Jerker Bäckström

Työkalu SaaS-palvelun käyttöasteen analysointiin

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

6.3.2016

Metropolia

Tekijä Otsikko Sivumäärä Aika	Jerker Bäckström Työkalu SaaS-ohjelmistoasennusten käyttöasteen analysointiin 47 sivua 6.3.2016
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaajat	Lehtori Juha-Pekka Kämäri Tuotekehityspäällikkö Jarkko Lakso

Insinöörityön tavoitteena oli luoda IMS Business Solutions Oy:n käyttöön työkalu, jolla voitaisiin saada hyödyllistä tietoa heidän SaaS-palveluna tarjoamastaan IMS-ohjelmiston käytöstä. Tietoa voitaisiin hyödyntää esimerkiksi IMS-ohjelmiston tuotekehityksessä ja koulutuksessa.

Toteutettavalla työkalulla tuli olla mahdollista suorittaa SQL-kyselyitä SaaS-asennusten tietokantoihin palauttaen asiakaskohtaisia tunnuslukuja, totuusarvoja ja aikasarjoja. Työkalulle toivottiin myös käyttöliittymää, josta voitaisiin hallita työkalun eri toimintoja, kuten kyselyiden ja yhteyksien määrittämistä. Toiveissa oli myös mahdollisuus luoda kerätystä tiedosta taulukoita ja kaavioita.

Työkalu päädyttiin toteuttamaan verkkosovelluksena käyttäen nykyään erittäin suosittua avoimeen lähdekoodiin pohjautuvaa AngularJS-sovelluskehystä, joka on tarkoitettu etenkin SPA-sovellusten luomiseen. Sovellukseen tallennettavien tietojen eli resurssien hallintaa varten päätettiin toteuttaa Spring-sovelluskehyksellä REST-rajapinta, jonka kautta hoituisi asiakas- ja palvelinpuolen välinen kommunikaatio.

Insinöörityön lopputuloksena syntyi työlle asetettuja tavoitteita vastaava, nykyaikaisilla teknologioilla toteutettu verkkosovellus. Sovelluksessa voidaan dynaamisesti määrittää ja hallita suoritettavia kyselyitä, tilastoitavia asiakkaita ja esitettäviä tilastoja. Kerätystä tiedosta voidaan myös vapaasti luoda halutun sisältöisiä ja näköisiä taulukoita sekä useita erityyppisiä kaavioita.

Avainsanat	AngularJS, Spring, REST

Author	Jerker Bäckström
Title	Analyzing Tool for Usage Statistics in SaaS Environment
Number of Pages	47 pages
Date	6 March 2016
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructors	Juha-Pekka Kämäri, Senior Lecturer Jarkko Lakso, Product Developement Manager

The objective of this thesis was to develop a tool for IMS Business Solution Oy that would help to gather usage statistics about their IMS software. IMS software is provided to clients using the SaaS delivery method and the gathered information could be used for example to improve product development, marketing and training.

The requirement of the tool included the ability to execute SQL queries to SaaS clients' databases which would return statistics, truth values and time series. It was wished for the tool to have a user interface from where it would be possible to manage different features of the tool, such as adding queries and managing connections. A possibility to create different kinds of charts from the gathered statistics was also hoped for.

The tool was decided to be developed as a web application with an open source JavaScript framework called AngularJS which is very popular these days and is especially used for creating SPA applications. For the data management of the application and communication between frontend and backend it was decided to implement a REST API using the Spring framework.

As a result of this thesis, a modern web application was created that meets the requirements introduced by IMS Business Solutions Oy. With the application it is possible to dynamically define and manage the executed queries, connections and presented statistics. From the gathered data it is also possible to freely create desired data tables and a variety of charts.

