Markku Stenius

Verkkopohjainen laadunvarmistustyökalu asiakaspalveluun

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

1.4.2017

Metropolia

Tekijä Otsikko Sivumäärä Aika	Markku Stenius Verkkopohjainen laadunvarmistustyökalu asiakaspalveluun 31 sivua 1.4.2017
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja	Lehtori Olli Hämäläinen

Insinöörityössä oli tavoitteena luoda laadunvarmistusohjelmisto asiakaspalveluympäristöön. Asiakaspalvelun kannalta tärkeintä oli päästä eroon käsin täytettävistä lomakkeista ja vaikeasti toteutettavasta tilastoinnista. Ohjelmiston ominaisuudet ja tarvittavat toiminnallisuudet suunniteltiin yhdessä asiakaspalveluosaston päälliköiden kanssa, jotta ne vastaisivat vaatimuksia mahdollisimman tarkkaan.

Ohjelmistolle asetettujen vaatimusten vuoksi sitä lähdettiin kehittämään usean käyttäjän järjestelmänä, joka sisältää kaksi eri käyttäjätasoa. Tavoitteena oli tarjota omat näkymät laadunvarmistuksesta vastaaville henkilöille ja arvioitaville asiakaspalvelijoille.

Insinöörityössä päätettiin käyttää RESTful-arkkitehtuurilla toimivaa sovellusrajapintaa ja JavaScript-pohjaista käyttöliittymää. Työkaluina päätettiin käyttää keveitä ohjelmointikehyksiä, PHP-pohjaista Slim-ohjelmointikehystä sovellusrajapinnan luomiseen ja JavaScriptiä käyttävää Backbone-ohjelmointikehystä käyttöliittymään. Erityistä huomiota kiinnitettiin palvelimen ja selainnäkymän erottamiseen toisistaan ja käyttöliittymän helppokäyttöisyyteen.

Insinöörityön tuloksena syntyi ohjelmisto, joka mahdollisti laadunvarmistuslomakkeiden luomisen ja täyttämisen samassa ympäristössä. Koska arvioinnit tallentuvat tietokantaan, tuloksia pystytään tarkastelemaan tarkemmin kuin aikaisemmin.

Tavoitteena oli helpottaa laadunvarmistusprosessia, ja ensimmäisten käyttökokemusten mukaan tämä tavoite saavutettiin jo ensimmäisessä versiossa. Testausvaiheen jälkeen ohjelmistoa on tarkoitus kehittää edelleen ja lisätä muun muassa tilastointiin liittyviä ominaisuuksia.

Avainsanat	laadunvarmistus, Backbone, JavaScript, PHP, Slim, RESTful, sovellusrajapinnat

Author Title Number of Pages Date	Markku Stenius Web based quality assurance application for customer service 31 pages 1 April 2017
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor	Olli Hämäläinen, Senior Lecturer

The goal of the final year project was to create a quality assurance application for customer service environment. The main purpose of the software application was to allow team leaders create quality assurance reports without relying on Excel sheets and storing the results directly to the database.

All necessary features were planned together with customer service team leaders in order to fit all necessary requirements. Due to the nature of the application, it was decided that it should support multiple user accounts with two different levels of access. Both customer service agents and team leaders should have their own application views with relevant features.

It was decided to separate server and browser views by using RESTful API and JavaScript based user interface. The primary development tools were lightweight frameworks Slim for server side, and Backbone.js for front-end.

The application allowed quality assurance reports to be created, viewed and filled in the same environment. Furthermore, as all the information is stored in structured fashion, results and trends can be observed with ease.

The purpose of the project was to simplify the quality assurance process. After a brief testing period, it was clear that this goal was reached with the first version of the application.

Keywords	Quality assurance, API, PHP, Backbone, JavaScript, Slim, RESTful

Sisällys

1 Johdanto

Insinöörityön tavoitteena on luoda laadunvarmistusohjelmisto asiakaspalveluympäristöön. Tarkoituksena on helpottaa laaduntarkistusprosessia siirtymällä pois käsin täytettävistä lomakkeista ja vaikeasti toteutettavasta tilastoinnista.

Asiakaspalvelun laadun tarkkailu yrityksen sisäisesti voi olla tärkeämpää kuin asiakastyytyväisyyden seuraaminen palvelutilanteen jälkeisillä kyselyillä. Asiakkaat vastaavat keskimäärin viidesosaan lähetetyistä kyselyistä, ja vastaukset ovat yleensä tapauksia, joissa asiakaskokemus on ollut joko huomattavan hyvä tai huono. Tämän vuoksi arviot ovat yleensä erittäin polarisoituneita, eikä niiden pohjalta pystytä tarkkaan arvioimaan asiakaspalvelun laatua.

Asiakaspalvelijoiden vastaukset ja yleinen tapa kommunikoida voivat kelvata suurelle osalle asiakaskuntaa, mutta jos tavoite on saada kommunikaatiotaidot ja asiantuntemus samalle tasolle, tarvitaan työkalu sisäisen laadun mittaamiseen. Laadunvarmistuksen hyvänä puolena on, että saadaan tarkkaa tietoa siitä, mitä alueita pitäisi kehittää, sekä mahdollisuus hyvien suoritusten palkitsemiseen.

Insinöörityön tilannut asiakasyritys tarjoaa VoIP- ja pilvipalveluita puhelinmyynti- ja asiakaspalvelutarkoituksiin. Tarjottavia palveluita käytetään yleensä puhelinpalvelukeskusten ”all-in-one”-ratkaisuna. Koska kommunikointi on pääasiassa yrityksien välistä, on noussut tarve kehittää keino seurata asiakaspalvelijoiden antaman palvelun laatua. Usein tähän tarkoitukseen käytetään Excel-pohjia, joiden käsittely ja täyttäminen voi olla aikaa vievää. Käytettyyn tiketti- ja asiakastukijärjestelmään on ollut tarjolla useita laadunvarmistamiseen liittyviä laajennuksia, mutta mikään niistä ei tarjonnut sellaisenaan vaadittuja ominaisuuksia. Verkossa tarjotut ratkaisut laadunvarmistukseen ovat erikoistuneita tuotannon laadunvarmistamiseen, ja sellaisenaan ne olisivat turhan monimutkaisia käyttötarkoitukseen.

Tässä raportissa käsitellään insinöörityönä tehtävän ohjelmiston luomisen vaiheet suunnittelusta ja toteutuksesta testausvaiheeseen asti. Työssä käydään myös läpi ohjelmoinnissa käytettävät kirjastot, tekniset ratkaisut, tietokannan rakenne sekä kehittämisen aikana kohdattavat ongelmat ja niiden ratkaisut.

2 Laadunvarmennusohjelmiston perusvaatimukset

2.1 Perustoiminnot

Kun laadunvarmennusohjelmistoa lähdettiin suunnittelemaan ja asetettuja vaatimuksia käytiin läpi, kävi selväksi, että ohjelma vaatisi useamman näkymän, jotta toiminta olisi mahdollisimman yksinkertaista. Tultiin tulokseen, että ainakin seuraavat näkymät olisivat välttämättömiä:

- kysymysten ja kysymyslistojen hallintanäkymä
- arviointilomakkeiden täyttö-, selaus- ja muokkausnäkymä
- käyttäjänhallinta.

Ohjelmassa on näiden lisäksi vielä kaksi eri käyttäjäryhmää: arvioijat ja arvioitavat. Arvioijan vastuulla on itse arviointilomakenäkymän lisäksi kysymyslistojen ja käyttäjien hallinnointi. Arvioitavan näkymä antaa ainoastaan mahdollisuuden tarkastella käyttäjän omia arvioita ja vaihtaa arvioinnin tilaa. Arviointilomakenäkymässä kummallakin käyttäjäryhmällä on turvallisuussyistä rajoituksia: arvioitavat näkevät ainoastaan omat arvionsa ja arvioijat pystyvät tarkastelemaan vain itse tekemiään arvioita. Kuviossa 1 voidaan nähdä ohjelmiston päätoiminnot ja eri käyttäjätasojen oikeudet niiden käyttämiseksi.

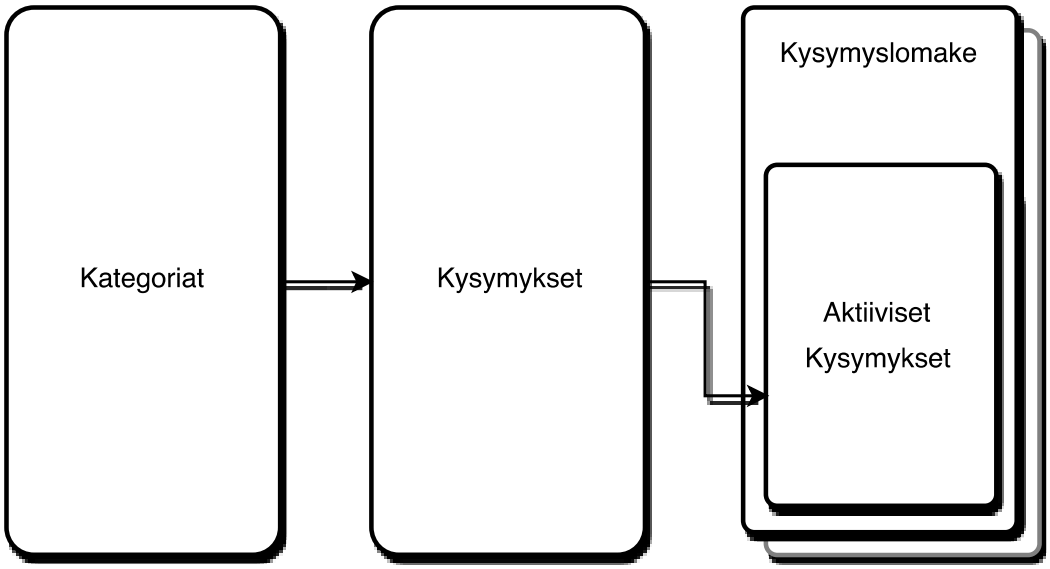
Kuvio 1. Laadunvarmistusohjelmiston toiminnallisuudet ja käyttöoikeudet.

