Kääntäjän kieltä voi myös vaihtaa lennossa `changeLang()`-funktiolla, esimerkiksi `translator.changeLang('en')`. Käännökset ovat kirjoitettu esimerkkikoodin 10 lailla `translations`-objektiin. Teksti käännetään käyttämällä kääntäjän `trans()`-funktiota. Funktio tarvitsee parametriksi käännettävän tekstin, ja jos tekstissä on muuttujia, ne merkitään tekstiin kirjoittamalla niiden paikalle `%value%`. Muuttujat lisätään erikseen omina parametreinä, esimerkiksi `translator.trans('Are you sure you want to delete %value% users', 4)`. Kääntäjä korvaa `%value%`-kohdat tekstistä parametreinä annetuilla muuttujilla siinä järjestyksessä, missä ne on annettu. Aiemmin mainittu teksti käännetään esimerkkikoodin 10 mukaan tekstiksi ”Oletko varma, että haluat poistaa 4 käyttäjää?” jos annettu muuttuja olisi ollut ”1”, kääntäjä olisi käyttänyt yksikkökäännöstä.

```
"Are you sure you want to delete %value% users?": {
  "en": "Are you sure you want to delete %value% users?",
  "fi": "Oletko varma, että haluat poistaa %value% käyttäjää?",
  "enSingle": "Are you sure you want to delete this user?",
  "fiSingle": "Oletko varma, että haluat poistaa tämän käyttäjän?"
}
```

Esimerkkikoodi 10. Esimerkkikäännös, jossa on käännökset monikko- ja yksikkömuodossa.

6.4 Järjestelmän lopputulos ja tulevaisuus

Järjestelmä toimii tällä hetkellä Metropolian alidomainilla, ja asiakkaat aikovat käyttää sitä tulevaisuudessa erilaisten kyselyiden luonnissa. Järjestelmällä on jo järjestetty muutama varsinainen kysely, jossa oli merkittävä määrä oikeita vastaajia. Nämä kyselyt oli luotu käyttäen jaettavaa linkkiä, joten järjestelmän alkuperäisen käyttötavan mukaista kyselyä ei ole vielä järjestetty. Saadun palautteen mukaan kyselyyn vastaaminen linkin kautta on yksinkertaista ja helppoa, niin kuin toivottiinkin. Valitettavasti kyselyn luojan puolella käytettävyys ei ole yhtä hyvä, kuin toivottiin. Kyselyn luojan puolella ongelmia tuo arviointiryhmän ja käyttäjäryhmien tarkoituksen hahmottaminen. Nämä ominaisuudet

ovat tärkeitä varsinkin kyselyn tulosnäkymän kuvaajien kannalta. Ongelmien välttämistä varten kirjoitettiin kattava käyttöopas järjestelmän eri näkymistä ja niiden käytöstä.

Vaikka tämä projekti tuli valmiiksi, järjestelmä ei silti ole valmis. Viimeisessäkin kokouksessa saatiin uusia jatkokehitysehdotuksia, kuten uusia kriteerityyppejä ja kysymyksen monistaminen. Nämä ehdotukset on kirjoitettu Trelloon, jotta järjestelmän seuraavat kehittäjät tietävät, mitä tehdä. Jatkokehitystä varten koodi on myös dokumentoitu paremmin kuin vuoden 2009 versio. Dokumentaatio oli asia, jota asiakas korosti alusta alkaen, ettei käy samoin kuin vanhalle versiolle.

7 Yhteenveto

Insinöörityön alkuperäinen tavoite oli luoda selaimessa toimiva sovellus, jolla asiakas voi luoda KOLA-itsearviointikyselyitä helpommin ja suurempia määriä. Kyselyiden tuloksien käsittelystä oli tarkoitus tehdä vaivatonta kuvaajilla ja valmiilla Excel-vedoksella. Projektin kehityksen aikana järjestelmän vaatimukset ja toivotut ominaisuudet muuttuivat ja lisääntyivät, mikä teki koko prosessista vaikeampaa ja aikaa vievää.