Keywords	AngularJS, Spring, REST

Sisällys

Lyhenteet

API	Application Programming Interface. Ohjelmointirajapinta.
AOP	Aspect-oriented programming. Aspektiperustainen ohjelmistokehitys.
CRUD	Create, Read, Update, Delete. Luonti, luku, päivitys ja poisto.
CSS	Cascading Style Sheets. Merkintäkielien tyylien kuvaamiseen tarkoitettu tyyliohjekieli.
CSV	Comma-separated values. Tiedontallennukseen tarkoitettu yksinkertainen tiedostomuoto.
DOM	Document Object Model. Tapa kuvata ja käsitellä dokumentin puurakennetta objekteina.
HTML	Hypertext Markup Language. Hypertekstin merkintäkieli, jota käytetään yleensä verkkosivujen ja dokumenttien luomiseen.
HTTP	Hypertext Transfer Protocol. Protokolla tiedonsiirtoon verkossa.
IoC	Inversion of Control. Suunnittelumalli, jossa eri komponenttien välisten suhteiden luonti ja hallinta ulkoistetaan.
Java EE	Java Enterprise Edition. Kokoelma Java-teknologioita yrityksille suunnattujen suurikokoisten sovellusten tekemiseen.
JDBC	Tietokantayhteyksien toteuttamiseen tarkoitettu ohjelmointirajapinta Java-ohjelmointikielelle.
JPA	Java Persistence API. Java-spesifikaatio tiedonhallintaan Java-objektien ja relaatiotietokantojen välillä.
JSON	JavaScript Object Notation. Tiedonvälitykseen tarkoitettu avoimen standardin tiedostomuoto.

MariaDB	Relaatietokantajärjestelmä.
MySQL	Relaatiotietokantaohjelmisto.
MVC	Model-View-Controller. Ohjelmistoarkkitehtuurimalli.
ORM	Object-relational mapping. Oliomallin mukaisen esityksen kuvaus relaatiomallin mukaiseksi esitykseksi.
REST	REpresentational State Transfer. Ohjelmointirajapintojen toteuttamiseen tarkoitettu arkkitehtuurimalli.
SaaS	Software as a Service. Ohjelmiston tarjoaminen asiakkaille palveluna esimerkiksi verkossa, ilman tarvetta hankkia ohjelmistolisenssiä tai tarvetta asentaa ohjelmistoa omille laitteille.
SPA	Single Page Application. Yhden HTML-sivun varaan rakennettu verkkosovellus.
SSH	Secure Shell. Protokolla salattuun tietoliikenteeseen.
SQL	Structured Query Language. Standardisoitu kyselykieli relaatiotietokantojen tiedonhallintaan.
URI	Uniform Resource Identifier. Resurssin yksilöintiin tarkoitettu tunnus.
XML	Extensible Markup Language. Tiedon rakenteen kuvaamisen tarkoitettu merkintäkieli.

1 Johdanto

Insinöörityön toimeksiantaja toimii IMS Business Solutions Oy, joka on yritysten johtamiskäytäntöjen, toimintaprosessien, palveluiden ja laadunhallinnan kehittämiseen keskittynyt yritys. Yrityksellä on kaksi päätuotetta: IMS- ja Architect-ohjelmistot. Ohjelmistojen ympärille yritys tarjoaa myös laajat koulutus- ja asiantuntijapalvelut, joilla pyritään tukemaan ohjelmistojen käyttöönottoa ja asiakaskohtaisen sisällön tuottamista.

IMS-ohjelmisto on selainpohjainen organisaatioiden toiminnan kehittämiseen tarkoitettu ohjelmisto, jonka avulla voidaan rakentaa yritykselle visuaalinen toimintajärjestelmä. Tämän insinöörityön tavoitteena on kehittää työkalu IMS-ohjelmiston käyttöasteen analysointiin SaaS (Software as a Service) -palveluna ohjelmiston hankkivien asiakkaiden kohdalla, koska tällä hetkellä on hyvin vaikeaa tietää, miten asiakasyritykset käyttävät IMS-ohjelmistoa. Kehitettävän työkalun tavoitteena on kerätä hyödyllistä tietoa IMS-ohjelmiston tuotekehittäjille, myynnille ja kouluttajille.