Käyttöliittymän osalta ainoa toivomus oli helppokäyttöisyys. Käyttöliittymää lähdettiin siis suunnittelemaan periaatteella, että sitä pystyisi käyttämään tablettimaisesti ”yhdellä sormella” ja niin, että ainoastaan tarvittavat ominaisuudet ovat kerrallaan näkyvissä. Käytettävyyden ohjenuoraksi valikoitiin Googlen ohjeistus mobiilisovellusten käyttöliittymän kehittämiseen [1] ja eritoten sen käytettävyysosio.

2.2 Kysymykset ja kysymyslomake

Kysymysten ja kysymyslomakkeiden koostamisnäkymää suunniteltaessa tultiin siihen tulokseen, että kysymysten ja lomakkeiden hallinta toimisi parhaiten, jos se koostuisi useasta osasta.

Aluksi luodaan kysymyskategoriat, esimerkiksi ”Peruskysymykset”, ”Sähköposti” ja ”Puhelin”, joiden sisälle lisätään haluttu määrä kysymyksiä. Tämän jälkeen luodaan lista, johon lisätään kysymykset halutuista kategorioista. Aktiiviselle kysymykselle määritellään painoarvo, joka pätee ainoastaan kyseisen listan sisällä, eli samalla kysymyksellä voi olla useita eri pistemääriä riippuen siitä, missä se sijaitsee (kuvio 2). Esimerkiksi puhelulomakkeessa oikeinkirjoitusta arvioiva kysymys ei ole yhtä tärkeä kuin sähköpostilomakkeessa.



Kuvio 2. Kysymyslomakkeiden luomisnäkymän toimintaperiaate ja hierarkia.

2.3 Käyttäjänhallinta

Käyttäjänhallinta on ehkä yksinkertaisin kaikista ohjelmiston osista. Sivun päätoiminnot ovat käyttäjien listaus, uusien käyttäjien luominen ja olemassa olevien käyttäjien tietojen muokkaaminen.

Sivun päänäkymä on lista, jossa näkyvät käyttäjien tiedot ja muita yksityiskohtia, kuten käyttäjätaso ja sähköpostiosoite. Käyttäjän luomista varten on oma näkymä, johon voidaan helposti syöttää käyttäjän tiedot.

Käyttäjätietojen muokkausnäkymä on kaikessa yksinkertaisuudessaan lomake, josta voidaan tarkastella käyttäjän tietoja ja tarpeen tullen tehdä muutoksia niihin. Koska ainoastaan arvioijalle päätettiin antaa mahdollisuus vaihtaa käyttäjien salasanoja, täytyy käyttäjälle lähettää muutoksen yhteydessä sähköpostiviesti, joka sisältää tiedon tapahtuneesta ja uuden salasanan. Salasanasähköpostit lähetetään ainoastaan sisäisesti, ja viestit ohjeistetaan poistettavaksi heti salasanan tallentamisen jälkeen.

2.4 Arviointilomakkeiden täyttäminen ja käsittely

Arviointilomakkeiden käsittelysivun päänäkymä on lista, josta pystyy tarkastelemaan arviointilomakkeita, niiden pistemääriä ja muita haluttuja ominaisuuksia. Listaa voi suodattaa hakusanoilla ja tuloksia rajata arvioinnin päivämäärällä.

Arviointinäkymässä yhdistetään kaikki ohjelmiston aiemmin luodut elementit: käyttäjät, kysymyslomakkeet ja kysymykset. Näiden tietojen lisäksi itse arvioitavaan kohteen, esimerkiksi sähköpostikeskustelun, tiedot pitää pystyä lisäämään lomakkeelle.

Arviointiprosessi kulkee seuraavalla tavalla: valitaan käyttäjä, jonka suoritusta halutaan arvioida, minkä jälkeen valitaan kysymyslista, joka parhaiten sopii toteutuneen viestinnän arviointiin. Valinnan jälkeen lista avautuu arviointinäkymään ja sen sisältämät kysymykset tulostetaan näkymään.

Listassa jokaisen kysymyksen kohdalla on valinta ”Kyllä” ja ”Ei” sekä tekstikenttä, johon voi tarvittaessa kirjoittaa lyhyen kommentin. Jos kysymykseen vastataan ”ei”, vähennetään kokonaispistemäärästä kysymyksen painoarvon mukaisesti pisteitä. Pisteas-

teikoksi valittiin prosenttiarvostelun mukaan 0–100 %, mutta ilman mahdollisuutta negatiiviseen arvoon, eli vaikka arvioija joutuisikin valitsemaan ”ei” jokaiseen kysymykseen, pistemäärä ei voi mennä nollan alapuolelle.

Toteutunut kirjeenvaihto kopioidaan sellaisenaan vastaavaan tekstikenttään ja linkki alkuperäiseen viestiketjuun. Kommenttikenttään voidaan tarvittaessa kirjoittaa lisätietoja arvioinnista. Taulukossa 1 ovat listattuna kentät, joista arviointilomake koostuu.

Taulukko 1. Arviointilomakkeen kentät, pakolliset kursivoituina.

Kenttä	**Kuvaus**
Arvioitava	Valittavana kaikki arvioitavat käyttäjät
Kysymyslomake	Listalla kaikki aikaisemmin luodut arviointilomakkeet
Arvioitavan tiketin tms. sulkemispäivä	Päivämäärävalitsin
Toteutunut viestintä	Tekstikenttä, johon kopioidaan asiakkaan ja agentin välinen kirjeenvaihto
Linkki	Linkki ulkoiseen resurssiin, esimerkiksi tikettiin tai äänitiedostoon
Kommentit	Tekstikenttä yleistä kommentointia varten
Pistemäärä	Arviointilomakkeen pistemäärä (automaattinen)

Kun arvio on tallennettu järjestelmään, sitä voi tarkastella muokkausnäkymässä. Muokkausnäkymässä näytetään toteutunut arviointi kaikkine tietoineen ja kysymyslista vastauksineen. Lisäkenttänä näkymässä on arviointilomakkeen tila -valikko, josta voi valita, onko arviointi valmis tai riitautettu vai odottaako se agentin tai arvioijan viimeistä tarkistusta.

Samasta näkymästä tehdään arvioitaville oma versio, jossa he voivat tarkastella arviointeja sekä muuttaa arviointilomakkeen tilaa riitautetuksi tai odottamaan arvioijan toimia.

3 Ohjelmistossa käytetyt tekniikat

Ohjelmiston rakenteessa päätettiin käyttää selkeää jakoa käyttöliittymän ja palvelimen välillä. Jokaiselle sivulle luodaan HTML-pohja, jonka sisältämä JavaScript-ohjelma alustaa vaadittavat toiminnallisuudet ja noutaa tiedot palvelimelta. Kaikki ohjelmiston tarvitsema tieto haetaan siis rajapinnasta JSON-tiedostomuodossa suoraan käyttöliittymään, jossa vastaanotettu tieto käsitellään ja tulostetaan näkymään. Käyttöliittymässä tehdyt muutokset lähetetään suoraan palvelimen rajapintaan, missä ne käsitellään ja tallennetaan tietokantaan.

Käyttöliittymän rakentamiseen päädyttiin käyttämään Backbone-ohjelmointikehystä sen käyttämän MVC-arkkitehtuurin vuoksi ja koska se käyttää oletuksena rajapintoja ja JSON-tiedostomuotoa tiedon siirtämiseen. Palvelimen puolelle käyttöön valittiin Slim Framework, joka on PHP-pohjainen ohjelmistokehys RESTful-periaatteella toimivien rajapintojen kehittämiseen.

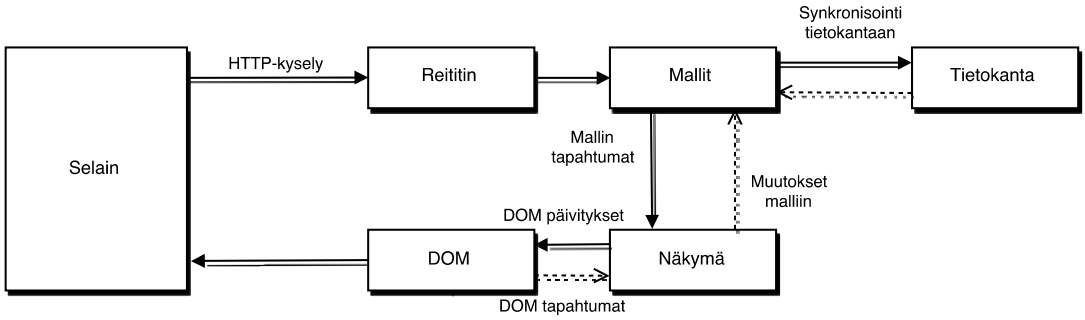
3.1 Käyttöliittymän luomiseen käytetyt kirjastot ja tekniikat

3.1.1 Backbone-ohjelmointikehys

Backbone on JavaScriptillä kirjoitettu kirjasto, joka mahdollistaa yksisivuisten verkko-ohjelmistojen luomisen. Kirjastolle on ominaista keveys (ainoa varsinainen vaatimus on saman kehittäjän työkalukirjasto Underscore), RESTful-arkkitehtuurin ja JSON-tiedostonmuodon käyttäminen mallien ja tietokannan synkronisoinnissa ja MVC-arkkitehtuuria noudattava toimintamalli [2].

Kuviosta 3 voidaan nähdä, että Backbone-ohjelma koostuu kolmesta osasta:

- malli tai kokoelma malleja
- näkymä, joka näyttää mallin sisältämät tiedot käyttäjälle halutulla tavalla ja käsittelee käyttäjän tekemät toiminnot
- sivupohjat, joihin näkymän tarjoama näkymä perustuu
- reititin, joka mahdollistaa yksisivuisen ohjelman sisällä liikkumisen ”takaisin”-toimintoa käyttäen ja tekee siitä niin sanotusti linkattavan eli eri näkymille voi määritellä oman URL-osoitteen.



Kuvio 3. Backbone-sovelluksen toimintaperiaate

Mallit viestivät tietokannan kanssa käyttäen JSON-tiedostomuotoa. Tietokantaan otetaan yhteys käyttäen RESTful-arkkitehtuurin mukaisia URL-osoitteita, jotka määritellään suoraan malliin. Samaa osoitetta käytetään tietojen lukemiseen, lisäämiseen, poistamiseen ja muokkaamiseen, ja itse toiminto määritellään RESTful-arkkitehtuurin mukaisesti HTTP-metodeita käyttäen. [3, s. 9.]