Olisi ollut hyödyllistä, jos jo ensimmäisten kokousten aikana olisi käyty asiakkaiden kanssa tarkasti läpi, mitä kullakin järjestelmän sivulla tulisi tapahtua. Mutta koska kyseessä oli minulle ensimmäinen oikea asiakasprojekti, ei osattu heti tuoda asiaa esille. Tämän vuoksi päädyttiin moneen väärinymmärrykseen ja sekaannukseen etenkin käytettyjen termien muodoissa, joiden korjaamiseen käytettiin turhaa aikaa.

Olisi myös ollut hyödyllistä, jos teknologiapinoa ei olisi vaihdettu kesken projektin, mutta Sails-kehyksen kanssa tulleita ongelmia ei osattu ennustaa. Teknologiapinon vaihto kuitenkin auttoi tekemään järjestelmästä vakaamman ja jatkokehityksen kannalta paremman.

Järjestelmän selainpuolen näkymät lopulta toteutettiin yhdistelemällä EJS:ää, moderneja JavaScript-ratkaisuja ja Reactia. Tyylien luomiseen käytettiin Foundation 6 -kehystä ja SCSS:ää. Datan visualisointiin käytettiin D3-JavaScript-kirjastoa. EJS, SCSS, Foundation 6 ja jotkin JavaScriptin uusista ominaisuuksista olivat minulle tuttuja asioita jo entuudestaan, joten osasin hyödyntää niitä järjestelmän kehityksessä.

D3 ja datan visualisointi olivat minulle uusia asioita, joiden opetteluun meni aikaa. D3:n syntaksi vaikutti aluksi hieman pelottavalta, mutta kun tutustui tarkemmin erilaisiin internetin opetusmateriaaleihin ja teki erilaisia kokeilukuvaajia, siitä tuli nopeasti ymmärrettävää ja pystyin tekemään projektissa tarvittavia kuvaajia ilman suuria ongelmia. React oli minulle myös uusi asia, joten sen opetteluun meni paljon aikaa. Onneksi Reactiin löytyi paljon opetusmateriaalia ja esimerkkejä, joista oppi ymmärtämään, miten se toimii ja miten sitä voi soveltaa tässä projektissa. Opin käyttämään Reactia parhaiten, kun aloin kääntää yhtä järjestelmän oikeaa komponenttia Reactille.

Järjestelmä saatiin lopulta käyttöön, kaikki vaaditut toiminnot löytyvät järjestelmästä ja järjestelmälle on luotu käyttöohjeet englanniksi ja suomeksi. Järjestelmän

helppokäyttöisyys kärsi hieman monimutkaisista ominaisuuksista, kuten arviointiryhmät ja käyttäjäryhmät, mutta ne ovat tuiki tärkeitä komponentteja tulosten oikein näyttämisessä. Näitä ominaisuuksia voisi yksinkertaistaa tulevaisuudessa jatkokehityksenä.

Lähteet

1 Schrey-Niemenmaa, Katriina & Karhu, Markku. 2018. Insinöörityön tilaajat, Metropolia Ammattikorkeakoulu. Espoo. Sähköposti 27.3.2018.

2 Schrey-Niemenmaa, Katriina & Hellman, Juha. 1995. KOLA-Käsikirja. Helsinki: Tekniikan Akateemisten Liitto.

3 Rouvrais, S., Lassudrie, C. & Landrac, G. Guidelines and Kit to Support Cross-Sparring Procedures. Verkkoaineisto. <http://projects.au.dk/fileadmin/cross-sparring/QAEMP-deliverable51-CSprocess.pdf>. Luettu 4.4.2018.

4 Self-evaluation Handbook. 2015. Verkkoaineisto. Cross-sparring. EU. <http://projects.au.dk/fileadmin/_migrated/content_uploads/Self_Evaluation_Hand_Book_v1.pdf>. Luettu 3.4.2018.