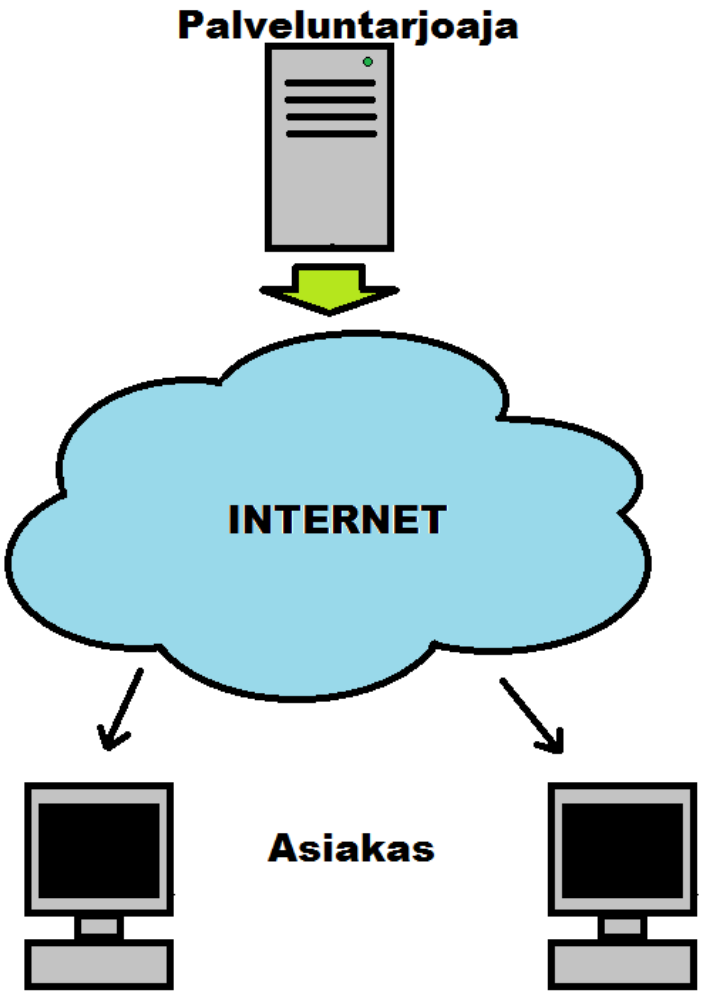
Analysointityökalulla on tarkoitus suorittaa SQL-kyselyitä SSH-yhteyden välityksellä yrityksen palvelimella sijaitsevien asiakasasennusten tietokantoihin. Kyselyiden tarkoituksena on tuottaa yksittäisiä tunnuslukuja tai aikasarjoja. Työkalun avulla ei ole mahdollista eikä sen tarkoitus ole kerätä tai tarkastella mitään asiakkaan järjestelmään tuottamaa varsinaista sisältöä. Työkalun minimivaatimuksena on tuottaa jokin rakenteellinen tiedosto, kuten JSON-, CSV- tai XML-tiedosto. Tarkoituksena on kuitenkin toteuttaa selainkäyttöliittymä, jonka avulla kerättyä tietoa voidaan esittää taulukoissa ja mahdollisesti kaavioiden muodossa.

Insinöörityössä käydään läpi projektin määritykset, käytettyjä teknologioita, oleellisia käsitteitä, sovelluksen rakennetta ja lopputulosta. Tavoitteena on antaa lukijalle selkeä kuva käytettyjen menetelmien perusteista ja tuotetun ohjelmiston toiminnallisuudesta.

2 Määrittely ja tavoitteet

Kehitettävä analysointityökalu tulee olemaan verkkosovellus, ja se on tarkoitettu IMS Business Solutions Oy:n omilla palvelimilla sijaitsevien IMS-ohjelmiston asiakaskohtaisten asennuksien tietokantojen analysointiin. IMS-ohjelmisto tarjotaan asiakkaille siis SaaS-

palveluna, jolla tarkoitetaan kuvan 1 tilannetta, jossa asiakkaat käyttävät ohjelmistoa verkon välityksellä sen sijaan, että asiakkaan tulisi hankkia ohjelmistolisenssi ja asentaa ohjelmisto omille laitteilleen.



Kuva 1. Ohjelmiston tarjoaminen SaaS-palveluna.

Palvelimia IMS-ohjelmistolle on noin 15 kappaletta ja niillä sijaitsevia asiakasasennuksia noin 300 kappaletta. Tietokannoista saatavan datan perusteella on tarkoitus selvittää ja analysoida IMS-ohjelmiston eri osioiden ja toimintojen käyttöastetta.

2.1 Yhteydet

Sovelluksen ja palvelimien välinen kommunikointi tapahtuu SSH-yhteyden ja MySQL-tunnelin välityksellä. Sovelluksella tulee siis olla mahdollista luoda SSH-yhteys ennalta

määritettyihin yrityksen palvelimiin ja luoda MySQL-tunneli niiden tietokantaan. SSH-yhteyden autentikointi toteutetaan käyttäen julkiseen avaimeen perustuvaa autentikointia, jotta yhteyksien luontiin tarvittavaa salasanaa ei tarvitse tallentaa ohjelmistokoodin sekaan. Palvelimille lisättävän käyttäjätunnuksen oikeudet palvelimella ja MySQL-tietokannoissa rajoitetaan varmuuden vuoksi mahdollisimman pieneksi.