Reititintä käytetään yleisesti ohjelman aloittamiseen, eli kun sivu on latautunut kokonaisuudessaan, avataan URL-osoitteen mukainen osa reitittimestä. Reitittimeen voi määritellä haluamansa määrän osoitteita, jotka avaavat halutun näkymän käyttäjälle [3, s. 62]. Tällaisia osoitteita voivat olla esimerkiksi

- ”/” eli sisällyssivu, joka avaa listan käyttäjistä
- ”/user/id”, joka näyttää id:llä määritellyn mallin sisältämät tiedot

Koska ensimmäisessä esimerkissä haetaan lista, on järkevää käyttää kokoelmaa yksittäisien mallien hakemisen sijaan. Kokoelma on kaikessa yksinkertaisuudessaan lista malleja, jossa yksi rivi sisältää yhden mallin [3, s. 48].

Kun käyttäjä avaa sisällyssivun osoitteen, reititin luo uuden instanssin käyttäjäkokoelmasta ja lataa tietokannasta kaikkien käyttäjien tiedot. Tämän jälkeen luodaan uusi näkymä, joka tulostaa mallin tiedot käyttäjälle käyttäen näkymässä määriteltyä HTML-pohjaa [3, s. 35].

Toisessa tapauksessa haetaan tietty käyttäjä tietokannasta, eli on loogista käyttää mallia kokoelman sijaan. Kun käyttäjä avaa esimerkiksi osoitteen ”/user/12”, reititin luo uuden instanssin käyttäjämallista ja hakee annetulla tunnisteella määritellyn tietueen

tietokannasta ja liittää sen malliin. Tämän jälkeen malli liitetään käyttäjän tiedot tulostavaan näkymään, joka näyttää tiedot HTML-pohjan mukaisesti. [3, s. 28.]

Pohjien tulostamiseen voi käyttää joko Backbonen mukana tulevan Underscoren tarjoamaa sivupohjienhallintaa tai kolmannen osapuolen kirjastoa, esimerkiksi Handlebars.js [3, s. 76]. On myös mahdollista tulostaa tiedot käyttäen Reactia, Polymeria tai aivan omaa tapaa, valinta on koodin kirjoittajan.

Näkymässä rakennetaan myös itse käyttöliittymä, jotta käyttäjät voisivat lisätä, poistaa tai muuttaa mallien sisältämää tietoa. Käytännössä tämä tarkoittaa, että näkymään lisätään tekstikenttiä, nappuloita ja muita käyttöliittymäkomponentteja, jotka kommunikoivat mallien ja kokoelmien kanssa. Kun mallien tai kokoelmien tiedot muuttuvat, ne synkronisoidaan tietokannan kanssa ja näkymä ladataan uudelleen päivitetyillä tiedoilla. Tällä tavoin ohjelma pysyy jatkuvasti ajan tasalla tietokannan kanssa.

3.1.2 jQuery-kirjasto

jQuery on JavaScriptillä kirjoitettu kirjasto, jolla voi animoida, valita ja muuttaa DOM-elementtien ominaisuuksia. Tämän lisäksi jQuery ja sen käyttöliittymälaajennus sisältävät useita käyttöliittymän luomista helpottavia työkaluja, esimerkiksi kalenterinäkymää käyttävä päivämäärävalitsin, ”vedä ja pudota” -ominaisuus ja syöttöelementtien ulkonäön ylikirjoittaminen.

jQueryn ensimmäinen versio julkaistiin vuonna 2006, ja kirjaston päätarkoituksena oli helpottaa JavaScript-koodin kirjoittamista. Ennen jQueryn tuloa JavaScript-kehittäjien täytyi kirjoittaa suurempi määrä koodia, koska kaikki ominaisuudet eivät toimineet samalla tavalla kaikilla selaimilla. jQuery ratkaisee tämän tarjoamalla valmiita toimintoja, joiden on jo valmiiksi varmistettu toimivan lähestulkoon kaikilla selaimilla [4].

jQueryn asema on kuitenkin joutunut uhatuksi, koska DOM-puuta uudella, tehokkaalla tavalla käsittelevät kirjastot, kuten React ja Googlen Polymer, ovat saaneet vahvempaa jalansijaa verkkokehityksessä. Tämän lisäksi jQueryn laajennukset ja tuettujen selaimien määrä ovat lisänneet kirjaston kokoa huomattavasti ja tehneet siitä monen mielestä turhan kookkaan. [5.]

Tehokasta ja kaunista koodia rakastavat kehittäjät ovat tuoneet saataville jQueryn toimintaa jäljitteleviä kirjastoja, joiden tarkoitus on tarjota samoja ominaisuuksia, mutta keveämmin ja pienemmässä koossa. Kenties reaktiona tähän kesällä 2016 julkaistu jQueryn kolmas versio on vähentänyt tukemiensa selaimien määrää ja optimoinut kirjaston toimintaa. [6.]

3.1.3 JSON-tiedostomuoto

JSON eli JavaScript Object Notation on kevyt tiedonvälitysformaatti, joka mahdollistaa tiedon siirtämisen avain-arvopareja käyttäen. JSON perustuu JavaScriptin object-tietuemuuttujaan, mutta tästä huolimatta tieto siirretään luettavassa muodossa, mikä tekee JSON:sta kieliriippumattoman formaatin. Tämän takia JSON-jäsennin on lähestulkoon jokaisessa modernissa ohjelmointikielessä.

JSON syntyi 2000-luvun alussa tarpeesta kehittää tilapohjainen tiedonsiirtoformaatti tiedon siirtämiseen selaimen ja palvelimen välillä. Aikaisemmin tähän tarkoitukseen oli käytetty selainlaajennuksia, kuten Adoben Flashia tai Java-appletteja, mutta niistä haluttiin eroon käytön kömpelyyden takia. [7.]

Kun AJAX (Asynchronous JavaScript and XML) alkoi yleistyä tapana välittää tietoa selaimen ja palvelimen välillä, JSON ylitti nopeasti suosiossa XML:n, eli aikaisemman samaan tarkoitukseen käytetyn formaatin. XML:ään verrattuna JSON:n etuna on sen yksinkertaisuus ja helppolukuisuus (koodiesimerkki 1). [8.]

```
{
”id”: 12,
”name”: ”John Smith”,
”age”, null,
”email”: ”email@mail.com”,
records: [
{”artist”: ”Pink Floyd”, ”album”: ”Meddle”, ”available”:
false},
{”artist”: ”Tangerine Dream”], ”album”: ”Stratosfear”,
”available”: true}
]
}
```

Koodiesimerkki 1. Esimerkki JSON-tietueesta.

Kuten esimerkistä voi huomata, tieto merkitään erittäin tiiviisti ja tietotyypit määritellään yksinkertaisesti merkintätapaa vaihtamalla. JSON:n tukemia tietotyyppejä ovat

- numero
- teksti
- Boolean eli true tai false
- järjestetty lista (Array), jota merkitään hakasulkeilla
- järjestämätön tietue (Object), joka sisältää sekalaisia avain-arvopareja
- null, eli tyhjä arvo.

Useimmat nykyaikaiset verkkosivut ja -palvelut käyttävät JSON:a tiedonsiirtoformaattina käyttöliittymän ja palvelimen välillä. Koska JSON sopii luettavissa olevan tiedon tallentamiseen, sitä käytetään ohjelmistojen asetustietojen tallennusformaattina.

3.2 Palvelinpuolen rakentamiseen käytetyt arkkitehtuurit ja kirjastot

3.2.1 RESTful-rajapinta

RESTful (Representational State Transfer) on arkkitehtuurimalli, jonka ajatuksena on lisätä ohjelmointirajapintojen suorituskykyä, skaalautuvuutta, luotettavuutta ja muokattavuutta. RESTful-palvelut viestivät HTTP:n kautta käyttäen samoja metodeja kuin mitä selaimet käyttävät verkkosivujen avaamiseen ja tietojen lähettämiseen mm. lomakkeiden kautta. [9, s. 13.]

REST-palvelun rajapintana käytetään tavallisia URI-osoitteita, joiden toiminnot muuttuvat sen mukaan millä HTTP–metodilla niitä kutsutaan. Yleisimpiä metodeja ovat GET, POST, PUT ja DELETE, jotka jo itsessään tarjoavat puitteet kaikkiin tarvittaviin toimintoihin ohjelmassa. RESTful-rajapinta yleensä tarjoaa mahdollisuuden luoda, muokata ja poistaa tietoa joko tietuekohtaisesti tai kokoelmakohtaisesti useita tietueita kerrallaan (taulukko 2). Useimmissa tapauksissa tieto välitetään rajapintaan JSON-muodossa, ja vastauksena kyselyyn annetaan yleensä haettu tai lisätty tietue (tai lista tietueita), vahvistus toiminnon onnistumisesta tai virheviesti epäonnistumisesta. [9, s. 14.]

Taulukko 2. RESTful-rajapinnan käyttötapauksia eri HTTP-metodeita käyttäen

GET	**POST**	**PUT**	**DELETE**
/users Palauttaa listan tietueita	**/users** Lähettää tietueen, joka lisätään tietokantaan	**/users** Olemassa oleva tietuelista korvataan lähetetyllä	**/users** Poistetaan koko tietuelista.
/users/id Palauttaa id-tunnisteen mukaisen tietueen	**/users/id** Ei yleisesti käytössä	**/users/id** Id-tunnisteen mukainen tietue korvataan lähetetyllä, jos sitä ei ole olemassa, luodaan uusi.	**/users/id** Id-tunnisteen mukainen tietue poistetaan

Jotta rajapinta voidaan luokitella puhtaasti RESTful-arkkitehtuuria noudattavaksi, sen pitää täyttää tietyt rajoitteet [10]. Nämä rajoitteet ovat kuitenkin puhtaasti suuntaa-antavia, eikä niissä määritellä tarkasti, miten rajapinta pitäisi teknisesti toteuttaa [11].

Rajoitteet ovat seuraavat:

- **Yhdenmukainen käyttöliittymä**
  URI-osoitteiden täytyy olla selkeästi nimettyjä ja rajapinnan tulee tunnistaa GET/POST/PUT/DELETE-metodit.