5 FusionCharts. Verkkoaineisto. Wikipedia. <https://en.wikipedia.org/wiki/FusionCharts>. Luettu 31.10.2017.

6 D3. Verkkoaineisto. D3.js. <https://d3js.org/>. Luettu 30.10.2017.

7 Bostock, Mike. 2017. For Protovis Users. Verkkoaineisto. <http://mbostock.github.io/d3/tutorial/protovis.html>. Luettu 30.10.2017.

8 Rahman, Syed Fazle. 2015. 15 Best JavaScript charting libraries. Verkkoaineisto. <https://www.sitepoint.com/15-best-javascript-charting-libraries/>. Luettu 30.10.2017.

9 Ocamica, Santiago. 2016. Pie Chart (d3js v4). Verkkoaineisto. <https://bl.ocks.org/santi698/f3685ca8a1a7f5be1967f39f367437c0>. Luettu 30.10.2017.

10 Bostock, Mike. 2017. D3 - Gallery. Verkkoaineisto. <https://github.com/d3/d3/wiki/Gallery>. Päivitetty 4.10.2017. Luettu 30.10.2017.

11 D3 Plugins. 2017. Verkkoaineisto. GitHub. <https://github.com/d3/d3/wiki/Plugins>. Päivitetty 17.10.2017. Luettu 31.10.2017.

12 Nelli, Fabio. 2013. Beginning JavaScript Charts. E-kirja. Apress.

13 EJS - JavaScript Templates. Verkkoaineisto. EmbeddedJS <www.embeddedjs.com>. Luettu 13.3.2018.

14 EJS - Embedded JavaScript Templates. Verkkoaineisto. EJS. <http://ejs.co/#docs>. Luettu 13.3.2018.

15 Ivanovs, Alex. 2017. Top 10 Templating Engines for JavaScript To Improve and Simplify Your Workflow 2017. Verkkoaineisto. Colorlib. <https://colorlib.com/wp/top-templating-engines-for-javascript>. Luettu 21.3.2018.

16 React. Verkkoaineisto. React.js. <https://reactjs.org/>. Luettu 8.3.2018.

17 Vipul A. M. & Prathamesh Sonpatki. 2016. ReactJS by Example - Building Modern Web Applications with React. E-kirja. Packt Publishing.

18 Lindley, Cody. 2016. React Enlightenment. E-kirja. Frontend masters.

19 Aranda, Michael. 2017. What's the difference between JavaScript and ECMAScript? Verkkoaineisto. freeCodeCamp. <https://medium.freecodecamp.org/whats-the-difference-between-javascript-and-ecmascript-cba48c73a2b5>. Luettu 15.3.2018.

20 Engelschall, Raif S. 2017. ECMAScript 6 - New Features: Overview & Comparison. Verkkoaineisto. ES6-features. <http://es6-features.org>. Päivitetty 20.12.2017. Luettu 19.3.2018.

21 Grover, Deepak. 2017. ES6 For Humans. E-kirja. Apress.

22 Promise - JavaScript. Verkkoaineisto. MDN. <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise>. Luettu 15.3.2018.

23 async function - JavaScript. Verkkoaineisto. MDN. <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function>. Luettu 16.3.2018.

24 Johnson, Nicholas. What is Babel, and how will it help you write JavaScript? Verkkoaineisto. Nicholas Johnson. <http://nicholasjohnson.com/blog/what-is-babel>. Luettu 17.3.2018.

25 Foundation. Verkkoaineisto. Foundation. <https://foundation.zurb.com>. Luettu 19.3.2018.

26 XY-grid. Verkkoaineisto. Foundation. <https://foundation.zurb.com/sites/docs/xy-grid.html>. Luettu 19.3.2018.

27 CSS preprocessor. Verkkoaineisto. MDN. <https://developer.mozilla.org/en-US/docs/Glossary/CSS_preprocessor>. Luettu 20.3.2018.