2.2 Kyselyt

Yrityksen omalla palvelimella sijaitsevat IMS-ohjelmiston asiakasasennukset käyttävät MySQL:ään pohjautuvaa relaatiotietokantajärjestelmää MariaDB:tä, johon voidaan tehdä hakuja SQL-kyselykielellä. Sovellukseen tulee voida määritellä SQL-kyselyjä, jotka tuottavat tulokseksi yksittäisen tunnusluvun asiakasta kohden, ja ne tallennetaan analysaattorin omaan tietokantaan. Sovellukseen tulee olla myös mahdollista määrittää taulukon palauttavia kyselyitä, jotta voidaan saada esimerkiksi jonkin tunnusluvun arvot tietyltä ajanjaksolta. Projektin aikana tuli myös tarve boolean-tyyppisille kyselyille, jotka palauttavat totuusarvon tosi tai epätosi.

2.3 Tunnuslukutaulukot ja kaaviot

Tunnuslukukyselyiden tuottamista tunnusluvuista tulee olla mahdollista koostaa tunnuslukutaulukoita. Tunnuslukutaulukon soluihin määritellään tunnusluvun tuottava kysely, jonka ansioista taulukot voidaan helposti luoda kaikista asiakkaista yhden määrityksen pohjalta. Tunnuslukutaulukoita ja taulukkokyselyiden tuottamia taulukoita tulisi voida esittää myös kaavioiden muodossa.

2.4 Käyttöliittymä

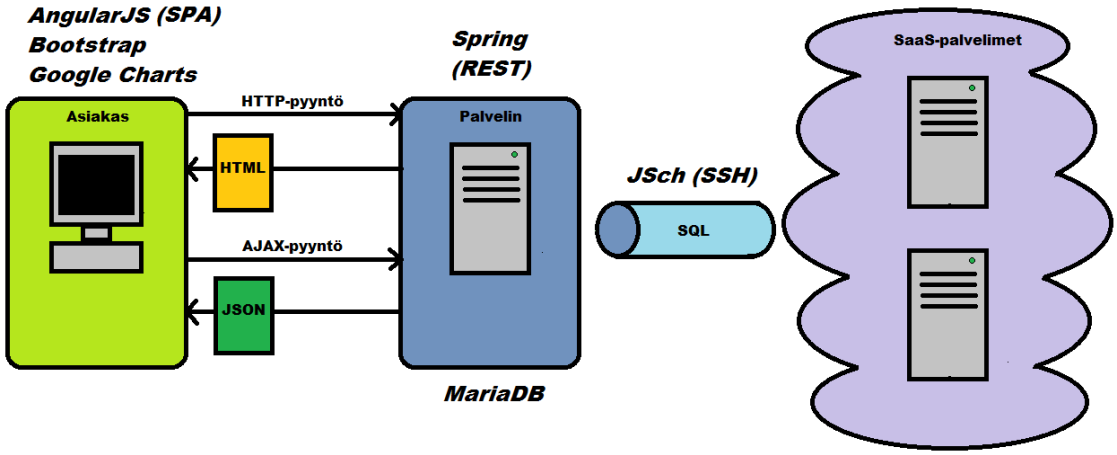
Sovellukseen on tarkoitus toteuttaa selainkäyttöliittymä, josta voidaan toteuttaa seuraavia toimenpiteitä

- SSH-yhteyksien määritys palvelimiin
- tilastoitavien asiakkaiden määritys

- tunnusluku-, boolean- ja taulukkokyselyiden lisääminen
- tunnusluvuista koostuvien tunnuslukutaulukoiden määritys
- taulukkokyselyistä tai tunnuslukutaulukoista luotavien kaavioiden määritys
- asiakaskohtaisten tilastojen tarkastelu
- kaikista asiakkaista koostuvien yleistilastojen tarkastelu.

3 Käsitteet ja teknologiat

IMS Business Solutions Oy:n puolelta annettiin vapaat kädet valita projektissa käytettävät teknologiat, mutta toivottiin kuitenkin Java-pohjaista ratkaisua. Kuvassa 2 on esitelty projektissa käytettävien tekniikoiden jakautumista sovelluksen arkkitehtuurissa.



Kuva 2. Toteutetun sovelluksen arkkitehtuuri.