- **Asiakas–palvelin**
  Asiakkaan eli selaimen pitää olla täysin irrallinen palvelimesta. Asiakasohjelman tulee tietää ainoastaan rajapinnan URI-osoitteet.

- **Tilattomuus**
  Rajapinta on tilaton, eli se käsittelee jokaisen kyselyn tai toiminnon uutena, tallentamatta mitään käyttöhistoriaa. Jos tiloja halutaan käyttää, ne täytyy määritellä asiakasohjelmaan.

- **Välimuisti**
  Rajapinnan pitää määritellä, halutaanko sen tarjoama tieto tallentaa välimuistiin vai ei.

- **Monikerroksinen järjestelmä**
  REST mahdollistaa järjestelmän resurssien hajauttamisen: rajapintaa voidaan pitää yllä kuormatasatussa osoitteessa ja käyttöliittymää sekä verkko-osoitteessa että mobiilisovelluksessa.

RESTful-arkkitehtuurin eduksi voidaan laskea myös se, että sillä rakennetut palvelut on hyvin helppo saada toimimaan millä tahansa laitteella tai järjestelmällä. Asiakkaan täytyy vain saada pääsy rajapintaan ja tietää, mitä tietoa syötetään ja mitä saadaan takaisin. Tällä tavalla sama rajapinta voi tarjota palveluita mobiilisovelluksille, verkko-

ohjelmistoille ja kolmannen osapuolen järjestelmille. Esimerkiksi Twitter tarjoaa pääsyn julkiseen rajapintaansa [12], jota käyttäen Twitterin ominaisuudet voi ottaa käyttöön vaikka omassa ohjelmassa.

3.2.2 Slim Framework -ohjelmointikehys

Slim Framework on PHP:llä kirjoitettu ohjelmistokehys, joka mahdollistaa rajapintojen ja verkkosovellusten kirjoittamisen suhteellisen pienellä vaivalla. Sen etuna on keveys, ja koska se on kirjoitettu PHP:llä, sitä voi tarpeen tullen laajentaa muilla PHP:llä kirjoitetuilla moduuleilla. Slim on joustava, ja sillä voi helposti rakentaa prototyyppejä palveluista tai vaikka kokonaisia verkkosovelluksia [13].

Slim Frameworkin ytimessä on HTTP-reititin, jolla voi suoraan luoda RESTful-ohjelmointirajapintojen vaatimia HTTP-metodeita ja URI-osoitteita käyttäviä toimintoja. Tämä lisäksi Slim tarjoaa pohjienhallintajärjestelmän, jota käyttäen pystytään luomaan rajapintojen lisäksi kokonaisia verkkosivustoja. SlimPHP:tä käytettäessä luodaan ensin uusi instanssi itse Slim-ohjelmasta, johon luodaan vastakutsufunktioita halutuille URI-osoitteille ja HTTP-metodeille.

Kun insinöörityöprojektin alkuvaihteessa tutkittiin vaihtoehtoja PHP-pohjaiselle RESTful-ohjelmistokehykselle, suurin ongelma oli löytää kehys, jonka toiminnot olisivat selkeästi dokumentoituina. Slim Frameworkin ominaisuudet selitetään projektin verkkosivuilla sangen kattavasti, ja verkosta löytyy runsaasti ohjelmointiesimerkkejä eri toimintojen kehittämiseen.

3.2.3 PDO-laajennus

PHP Data Objects eli PDO on PHP:n laajennus, joka mahdollistaa kevyen ja yhdenmukaisen tavan luoda yhteys PHP:n ja tietokannan välille [14]. PDO:n eduiksi voidaan lukea seuraavat ominaisuudet:

- oliopohjaisuus
- turvallisuus
- try-catch-periaatteella toimiva virheenhallinta
- eri tietokantoihin yhdistäminen toimii samalla tavalla.

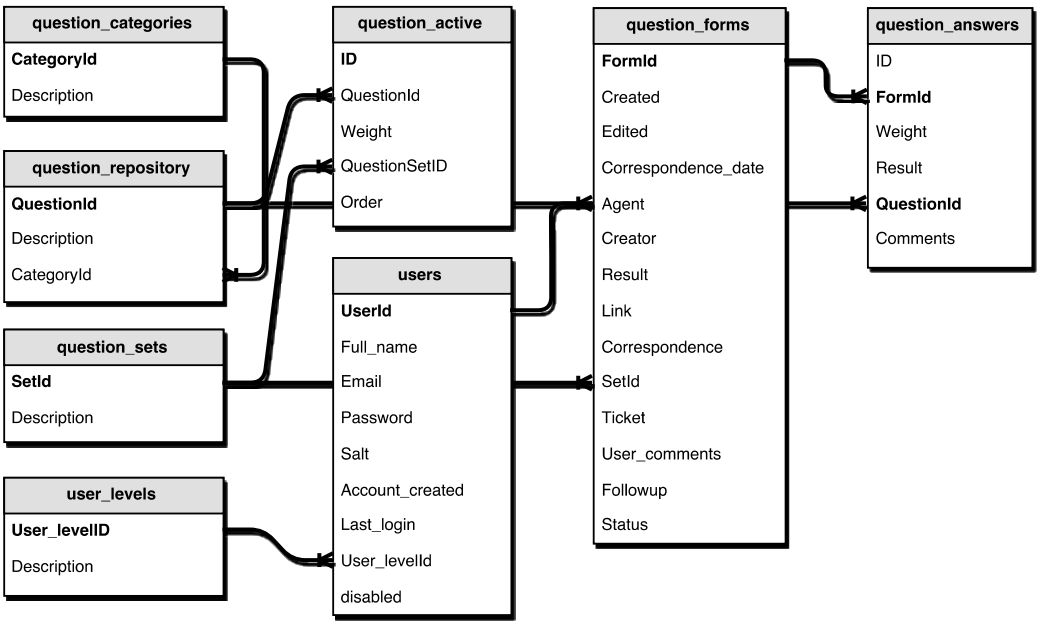
Vaikka PDO:lla muodostetaan yhteys haluttuun tietokantaan kirjaston omia metodeja käyttäen, se ei kuitenkaan sisällä eri tietokantojen syntaksia tulkitsevia metodeja. Tämän vuoksi sitä ei pidä sekoittaa ORM:iin (Object Relational Model), jota käytetään varsinkin Python-kehityksessä (Django) ja itsenäisenä laajennuksena (SQLAlchemy). Vaikka myös PHP:lle on olemassa ORM-periaatteella toimivia laajennuksia, tässä projektissa niitä ei kuitenkaan päädytty käyttämään. [15.]

4 Ohjelmiston tietokanta

Kun ohjelmiston tietokantaa lähdettiin suunnittelemaan, johtoajatuksina olivat tallennettujen tietojen eheys, tehokkuus ja yksinkertaisuus. Niiden pohjalta luotiin ensimmäinen versio tietokannasta käyttäen seuraavia periaatteita: kaiken tiedon pitää olla tietokannassa ainoastaan yhden kerran, tiedot, jotka toistuvat useissa tauluissa, ovat viiteavaimia (foreign key) ja niiden väliset suhteet ovat aina one-to-many-tietomalleilla. One-to-many tarkoittaa käytännössä sitä, että äiti-elementillä voi olla useita lapsielementtejä, mutta kaikilla lapsilla ainoastaan yksi äiti-elementti. [16, s. 4.]

4.1 Tietokannan rakenne

Tietokantaa lähdettiin alun perin kehittämään samaan aikaan kuin ohjelmiston muita osia, ja molempiin tehtiin muutoksia aina tarpeen tullen. Tietokannan rakenne ei siis ollut suoraan tiedossa, eli kerralla keskityttiin vain yhteen osaan toiminnoista ja tietokantaan luotiin uusia tauluja ja kenttiä tarpeen vaatiessa. Kun yksi ohjelman kokonaisuuksista saatiin valmiiksi, tietokannan rakennetta tarkasteltiin ja muokattiin, jotta se täyttäisi ennalta asetetut vaatimukset (kuvio 4).



Kuvio 4. Ohjelmiston rakenne ja taulujen väliset suhteet.

Ainoa taulu, joka sisältää duplikaattitietoa, on ”question_answers”-taulu, johon kysymyslomakkeen vastaukset tallennetaan. Jos kysymyksien senhetkisiä tietoja ei tallennettaisi tauluun, kaikki vanhatkin arvioinnit muuttuisivat, jos arvioija päättäisi muuttaa kysymyksien painoarvoja jälkeenpäin. Tämä mahdollistaa siis saman kysymyslomakkeen käyttämisen uusilla kysymyksillä ilman, että muutokset vaikuttaisivat aikaisempiin arvioihin.

”Question_answers” on myös ainoa taulu, jossa oli pakko käyttää yhdistelmäavainta, koska arviointilomakekohtaiset vastaukset noudetaan ”FormId”-viiteavainta käyttäen ja tarkempi tunniste on itse kysymyksen viiteavain eli ”QuestionId”.

4.2 Viiteavaimet ja rajoitteet

Seurauksena viiteavaimien käyttämisestä tietokannan täytyy noudattaa hierarkkista arkkitehtuuria, mikä tarkoittaa sitä, että kysymystä ei voi luoda ennen kategoriaa eikä kategoriaa voi poistaa, ennen kuin kaikki siihen liitetyt kysymykset on poistettu. Viiteeheyden ylläpitämiseksi kysymystä ei voi luoda ilman kategoriaa eikä kysymystä voida poistaa, jos se on määriteltynä yhdessäkin kyselylomakkeessa.

Viiteavaimia ja rajoitteita (constraints) käyttäen pystyttiin luomaan tarvittavat rajoitukset liittyen tietojen lisäämiseen ja poistamiseen. Tästä esimerkkinä voidaan käyttää sitä, että kysymystä ei voida luoda ”question_repository”-tauluun, ennen kuin itse kategoria on luotu, koska taulun tuple sisältää pakollisen viiteavaimen ”question_categories”-tauluun. Vastaavasti kun kysymyksiä sisältävää kategoriaa yritetään poistaa, viiteavaimen ”ON DELETE NO ACTION” -rajoite estää rivin poistamisen, jos kyseiseen kategoriaan viittaavia kysymyksiä on jäljellä. [17.]