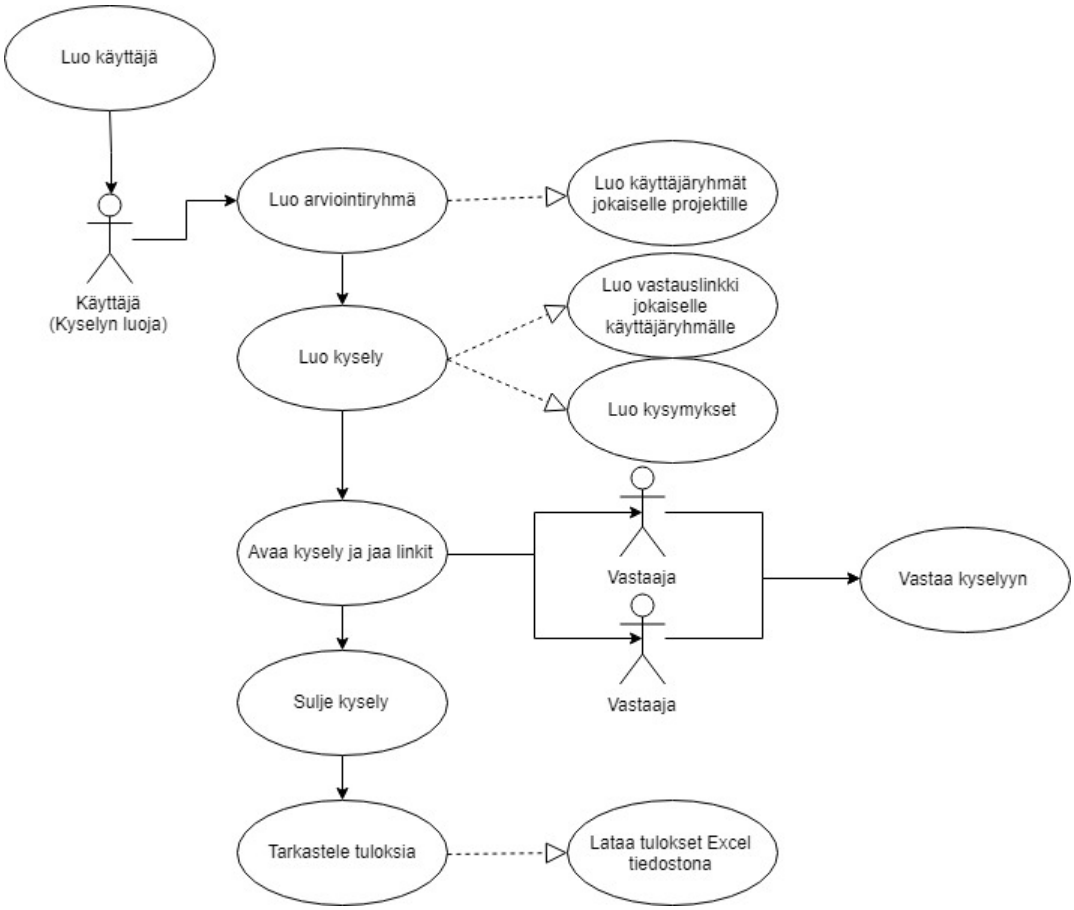
28 Sass documentation. Verkkoaineisto. Sass. <https://sass-lang.com/documentation/file.SASS_REFERENCE.html>. Luettu 20.3.2018.