Sovelluksen palvelinpuolen toiminnallisuus päätettiin toteuttaa käyttäen Javalle suunniteltua Spring-sovelluskehystä ja tarkemmin ottaen sen päälle rakennettua Spring Boot -sovelluskehystä. Sovelluksen tietokantana toimii MySQL-pohjainen MariaDB-relaatiotietokantajärjestelmä. SSH-yhteyksien luonti ja hallinta toteutetaan käyttäen JSch-kirjastoa. Sovelluksen asiakaspuolen toiminnallisuus toteutetaan JavaScript-pohjaisella Angu-

larJS-sovelluskehyksellä ja kaaviot piirretään Google Chart -kirjaston avulla. Tiedonvälitykseen asiakaspuolen ja palvelimen välillä käytetään JSON-tiedostomuotoa. Seuraavaksi käydään läpi hieman tarkemmin projektissa oleellisia käsitteitä, menetelmiä sekä mainittuja teknologioita ja niiden ominaisuuksia.

3.1 REST

REST (REpresentational State Transfer) on etenkin verkkosovellusten suunnittelussa käytetty arkkitehtuurityyli, jonka Roy Fielding esitteli teoreettisena mallina väitöskirjassaan vuonna 2000. REST-arkkitehtuuriin pätee seuraavat kuusi rajoitetta;

- Asiakas-Palvelin (Client-Server): Sovelluksen asiakas- ja palvelinpuoli tulee erottaa toisistaan. Erottamisen avulla voidaan parantaa asiakaspuolen siirrettävyyttä eri alustoille ja palvelinpuolen yksinkertaistumisen kautta sovelluksen skaalautuvuutta. Erotus sallii myös molempien komponenttien kehittämisen itsenäisesti.

- Tilaton (Stateless): Asiakkaan ja palvelimen välisen yhteydenpidon tulisi olla tilatonta, eli palvelimen ei tarvitse muistaa missä tilassa asiakas on. Asiakkaan tulee siis lähettää pyynnössään kaikki palvelimen tarvitsema tieto sen käsittelemiseksi.

- Kerrostettu järjestelmä (Layered System): Asiakkaan ja palvelimen välissä voi olla useita eri kerroksia, kuten yhdyskäytäviä, palomuureja tai välityspalvelimia. Kerroksia voidaan tarvittaessa lisätä, muokata, poistaa tai järjestellä uudestaan.

- Välimuisti (Cache): Palvelimelta tulevat vastaukset tulee määritellä sen mukaan, voidaanko niiden kanssa mahdollisesti hyödyntää välimuistia. Välimuistia käytettäessä asiakas voi hyödyntää palvelimelle jo aiemmin tehtyjen pyyntöjen vastauksia vähentäen palvelimen kuormitusta ja parantaen suorituskykyä.

- Yhtenäinen rajapinta (Uniform Interface): Kaikkien REST-palvelun ja asiakaspuolen kommunikaatioon käytettävien komponenttien tulisi tarjota yhdenmukainen rajapinta.

- Koodi pyynnöstä (Code on Demand): Valinnaisena rajoitteena asiakaspuoli voi laajentaa toiminnallisuuttaan lataamalla palvelimelta koodia, kuten Java-appletteja tai JavaScript-skriptejä. [1, s. 1-2; 2 s.6-8.]

REST-palveluissa avainasemassa ovat resurssit, joita ovat oikeastaan kaikki asiat sovelluksessa, mihin voidaan päästä käsiksi tai muokata. Ennen kuin resursseja voidaan käsitellä, täytyy ne voida yksilöidä ja tunnistaa. Verkkosovelluksissa resurssit normaalisti