Poistamisen estämisen lisäksi tietokantayhteys palauttaa tapahtumasta virheen, joka voidaan PDO:n virheenkäsittelyä käyttäen lähettää suoraan käyttöliittymään. Virheenkäsittelyn ansiosta tietojen poistamiseen liittyvää logiikkaa ei tarvitse välttämättä koodata itse käyttöliittymään, vaan se voidaan hoitaa täysin käyttämällä itse tietokannasta tulevia virhe- ja onnistumisviestejä.

Täysin päinvastaista toimintoa käytettiin ”question_answers”-taulussa, joka sisältää kaikki arviointilomakkeen kautta vastatut kysymykset ja niiden tiedot. Kun arviointilo-

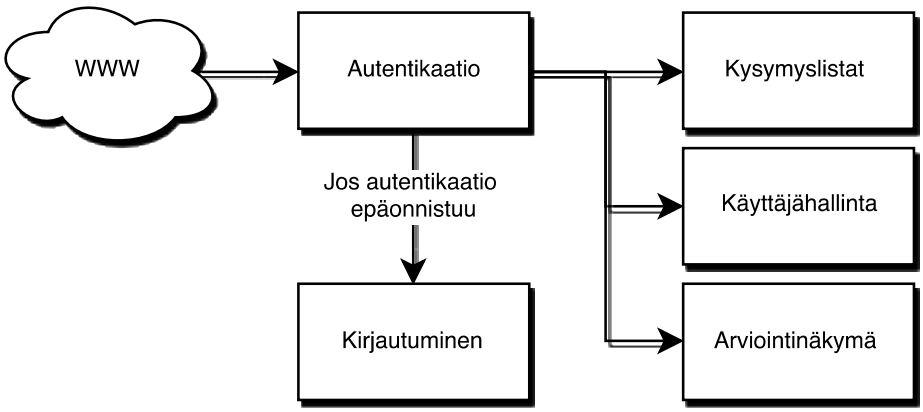
make poistetaan, käytetään CASCADE-ominaisuutta, joka suorittaa äiti-taulussa tehdyt muutokset lapsi-tauluun – eli jos arviointilomake poistetaan, samalla myös kaikki lomakkeeseen viittaavat vastaukset poistetaan. Tietokannan viite-eheys säilyy, kun kaikki taulut ovat yhteydessä toisiinsa eli tieto ei pääse katoamaan näkyvistä eikä orpoja lapsi-elementtejä pääse syntymään.

5 Ohjelmiston kehittäminen

Laaduvarmistusohjelmiston kehittäminen aloitettiin Backbonea ja JavaScriptiä käyttäen ilman rajapintaa, eli kaikki tieto tallennettiin ainoastaan selaimen muistiin. Nopeasti kuitenkin kävi ilmi, että käyttöliittymän ja palvelinpuolen kehittämisen täytyi kulkea samaa tahtia. Tämän lisäksi täytyi kehittää toimiva ja turvallinen tapa hallinnoida sisään- ja uloskirjautumista sekä keino esittää eri käyttäjätasojen näkymät ja sallitut toiminnot.

Palvelimen puolella kyseessä ei siis ole pelkkä rajapinta, vaan ohjelmiston toimintamalli ja se, miten eri osat näytetään yhdessä. Näkymien esittämiseen päätettiin käyttää Slim Frameworkin omia työkaluja ja luoda pohjat jokaiselle sivulle erikseen. Kirjautumistietojen tallentamiseen käytettiin PHP:n omaa sessionhallintaa sen toimintavarmuuden ja yksinkertaisuuden vuoksi.

Kun käyttäjä saapuu ohjelmistosivulle, hänelle tarjotaan heti suoraan kirjautumisnäkymä. Ennen jokaisen toiminnon käynnistämistä tarkistetaan, onko käyttäjällä aktiivinen sessio palvelimella, ja jos tarkistus epäonnistuu, käyttäjä ohjataan takaisin kirjautumisnäkymään (kuvio 5).



Kuvio 5. Arvioijanäkymän toimintaperiaate palvelimella

Jokaisella sivupohjalla on omat toimintonsa, jotka on kirjoitettu Backbonea ja JavaScriptiä käyttäen. Jokainen sivu on itsenäinen osa, jonka sisällä voidaan liikkua selainhistoriassa ilman, että näkymää virkistetään perinteisellä tavalla.

Jotta säästyttäisiin versiointiin liittyviltä ongelmilta, versionhallintaan käytettiin GIT:ä ja Bitbucketin ilmaista palvelua. GIT:ä käyttäen pystyttiin seuraamaan tehtyjä muutoksia, ja Bitbucketin tiketöintijärjestelmä mahdollisti ongelmien ja lisäominaisuuksien hallinnan.

5.1 Palvelinpuolen rakentaminen

Kehitysympäristönä päätettiin käyttää tietokonetta, johon asennettiin tarvittavat palvelimet, ohjelmat ja ominaisuudet Chris Mallinsonin ohjeita seuraten [18]. Tarkoituksena oli siis kehittää ohjelmistoa paikallisesti ja toteuttaa se niin, että tiedostot ja tietokanta voidaan vain kopioida toiselle palvelimelle ja saada toimintakuntoon mahdollisimman pienillä muutoksilla. Tämä toimintatapa ja ohjelmiston modulaarisuus mahdollistivat sen, että ohjelmistoon pystyttiin tekemään suuriakin muutoksia muokkaamalla ainoastaan muutamaa tiedostoa.

CSS-tiedostoille on oma kansionsa, ja kaikki JavaScriptiin liittyvä, käytetyt kirjastot ja laajennukset ovat JS-kansion sisällä. Koko ohjelmisto käynnistyy lukemalla juuressa olevan ”index.php”-tiedoston, joka käynnistää Slim-instanssin ja sisältää kaikki rajapinnan toiminnot.

Jotta kaikkia toimintoja ei tarvitsisi aina lukea muistiin, index.php-tiedosto päätettiin pilkkoa eri tiedostoihin niiden käyttötarkoituksen mukaan. Templates-kansio sisältää sivupohjat, jotka ladataan käyttäjän mennessä esimerkiksi arviointinäkymään. Sivupohjat ovat staattisessa HTML-muodossa, mutta niiden sisältö tuotetaan JS-kansiossa olevia JavaScript-kirjastoja ja laajennuksia käyttäen. Sivupohjien lisäksi kansiossa on otsikkotiedosto, joka pitää sisällään ylimmän tason käyttöliittymän eli sivunavigoinnin, uloskirjautumisen ja sivuston yleisen ilmeen.

Kun käyttäjä avaa ”/users”-osoitteen GET-komentoa käyttäen, tarkastetaan ensin, onko käyttäjä kirjautunut sisään, ja jos näin on, ajetaan funktio, joka tarkistaa, onko käyttäjätaso tarpeeksi korkea sivun näyttämiseen. Jos tämäkin tarkistus onnistuu, näytetään käyttäjänhallinnan sivupohja käyttäjälle.

Rajapinnan toiminta ei eroa paljon edellä mainitusta. Ainoana erona tietenkin se, että tietoa haetaan, lisätään, muokataan tai poistetaan tietokannasta ja vastaus annetaan JSON-muodossa.

5.1.1 Rajapinnan rakentaminen

Rajapinnan rakentaminen aloitettiin miettimällä, mitä toimintoja tarvitaan, jotta käyttöliittymä ja ominaisuudet toimisivat halutulla tavalla. Esimerkkinä kategoria, jonka käsittely rajoittuu käyttöliittymässä lukemiseen, valitsemiseen, luomiseen ja poistamiseen. Edellä mainituista valinta on palvelimen näkökulmasta turha ja liittyy enemmänkin kysymyksien lisäämiseen, eli ainoat tarvittavat toiminnot ovat luku, lisäys ja poistaminen.

Ensimmäinen rajapintakokonaisuus, kysymyslistojen luomisnäkymä, koostuu varsinaisesti neljästä eri osasta: kategoriat, kysymykset, kysymyslomakkeet ja niiden sisältämät kysymykset. Kaikkien tietueiden käsittelyyn ei voi käyttää samaa rajapintaa, eli jokaiselle tietueelle pitää luoda oma. Kaikki tietenkin säästeliäästi ainoastaan vaaditut ominaisuudet tarjoten [19].

Kysymyksiin liittyvät rajapinnat päätettiin nimetä seuraavasti:

- kategoriat: /category (GET, CREATE, DELETE)
- kysymykset: /repository (GET, CREATE, DELETE)
- kysymyslistat: /questionset (GET, CREATE, DELETE)
- aktiiviset kysymykset: /questions/id (GET, CREATE, CHANGE, DELETE).

Voidaan huomata, että ainoastaan aktiiviset kysymykset vaativat muokkausominaisuuden. Tämä johtuu siitä, että käyttäjällä täytyy olla mahdollisuus muuttaa kysymyksen painoarvoa ja järjestystä tietokantaan lisäämisen jälkeen. Kaikki edellä mainitut rajapinnat palauttavat lukiessa (GET) listan sisällöstään, ei yksittäisiä tietoja.

Kun ensimmäinen kokonaisuus saatiin valmiiksi, loput rajapinnat pystyttiin luomaan niiden pohjalta nopeaan tahtiin:

- käyttäjänhallinta: /user/id (GET, CREATE, CHANGE, DELETE)
- arviointilomakkeet: /form/id (GET, CREATE, CHANGE, DELETE)

- vastaukset: /answers/id (GET, CREATE, CHANGE, DELETE).

Kun kyseessä on tieto, joka haetaan tietokannasta käsittelyä varten, rajapintaan täytyy myös lähettää tunniste, jolla haluttu tieto löydetään tietokannasta. Tämän lisäksi rajapintaan täytyy lisätä mahdollisuus lähettää lisätietoja parametreja käyttäen, esimerkiksi tulosten sivuttamista varten.

Tunnisteiden käyttö toimii niin, että ”/form” palauttaa listan kaikista lomakkeista, mutta ei tarkempia tietoja, ”/form/12” taas palauttaa tunnisteella 12 olevan lomakkeen kaikki tiedot, jotta niitä voidaan muokata ja tarkastella. Koodiesimerkissä 2 näytetään käyttäjän poistaminen tunnistetta käyttäen.

```
$app->delete('/user/:id', $authenticate($app),
function($id) use ($app){
    if($_SESSION['user_level']<=2)  {
        $sql = "DELETE FROM users WHERE UserId=:id";
         try {
            $db = getConnection();
            $stmt = $db->prepare($sql);
            $stmt->bindParam("id", $id);
            $stmt->execute();
            $db = null;
        } catch(PDOException $e) {
            $app->halt(400, $e->getMessage());
        }
        echo('true');
    }   else  {
        $app->halt(401);
    }
});
```

Koodiesimerkki 2. Esimerkki käyttäjän poistamisesta tunnistetta käyttäen.