Piirakkakuvaajan JavaScript-koodi D3-kirjastolla

```
// Data josta tehdään kuvaaja.
var data = [{ label: 'Kahvi', count: 3 },
{ label: 'Tee', count: 2 },
{ label: 'Vesi', count: 4 },
{ label: 'Mehu', count: 1 }];

// Määritellään kuvaajan leveys ja korkeus.
var width = 400,
height = 360,
radius = Math.min(width, height) / 2;

// Annetaan kuvaajan väriskaala. Värit voi itse määrittää.
var color = d3.scaleOrdinal()
.range(['#A60F2B', '#648C85', '#B3F2C9', '#528C18', '#C3F25C']);

// Luodaan SVG elementti, jossa itse kuvaaja näkyy. SVG:lle annetaan koot,
jotka määritettiin aikaisemmin.
var svg = d3.select('#graph').append('svg')
.attr('width', width).attr('height', height)
.append('g').attr('transform', 'translate(' + (width / 2) + ',' +
(height / 2) + ')');

// Koska kyseessä on piirakkakuvaaja, kuvaajasta tulee tehdä pyöreä. Tässä
määritellään, millainen kaari kuvaajalle tulee.
var arc = d3.arc()
.innerRadius(0)
.outerRadius(radius);

// Jos kuvaajalle annetaan selitteet jokaiselle lohkolle, tarvitaan myös kaari
selitteteksteille.
var labelArc = d3.arc()
.outerRadius(radius - 40)
.innerRadius(radius - 40);

// Piirakkakuvaajaa varten D3:lla on pie() funktio.
var pie = d3.pie()
.value(function(d) { return d.count; })
.sort(null);

// Tässä piirretään itse kuvaaja edellä määritellyllä koodilla.
var g = svg.selectAll(".arc")
.data(pie(data))
.enter().append("g")
.attr("class", "arc");
g.append("path")
.attr("d", arc)
.style("fill", function(d) { return color(d.data.label); });
g.append("text")
.attr("transform", function(d) { return "translate(" + labelArc.centroid(
d) + ")"; })
.attr("dy", ".35em")
.text(function(d) { return d.data.label; });
```

Koodi, jossa luodaan yksinkertainen kaksiulotteinen piirakkakuvaaja [9].

UML-kaavio käyttötapauksesta



Kääntäjäkomponentti

```
const translations = require('./translations');

class Translator {
  constructor(lang = 'en') {
    this.lang = lang;
    this.translations = translations;
    this.re = new RegExp("%value%", "g");
  }
  static getSupportedLocales() {
    return ['en', 'fi'];
  }

  changeLang(lang) {
    this.lang = lang;
  }

  trans(key, ...extra) {
    if(!key) return null;
    let translation = this.translations[key];
    if (!translation) {
      console.log(`Translator: Missing value for key "${key}"`);
      return key;
    } else translation = translation[this.lang];

    if (extra && translation.match(this.re)) {
      // Mismatching %values% and extras
      if (extra.length !== translation.match(this.re).length) {
        console.error('Number of %value%:s and given replace values does not
match');
        return null;
      }
      // Single variable
      else if (extra && extra.length === 1) {
        if (parseInt(extra) === 1) {
          // Try to find a singular version of the string, if not found use
the default string
          translation = this.translations[key][this.lang + "Single"] ||
this.translations[key][this.lang];
          return translation.replace(this.re, extra);
        }
        return translation.replace(this.re, extra);
      }
      // Multiple variables
      else if (extra && extra.length > 1) {
        if (extra.includes(1) || extra.includes('1')) {
          translation = this.translations[key][this.lang + "Single"] ||
this.translations[key][this.lang];
        }
        if (extra.length !== translation.match(this.re).length) {
          console.log(extra);
          console.error('Number of %value%:s and given replace values does not
match');
          return null;
        } else {
          let i = 0;
          return translation.replace(this.re, () => {
            const extraKey = extra[i];
            i++;
            return extraKey;
          });
        }
      }
```

```
    }
    // No variables
    else if (extra.length === 1 && !translation.match(this.re)) {
      // Single variable
      if (parseInt(extra) === 1) {
        // Try to find a singular version of the string, if not found use the
default string
        return this.translations[key][this.lang + "Single"] || this.transla-
tions[key][this.lang];
      }
      return this.translations[key][this.lang];
    }
    else if (key.match(this.re)) {
      console.error('You must provide a value to replace %value% in your
string');
      return null;
    } else {
      return translation;
    }
  }
}
```