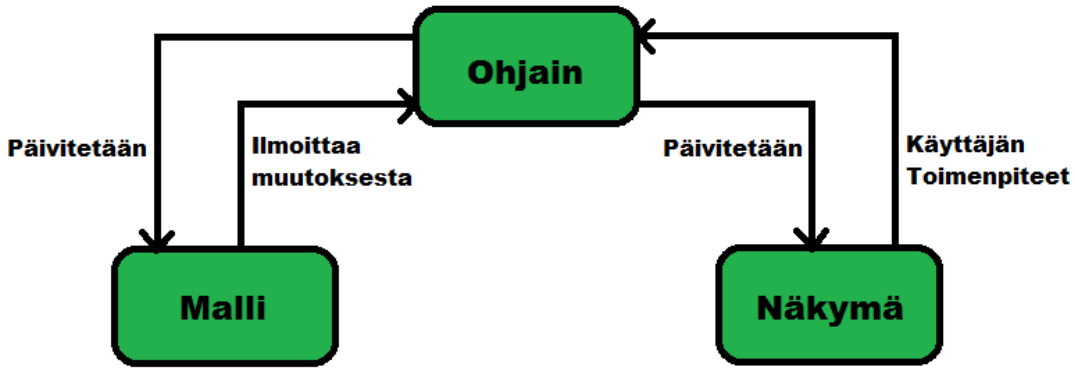
yksilöidään URI:n (Uniform Resource Identifier) avulla. Käytännössä asiakas pääsee käsiksi resursseihin lähettämällä palvelimelle resurssin URI:n kohdistetun HTTP-pyynnön. REST-rajapinnoissa käytetyimmät HTTP-metodit ovat GET, POST, PUT ja DELETE, joilla voidaan hakea, lisätä, päivittää tai poistaa resursseja. Palvelin vastaa HTTP-pyyntöön palauttamalla resurssin mukauttaen sen ensin yhtenäisen rajapinnan mukaisesti määritettyyn muotoon, joka voi olla esimerkiksi JSON tai XML. [1, s. 2-4.]

3.2 MVC-malli

Ohjelmistokehityksessä suunnittelumallilla tarkoitetaan yleistä uudelleenkäytettävää ratkaisua ohjelmistojen suunnittelussa esiintyviin ongelmatilanteisiin. MVC eli Model-View-Controller-malli on erityisesti käyttöliittymäohjelmointiin hyvin soveltuva suunnittelumalli, jonka perusperiaatteena on selkeästi jakaa sovellus kolmeen komponenttiin; malliin (Model), näkymään (View) ja ohjaimeen (Controller). MVC-mallin mukaisten kolmen komponentin vastuut ja tehtävät voidaan määritellä seuraavasti.

- Malli esittää sovelluksen tilaa ja sillä olevaa tietoa.
- Näkymä on vastuussa tiedon esittämisestä käyttäjälle.
- Ohjain vastaa mallin ja näkymän muuttamisesta käyttäjän toimenpiteiden johdosta.

Yksi tapa toteuttaa ja esittää komponenttien välinen vuorovaikutus voidaan nähdä kuvassa 3: ohjain reagoi käyttäjän toimenpiteisiin näkymässä ja päivittää tarvittaessa mallin. Tyypillisesti malli ei tiedä mitään näkymistä tai ohjaimista, vaan mallin muuttuessa se ilmoittaa tarkkailijoilleen, että muutos on tapahtunut. Ohjain reagoi mallin muutokseen ja päivittää tarvittaessa näkymän.

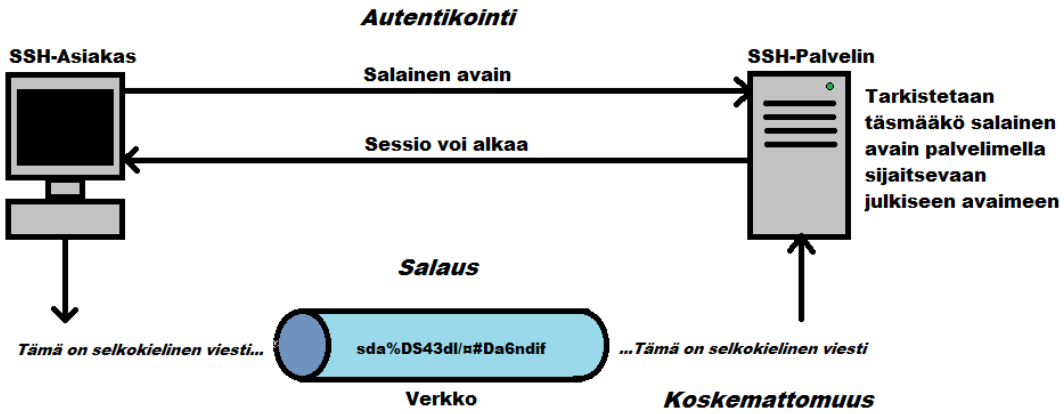


Kuva 3. MVC-mallin mukaisten kolmen komponentin välinen vuorovaikutus [3].