Poikkeuksena tästä toimintatavasta on, että käyttöliittymään tehtiin toiminto, joka lähettää kaikki kysymyslistojen vastaukset rajapintaan muutoksena (CHANGE). Tämä tarkoittaa rajapinnassa sitä, että ”/answer/13”-komentoa käyttäen tarkastetaan ensin, onko tietokannassa vastausta kysymykseen 13. Jos vastaus löytyy, tiedot päivitetään olemassa olevaan riviin, ja jos vastaus puuttuu, luodaan uusi rivi tietokantaan annetuilla tiedoilla.

Toinen poikkeus on arviointilomakkeiden lukutoiminto. Toiminnon vaatimuksena oli tarjota useampia sisäänrakennettuja aikasuodattimia ja vapaasti valittavat päivämäärät, joiden väliltä tiedot tulostettaisiin. Näiden tietojen välittämiseen päätettiin käyttää parametreja, jotka lähetetään GET-pyynnön mukana.

5.1.2 Turvallisuus, kirjautuminen ja sessionkäsittely

Laadunvarmistusohjelmisto sisältää suhteellisen arkaluontoista materiaalia, kuten käyttäjien tietoja ja yrityksen ja asiakkaan välistä kirjeenvaihtoa, joten turvallisuuteen päätettiin kiinnittää erityistä huomiota. Tietoturvan ohjenuorana päätettiin käyttää tietoturvan asiantuntijoiden eli hakkerien tarjoamaa salasanojen käsittelyn opasta.

Salasanan tallentamisen ytimessä on tiivistealgoritmi, joka käytännössä tarkoittaa yksisuuntaista kryptaamista. Tällä tavoin esimerkiksi salasanasta keppihevonen saadaan sha-1-tiivistealgoritmilla tulokseksi b60331e508e31001568bb8944037583b9e415528. Koska tiiviste on aina yksisuuntainen, alkuperäisen salasanan selvittäminen voi olla vaikeaa, mutta tällaisenaan tuskin mahdotonta. Ongelmana toimintatavassa on myös se, että jos kahdella käyttäjällä sattuu olemaan sama salasana, myös tiiviste on sama.

Toinen elementti turvallisen salasanan luomisessa on suolaus, joka poistaa edellä mainitun ongelman. Nimen mukaisesti kyseessä on varsinaiseen salasanaan lisättävä osa, usein satunnaisesti luotu merkkijono. Tämä merkkijono yhdistetään salasanaan esimerkiksi seuraavalla tavalla: AVjwhW2vubrrmJHxkeppihevonen. Tuloksena syntyvä merkkijono tiivistetään halutulla tiivistealgoritmilla ja tallennetaan tietokantaan salasanana. [20.]

Jotta vältyttäisiin samankaltaisilta tiivisteiltä, suola on luotava uudelleen jokaisen salasanan luonnin yhteydessä ja tallennettava tietokantaan salasanasta luodun tiivisteen mukana. Salasanaa vaihdettaessa myös suola luodaan uudelleen ja tallennetaan tietokantaan, jotta salasanan tiiviste pysyisi mahdollisimman turvallisena. Tiivistealgoritmina päädyttiin käyttämään PHP:n crypt-funktiota ja suolana mcrypt_create_iv-funktiota, joiden synergiasta syntyy suhteellisen väkevästi kryptattu tiiviste [21 ; 22].

Käyttäjän kirjautuessa käyttöliittymään tietokannasta haetaan ensin käyttäjätunnuksen mukainen suola, joka liitetään salasanaan, tiivistetään ja verrataan tietokannassa olevaan salasanatiivisteeseen. Jos tiivisteet ovat identtisiä, luodaan PHP:llä sessio, johon tallennetaan käyttäjän tietoja, muun muassa nimi ja käyttäjätaso. Jokaisen tietokantatoiminnon yhteydessä tarkistetaan, onko käyttäjällä olemassa olevaa sessiota, ja jos sessio puuttuu, käyttäjä ohjataan kirjautumissivulle.

Sessio eroaa evästeestä sillä, että tiedot tallennetaan käyttäjän koneen sijaan palvelimelle. Kun sessio aloitetaan, luodaan satunnainen merkkijono, jota käytetään session

tunnisteena. Session tunnisteen pitää olla sama sekä käyttäjällä että palvelimella, muuten sessio suljetaan. On mahdollista kaapata sessio selvittämällä olemassa oleva sessiotunniste, mutta tämän voi helposti välttää luomalla tunniste uudelleen tietyin aikavälein.

Tunnisteen muuttaminen jokaisen rajapintatoiminnon yhteydessä aiheutti sen, että sessio ei pysynyt enää mukana vauhdissa ja seurauksena oli uloskirjautuminen. Mutkattomimpana ratkaisuna oli siis uusia sessiotunniste ainoastaan, kun sivu vaihdetaan tai ladataan uudelleen.

5.2 Käyttöliittymä

Kun kyseessä on ohjelmisto, jossa on kolme erilaista näkymää, on tärkeä päättää jo alussa, minkälaista ulkoasua ja toimintalogiikkaa käytetään. Näkymien yksinkertaistamista ajatellen päätettiin, että kaikki muokkausnäkymät avautuvat koko ruudun kokoisiksi listanäkymän jäädessä taustalle. Tällöin sivu ladataan kerran näkyviin, eikä mikään toiminto pakota sivua virkistymään. Sivulta toiselle liikutaan sivun ylälaidassa olevan navigaatiopalkin linkkejä napauttamalla, ja ohjelman sisällä liikkumisen helpottamiseksi navigaatiopalkki jää kellumaan sivun ylälaitaan sivua alas vierittäessä.

Listanäkymän yläpuolelle sijoitettiin ”lisää”- ja ”poista”-napit, joista ”lisää” avaa lisäysnäkymän ja ”poista” aktivoi poisto-ominaisuuden, minkä jälkeen listalta voidaan poistaa halutut rivit napauttamalla niitä. Tavoitteena oli myös saada kaikki toiminnot sellaisiksi, että näppäimistöön ei tarvitse koskea kuin tekstin syöttövaiheessa, eli käyttöliittymässä käytettiin runsaasti pudotusvalikkoja ja jQueryn käyttöliittymäelementtejä, kuten päivämääränvalitsinta ja ”raahaa ja pudota” -ominaisuutta.

Kuten luvun 5.1 alussa kuvailtiin, sivupohjat lataavat tyylitiedostot, Backbonen, kaikki tarvittavat kirjastot ja niitä käyttävät laajennukset, eli varsinaisen ohjelman. Selkeyden takia ohjelma on jaettu neljään eri osaan: malleihin, kokoelmiin, näkymiin ja ohjelman käynnistävään reitittimeen. Erillistä käynnistysskriptiä ei tarvitse luoda, koska selain rakentaa yhtenäisen kokonaisuuden linkitetyistä resursseista. Kun sivu on latautunut, ohjelman reititin suorittaa sisältösivulle määritellyt toiminnot eli alustaa kaikki tietueet ja pohjanäkymän.

5.2.1 Kysymyslistojen luomisnäkymä

Käyttöliittymän kehittäminen aloitettiin kysymyslistojen luomisnäkymästä, koska ohjelman muut osat ovat täysin riippuvaisia siitä. Näkymä päätettiin jakaa kolmeen osaan, kategorioihin, kysymyksiin ja kysymyslistoihin.

Käyttöliittymän elementtien toimintojen hallintaan päätettiin käyttää Backbonen omaa tapahtumienhallintaa, mikä mahdollisti elementtien sitomisen haluttuihin funktioihin. Elementit voidaan määritellä elementin tyypillä, luokalla tai tunnisteella. [3, s. 55–61.]

Koska kysymyksen on pakko kuulua johonkin kategoriaan, sellainen pitää ensin luoda tai valita. Valitun kategorian tunniste tallennetaan globaaliin muuttujaan, jotta jokaiseen luotuun kysymykseen voidaan lisätä kategoriaviite. Tapahtumafunktioon tuodaan aina mukana käytetyn elementin tiedot muuttujana, jonka sisältämiä tietoja voidaan käyttää hyväksi.

Kysymyslistat sijoitettiin allekkain sivun oikealle puolelle. Napauttamalla listan otsikkoa sen sisältämät kysymykset avautuvat näkyviin ja listan tunniste tallennetaan globaaliksi muuttujaksi. Kysymykset lisätään listaan ”raahaa ja pudota” -periaatteella, ja lisättävän kysymyksen HTML-elementti sisältää viitetunnisteen, jota käytetään listakokoelmaan lisättäessä.

Kysymysten painoarvo määritellään vetämällä hiirellä kysymyksen alareunan tartuntapalkkia alaspäin. Vedetyn etäisyyden mukaan painoarvon voi määritellä välille 0–50 %. Tartuntapalkki venyttää kysymyselementtiä alaspäin ja palauttaa sen normaaliksi, kun haluttu painoarvo on valittu (koodiesimerkki 3). Tiedon muuttuessa painoarvo tallennetaan tietueeseen ja päivitetään suoraan tietokantaan.

```
resize: function(event, ui) {
var weight = (ui.size.height-30)/2;
$( '#weight-'+$(this).attr('id') ).val(weight);
},
stop: function(event, ui)        {
var id = parseInt($(this).attr('id'));
activeQuestions.updateWeight(id, (ui.size.height-30)/2);
$(this).animate( {height: "30px"}, 100 );
}
```

Koodiesimerkki 3. Painoarvon määrittelemiseen käytetyt resize- ja stop-funktiot.

Kysymysten järjestystä voi muuttaa tarttumalla kysymykseen hiirellä ja liikuttamalla listaelementtiä listalla ylös tai alas. Kun käyttäjä vapauttaa hiiren napin, kysymyslistasta muodostetaan taulu, joka sisältää kysymysten tunnisteet järjestyksessä. Taulu lähetetään palvelimelle rajapintaan, jossa tiedot päivitetään tietokantaan joukolla update-komentoja.

5.2.2 Käyttäjänhallinta

Käyttäjienhallinta oli kaikkein helpoin osa toteuttaa, koska se vaati toimiakseen ainoastaan yhden mallin, kokoelman ja näkymän. Käyttäjienhallinta koostuu kahdesta eri näkymästä: käyttäjälistasta, jossa pystytään tarkastelemaan, poistamaan ja lisäämään käyttäjätilejä, ja käyttäjänmuokkausnäkymästä, jota kautta käyttäjän tietoja voidaan muokata ja tarkastella tarkemmin.

Käyttäjienhallinnan sivun avautuessa rajapinnasta luetaan kaikkien käyttäjien tiedot, jotka tallennetaan users-kokoelmaan. Tämän jälkeen kokoelma liitetään users-näkymään, joka tulostaa tiedot Underscoren sivupohjanhallintaa käyttäen käyttäjälistalle varattuun elementtiin [23]. Koodiesimerkissä 4 on silmukka, joka kirjoittaa kaikki users-kokoelman sisältämät rivit HTML-tauluun. Käyttäjän tunniste tallennetaan rivielementin id-kenttään, josta se voidaan napsautustapahtuman yhteydessä hakea.

```
<% _.each(collection, function(model) { %>
<tr class="<%= model.user_level %> user" id="
<%= model.UserId %>">
<td id="<%= model.UserId %>"><%- model.full_name
%></td><td><%- model.email %></td><td><%- us-
er_levels[model.user_level] %></td><td><%-
model.last_login %></td><td><%- model.account_created
%></td><td><%- disabled[model.disabled] %></td>
</tr>
<% }); %>
```

Koodiesimerkki 4. Underscoren sivupohjahallintaa käyttämällä luotu for-silmukka, joka tulostaa käyttäjien tiedot listaksi.

Jo aikaisessa vaiheessa huomattiin, että vaikka lista onkin suhteellisen helppolukuinen, se voisi olla vieläkin selkeämpi. Näkymässä pitää siis olla mahdollisuus jakaa lista useammalle sivulle, hakuominaisuus käyttäjien hakemiseen ja sarakkeiden järjestämisominaisuus. Tähän tarkoitukseen päätettiin käyttää DataTables-kirjastoa, jonka käyttöönotto oli yksinkertaisimmillaan kirjaston aktivointi haluttuun elementtiin. Nopeasta käyttöönotosta huolimatta ongelmiin törmättiin hakukentän sijoittamisessa, koska se

haluttiin itse tauluelementin ulkopuolelle. DataTables kuitenkin sallii pääsyn haltuun ottamansa elementin tietoihin, joten hakukenttä pystyttiin sijoittamaan haluttuun paikkaan ja saatiin toimimaan moitteetta lisäämällä ylimääräinen funktio, joka välittää kentän sisältämät tiedot tauluelementtiin.

Uusia käyttäjiä voi lisätä painamalla listan yläpuolella olevaa ”lisää”-nappia, joka avaa ruudulle kelluvan lomakkeen, jolle syötetään käyttäjän sähköpostiosoite, nimi ja salasana. Käyttäjien lisääminen voi olla suhteellisen mekaanista toimintaa, mutta salasanojen keksiminen voi olla hankalaa. Salasanoiksi voi siis valikoitua ”kissa123” tai jokin variaatio käyttäjän nimestä, mikä tietenkin on potentiaalinen tietoturvariski.

Paras ratkaisu ongelmaan on, että poistetaan käyttäjältä salasanan keksiminen kokonaan. Salasana-kenttien viereen lisättiin siis nappi, jota painamalla voidaan luoda käyttäjälle satunnainen salasana, joka sijoitetaan suoraan salasanakenttiin. Kun käyttäjän tiedot on täytetty, tiedot lähetetään rajapintaan, jossa käyttäjä lisätään tietokantaan, muodostetaan salasanasta ja suolasta tiiviste ja kaiken onnistuessa lähetetään käyttäjän sähköpostiosoitteeseen tunnus ja tiivistämätön salasana.

Käyttäjien poistamisominaisuus kytketään päälle sivun ylälaidassa olevasta miinusnapista, joka aktivoinnin jälkeen hehkuu vihreänä. Poiston ollessa aktivoituna jokaiseen tauluriviin lisätään taustan punaiseksi muuttava CSS-luokka, joka aktivoituu hiiren ollessa rivin päällä. Poistamisominaisuuden aktiivisuudesta pitää kirjaa näkymän sisäinen boolean-muuttuja ”del”, joka poistonappia napsautettaessa muutetaan aina päinvastaiseksi edellä mainittujen toimintojen lisäksi.

Taulukon riveille määriteltyä napsautustapahtuman käsittelijää päätettiin käyttää myös poistamisominaisuuteen. Riviä napsauttaessa käynnistyvä funktio siis tarkistaa, onko poistamisominaisuus aktivoitu, ja näyttää vaihtoehtoisesti poistovahvistuksen tai suorittaa normaalin toiminnon eli avaa käyttäjän tiedot muokattavaksi.

Käyttäjätietojen muokkausnäkymä avautuu koko sivun kokoiseen elementtiin, joka peittää alleen listanäkymän. Näkymä tulostaa elementtiin rivitunnisteella määritellyn käyttäjän tiedot samaan tapaan kuin käyttäjätiedot tulostettiin listanäkymään. Näkymässä muokattavat syötteet tarkistetaan muuttamisen yhteydessä, ja jos tiedot ovat puutteelliset, ”tallenna”-nappi pysyy toimimattomana.

”Tallenna”-nappia painettaessa kaikkien kenttien arvot lähetetään rajapintaan, jossa ne päivitetään tietokantaan. Jos myös salasana on muutettu, se käsitellään ja tallennetaan tietokantaan. Tämän jälkeen käyttäjälle lähetetään salasanan vaihdosta ilmoittava sähköposti, joka sisältää myös uuden salasanan.

5.2.3 Arviointinäkymä

Arviointinäkymä on ohjelmiston osa, jossa kaikki aikaisemmissa osissa luodut tiedot yhdistetään yhdeksi kokonaisuudeksi eli arviointilomakkeeksi. Arvioijan haluttiin pystyvän täyttämään arviointilomakkeen suoraan kaikilta osin, eli vastaamaan kaikkiin kysymyslistan kysymyksiin ja lisäämään tiketin sisältämät tiedot lomakkeeseen. Tämä aiheutti päänvaivaa, koska vastaukset täytyi tallentaa erilliseen tauluun ja lomakkeen viitetunniste täytyi olla tiedossa ennen tietojen tallentamista.

Ongelma ratkaistiin pakottamalla käyttäjä toimimaan halutulla tavalla. Ensin on valittava arvioitava henkilö, minkä jälkeen vasta voi valita käytettävän kysymyslistan. Kysymyslistan valitseminen poistaa valitsimen käytöstä ja luo tietokantaan uuden arviointilomakkeen, jonka tunnisteen rajapinta palauttaa käyttöliittymään.

Jos käyttäjä valitsee väärän kysymyslistan, ainoa mahdollisuus on sulkea lomakenäkymä ja aloittaa uuden arvioinnin tekeminen. Tästä menettelystä seuraa se, että tietokantaan syntyy useita tyhjiä arviointeja, joihin ei enää pääse käsiksi. Tämän vuoksi lomakkeen luomisen yhteyteen lisättiin ylimääräinen komento, joka automaattisesti poistaa kaikki aikaisemmin luodut tyhjät arviointilomakkeet ja niiden mukana, viiteeheyden säilyttämiseksi, kaikki niihin viitatut vastaukset.

Kysymyslistan ulkoasun selkeyttämiseksi päätettiin käyttää jQueryn UI-kirjaston tarjoamaa button-luokkaa. Se mahdollistaa valintanappien ulkonäön muokkaamisen ilman, että niiden toimintatapa muuttuu millään tavalla. Tällä tavoin saatiin luotua ”yes”-, ”no”- ja ”n/a” -valinnat niin, että vain yksi on valittuna kerrallaan jQueryn huolehtiessa toimintalogiikasta taustalla. Valintanappien lisäksi jokaisen kysymyksen alla on tekstikenttä, johon voi tarvittaessa kommentoida arvioinnin perusteita. Kaikkiin kysymyslistan elementteihin on liitetty muutoksia seuraava metodi, joka päivittää jokaisen päivityksen tietokantaan.

Ohjelman aiemmissa osissa lähes kaikki tiedot ladattiin kerralla muistiin, mutta arviointinäkymän kohdalla tämä ei enää onnistunut. Backbone ja muut JavaScriptillä toimivat ohjelmointikehykset lataavat tiedot asynkronisesti, mikä tarkoittaa sitä, että tiedoista tehdään pyyntö rajapintaan ja tieto käsitellään siinä vaiheessa, kun se saadaan palvelimelta. Ohjelma ei siis yleensä jää odottamaan tietojen saapumista, vaan jatkaa suoritusta välittämättä siitä, onko tietoja saatu vai ei. Tämän seurauksena tietoja jäi satunnaisesti puuttumaan näkymästä ja ohjelman toiminta häiriintyi. Kävi selväksi, että tarvittiin tapa ladata kaikki eri tietueet hallitusti muistiin ennen ohjelman käynnistämistä.

Yksi tähän tarkoitukseen käytetyistä tavoista on tietueiden esilataaminen, eli tiedot liitetään jo sivupohjaa ladattaessa JavaScript-koodin sekaan ja alustetaan muuttujiksi. Tämä on kuitenkin hieman vanhanaikainen ratkaisu, joten ongelma ratkaistiin jatkamalla asynkronisen lataamisen käyttämistä, mutta suunnitellen tarkkaan, miten ja missä järjestyksessä tiedot ladataan. Backbone tarjoaa tähän tarkoitukseen mahdollisuuden käyttää vastakutsufunktioita tietokantatoimintojen yhteydessä: kun lukupyyntö lähetetään, ohjelman suoritusta ei jatketa, ennen kuin se saa vastauksen palvelimelta, ja vastauksesta riippuen jatketaan taas ohjelman toiseen osaan (koodiesimerkki 5). Tällä tavoin rakennettiin sisäkkäisiä tietokantakyselyitä, joiden etuna oli, että tiedot pystyttiin lataamaan kerralla hallitusti muistiin, mikä helpotti virheenkäsittelyä.

```
index: function()     {
    myForms.fetch({data: {dateRange: sDate, customDate1:
$('#customDate1').val(), customDate2: $('#cus-
tomDate2').val()},
    success: function()          {
    myFormView = new FormListView({collection: myForms});
    myFormView.render();
    },
    error: function() {
        promptBox.alert('Unable to initialize');
    }
});}
```

Koodiesimerkki 5. Arviointinäkymän sisällyssivun alustava funktio.