Ohjelmiston arkkitehtuurin jakaminen selkeisiin osiin parantaa muun muassa ohjelmiston uudelleenkäytettävyyttä ja testattavuutta, sekä helpottaa ylläpitoa. Komponenttien vaihtaminen ja esimerkiksi useampien erilaisten näkymien toteuttaminen on myös huomattavasti vaivattomampaa. [3; 4, s. 51-52.]

3.3 SSH ja JSch

SSH (Secure Shell) on turvalliseen tiedonsiirtoon kehitetty protokolla, joka tarjoaa kahden tietokoneen väliselle verkkoyhteydelle autentikoinnin tiedon salauksen ja tiedon koskemattomuuden kuvan 4 mukaisesti. Kahdesta tietokoneesta toinen on aina asiakas ja toinen palvelin. Yhteyden muodostamiseksi asiakkaalla täytyy olla asennettuna SSH-asiakasohjelmisto ja palvelimella SSH-palvelinohjelmisto. [5, s.3; 6.]



Kuva 4. Autentikointi, tiedon salaus ja tiedon koskemattomuus [5, s.3].

SSH-palvelin pyytää yhteyden luonnin yhteydessä SSH-asiakasta tunnistautumaan, ja se kykenee luotettavasti varmistamaan yhteydenottajan identiteetin. Yhteyden välillä kulkeva tieto sekoitetaan niin, että vain lähettäjä ja vastaanottaja voivat lukea sen selkokielisenä. SSH takaa myös tiedon muuttumattomuuden ja osaa havaita mahdolliset kolmannen osapuolen tahalliset muutokset yhteyden välillä kulkevaan tietoon. [5, s. 3.]

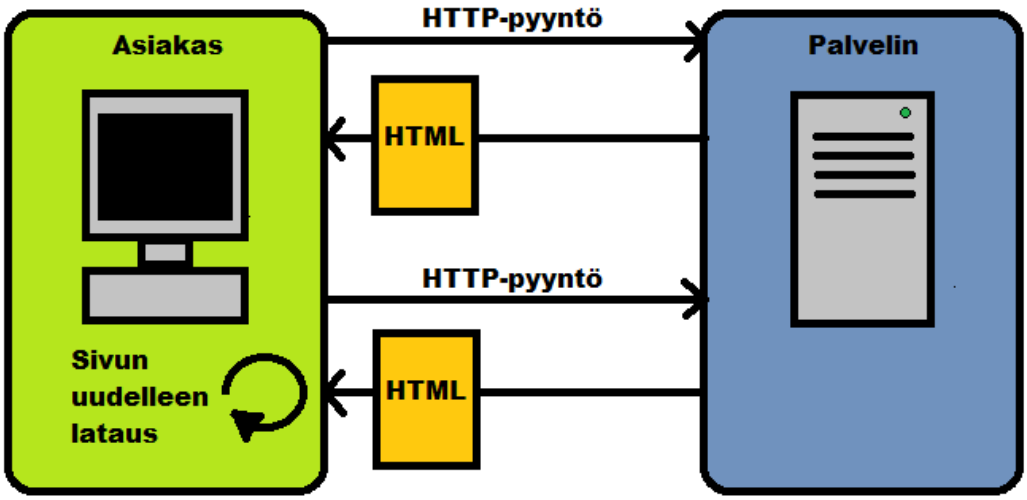
SSH-yhteyden autentikointi voi tapahtua, joko käyttäjätunnuksen ja salasanan avulla tai SSH-identiteetin avulla käyttämällä julkiseen avaimeen perustuvaa tunnistautumista. SSH-identiteetti koostuu julkisesta ja salaisesta avaimesta, jotka toimivat avainparina. Käytettäessä SSH-identiteettiä tunnistautumiseen yhteydenottajalla täytyy olla käyttäjätunnus ja löytyä salaisen avaimen sisältävä tiedosto, joka täsmää palvelimelta löytyvään julkiseen avaimeen. [5, s. 166; 6.]

JSch on puhtaasti Javalla luotu avoimen lähdekoodin SSH-toteutus, ja se mahdollistaa SSH-yhteyden luomisen lisäksi mm. porttiohjauksen (port forwarding) ja tiedostojen siirtämisen. JSch on suhteellisen matalan tason kirjasto, joten ohjelmoijan täytyy kirjoittaa suhteellisen paljon koodia tehdäkseen yksinkertaisiakin toimenpiteitä. JSch kirjaston dokumentaatio on myös käytännössä olematonta, muutamia esimerkkejä lukuun ottamatta. Toisaalta Java-pohjaisia vaihtoehtoja on hyvin vähän ja JSch vaikutti niistä sopivimmalta ratkaisulta tämän projketin kannalta, koska sovelluksessa SSH:n välityksellä tehtävät toimenpiteet ovat hyvin yksinkertaisia. [7.]