Kun arviointilomaketta täytetään, siihen on myös liitettävä tikettiin liittyvä kirjeenvaihto, mikä tarkoittaa yleensä suhteellisen pitkää sähköpostikeskustelua. Kun täyttämisprosessia käytiin läpi, kävi selväksi, että kirjeenvaihtoon pitäisi pystyä lisäämään muotoiluja tärkeimpien kohtien painottamiseksi. Kenttään siis lisättiin WYSIWYG-periaatteella toimiva muokkausnäkymä, mihin tarkoitukseen valittiin kevyt TinyMCE. Kirjasto liitetään haluttuun elementtiin samaan tapaan kuin DataTables, mutta siitä poiketen se ylikirjoittaa elementin itsellään. Tämän ominaisuus ei kuitenkaan aiheuttanut tyylitiedoston

muokkaamista kummempia toimenpiteitä, joten kirjasto saatiin nopeasti toimintakuntoon ja halutut muokkausominaisuudet näkyviin.

Listanäkymään käytettiin suoraan samoja työkaluja kuin käyttäjänhallinnassa, eli poisto-ominaisuutta pystyttiin käyttämään melkeinpä sellaisenaan, kuten myös muita taulukon ominaisuuksia. Ainoa lisäominaisuus oli lomakkeiden luomispäivämäärän suodatuksen muuttaminen, joka tarjoaa oletuksena yhden viikon, yhden kuukauden tai kolmen kuukauden valinnat ja vapaavalintaisen vaihtoehdon. Viimeisen vaihtoehdon valitseminen avaa näkymään kaksi päivämäärävalitsinta, joiden tiedot lähetetään parametreina rajapintaan GET-pyynnön mukana. Suodatusvalinta ja päivämäärävalitsimien arvot tallennetaan muuttujiin, jotta suodatus ei muuttuisi näkymän elementtien virkistyessä.

Täytettyjen arviointilomakkeiden muokkaustilaan pääsee napauttamalla listanäkymässä haluamaansa riviä. Muokkaustilaa päätettiin käyttää myös yleisenä selausnäkymänä, eli lomakkeita tarkasteltaessa kaikki kentät ovat suoraan muokattavissa, kuten myös kysymyslistan vastaukset. Ainoa lisäkenttä on arvioinnin tila-asetus, jolla arvioija pystyy pitämään kirjaa siitä, onko arvioitava hyväksynyt arvion.

Arvioitavan näkymässä on tarjolla ainoastaan mahdollisuus selata arvioita, jotka on lähetetty hänelle hyväksyttäväksi, sekä loppuun asti käsiteltyjä, jo hyväksyttyjä arviointeja. Näkymä on identtinen verrattuna arvioijan näkymään, mutta poissa ovat mahdollisuudet muokata ja luoda arviointeja sekä antaa arvioinnille muita tiloja kuin ”hyväksytty” tai ”riitautettu”.

6 Yhteenveto

Insinöörityössä toteutettiin verkkopohjainen laadunvarmistustyökalu asiakastukiympäristöä varten. Työn tarkoituksena oli helpottaa arviointien tekemistä siirtämällä työ Excel-pohjista älykkäämpään ja samalla helpommin hallittavaan ympäristöön.

Suurimpana haasteena oli, että työkalu täytyi rakentaa kokonaisuudessaan itse, vaikka työn apuna olikin useita kirjastoja ja ohjelmointikehyksiä. Työmäärä oli suuri: käyttöliittymän ja palvelimen ympäristöt täytyi rakentaa täysin erilaisista elementeistä ja eri ohjelmointikielillä.

Ohjelmiston kehittäminen antoi mahdollisuuden kokeilla uusia tekniikoita ja ratkaisumalleja. Uudenlaisesta lähestymistavasta johtuneen jyrkän oppimiskäyrän vuoksi osa ominaisuuksista täytyi kirjoittaa useaan kertaan uudelleen. Kaikki tämä voidaan kuitenkin laskea positiiviseksi asiaksi, koska toimiviksi ratkaisuiksi miellettyjä toimintatapoja on tämän jälkeen käytetty muissakin asiakasyrityksen projekteissa.

Insinöörityöprojektin aikana myös tekniikka on mennyt eteenpäin, käytettyjä kirjastoja on kehitetty paremmiksi ja tarjolle on tullut uusia vaihtoehtoja. Jos käytetyt työkalut ja kielet täytyisi valita uudelleen, valinta osuisi käyttöliittymäpuolella Googlen Polymeriin, koska siihen on saatavilla suuri määrä suoraan käytettävissä olevia ominaisuuksia, ja palvelinpuolella valinta osuisi todennäköisesti Pythoniin ja Flaskiin tai Djangoon. Tämän lisäksi ohjelmiston käyttöliittymän ja rajapinnan välistä eroa kasvatettaisiin käyttämällä tunnistepohjaista käyttöoikeustarkistusta, mikä mahdollistaisi rajapinnan käyttämisen esimerkiksi kolmannen osapuolen palveluissa.

Työkalu on tällä hetkellä testausvaiheessa, eli muutoksia ja lisäyksiä tehdään tarpeiden mukaan. Tarkoituksena on saada työkalu yleiseen käyttöön lähiaikoina ja lisätä siihen ominaisuuksia ja hioa olemassa olevia toimintoja. Suunnitelmissa on lisätä muun muassa tietojen ulosvienti CSV-muodossa ja tilastonäkymä, jonka kautta pystytään saamaan parempi kuva asiakaspalvelun vahvuuksista ja siitä, mitä osaa tulisi vielä kehittää.

Arvioitavat ovat luonnollisesti olleet vähemmän innostuneita arvioinnin helpottumisesta. Uskoisin kuitenkin, että jos hyvästä työstä annetaan tunnustusta ja sopiva palkinto, mielipide muuttuu heti myönteisempään suuntaan.

Lähteet

1 Usability: Accessibility. 2016. Verkkodokumentti. Google. https://www.google.com/design/spec/usability/accessibility.html. Luettu 15.2.2016.

2 Backbone. 2015. Verkkodokumentti. Backbone. http://backbonejs.org/. Luettu 2.10.2015.

3 Osmani, Addy. 2013. Developing Backbone.js Applications. USA: O'Reilly Media.

4 jQuery: Write less, do more. 2016. Verkkodokumentti. The jQuery foundation. https://jquery.com/. Luettu 15.2.2016.

5 Brady, Zachary. 2014. Replacing jQuery with Vanilla JavaScript. Verkkodokumentti. https://modernweb.com/2014/05/05/replacing-jquery-with-vanilla-javascript/ Luettu 10.2.2016.

6 Willison, Timmy. 2016. jQuery 3.0 Final Released. Verkkodokumentti. The jQuery foundation. https://blog.jquery.com/2016/06/09/jquery-3-0-final-released/ Luettu 12.2.2016.

7 Introducing JSON. 2016. Verkkodokumentti. http://www.json.org Luettu 4.10.2015.

8 Bray, Tim. 2014. The JavaScript Object Notation (JSON) Data Interchange Format. Verkkodokumentti. https://tools.ietf.org/html/rfc7159 Luettu 13.5.2016

9 Richardson, Leonard & Ruby, Sam. 2007. Restful Web Services. USA: O'Reilly Media.

10 What are RESTful Web Services? 2013. Verkkodokumentti. Oracle. http://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html Luettu 10.5.2016.

11 Restful API tutorial. 2016. Verkkodokumentti. RESTfulAPI.net. http://restfulapi.net/ Luettu 10.5.2016.

12 Rest APIs. 2016. Verkkodokumentti. Twitter Inc. https://dev.twitter.com/rest/public Luettu 12.5.2016.

13 Slim a micro framework for PHP. 2016. Verkkodokumentti. Slim Framework Team. http://www.slimframework.com/ Luettu 1.9.2015.

14 PHP Data Objects. Verkkodokumentti. The PHP Group. http://php.net/manual/en/book.pdo.php Luettu 2.9.2015.

15 Wurzer, Erik. 2012. Why you should be using PHP's PDO for Database access. Verkkodokumentti. https://code.tutsplus.com/tutorials/why-you-should-be-using-phps-pdo-for-database-access--net-12059 20.5.2016.

16 Beaulieu, Alan. 2009. Learning SQL. USA: O'Reilly Media.

17 Rajoitteet (CONSTRAINT). 2009. Verkkodokumentti. Jyväskylän yliopiston IT-tiedekunta ja avoin yliopisto. http://appro.mit.jyu.fi/doc/tiedonhallinta/sql/ddl/index3.html Luettu 10.6.2016.

18 Mallinson, Chris. 2016. The perfect web development environment for your new Mac. Verkkodokumentti. https://mallinson.ca/osx-web-development/ Luettu 1.9.2015.

19 Coenraets, Christophe. 2011. RESTful services with jQuery, PHP and the Slim framework. Verkkodokumentti. http://coenraets.org/blog/2011/12/restful-services-with-jquery-php-and-the-slim-framework/ Luettu 2.9.2016.

20 Salted password hashing – doing it right. 2016. Verkkodokumentti. Defuse security. https://crackstation.net/hashing-security.htm Luettu 20.6.2016.

21 Crypt. Verkkodokumentti. The PHP Group. http://php.net/manual/en/function.crypt.php Luettu. 20.6.2016.

22 Mcrypt_create_iv. Verkkodokumentti. The PHP Group. http://php.net/manual/en/function.mcrypt-create-iv.php Luettu 20.6.2016.

23 Using templates in Backbone. Js. Verkkodokumentti. 9bit Studios. http://www.9bitstudios.com/2013/05/using-templates-in-backbone-js/ Luettu 5.9.2015.