3.4 SPA

SPA on lyhenne termistä single page application eli suomennettuna yhden sivun sovellus. Käytännössä tällä tarkoitetaan verkkosovellusta, joka ladataan palvelimelta yhtenä HTML-verkkosivuna, jota käytetään kehikkona kaikelle sovelluksen sisällölle. Tyypillisesti samalla ladataan kaikki sovelluksen ja käyttäjän vuorovaikutuksen toteuttamiseksi tarvitut JavaScript-, HTML- ja CSS-tiedostot, mutta niitä voidaan tarvittaessa ladata dynaamisesti myös jälkikäteen. SPA-sovellusten ei siis tarvitse tehdä kuin yksi kutsu palvelimelle, jonka jälkeen kaikki sovelluksen eri näkymät ja toiminnallisuus toteutetaan käyttäjän selaimessa, ilman tarvetta ladata sivua uudelleen. SPA-sovelluksissa palvelimella ei ole mitään tietoa käyttöliittymän toimintalogiikasta ja suurin osa sovelluksen toiminnallisuudesta rakennetaankin asiakaspuolella. Tämä poikkeaa perinteisen verkkosovelluksen toteutustavasta, jossa sovellus on hyvin riippuvainen vuorovaikutuksesta palvelinpuolen kanssa ja uusia sivuja ladataan palvelimelta aina navigoinnin yhteydessä. [8, s.3.]

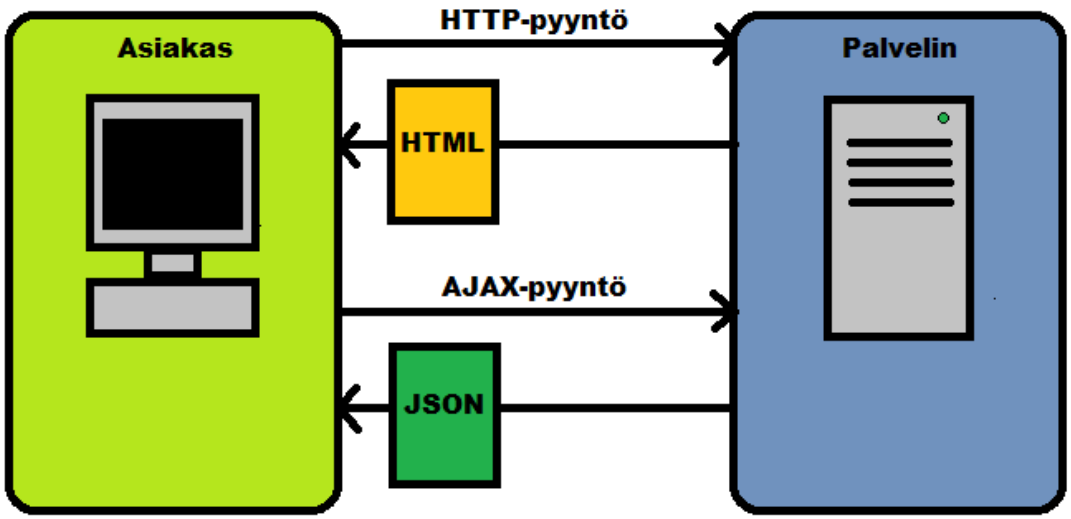
Perinteinen verkkosovellus luo eli renderöi verkkosivun saatuaan HTTP-pyynnön selaimelta ja luotu verkkosivu lähetetään selaimeen HTTP-vastauksena, joka aiheuttaa verkkosivun päivityksen selaimessa. Päivityksen yhteydessä siis koko verkkosivu korvataan kokonaan uudella sivulla. Kuvassa 5 esitellään perinteisen verkkosovelluksen elinkaarta.



Kuva 5. Perinteisen verkkosovelluksen elinkaari [9].

Kuvasta voidaan huomata, kuinka käyttäjän toimenpiteistä aiheutuvien HTTP-pyyntöjen perusteella palvelimelta palautetaan vastaus HTML-sivuina. Perinteisten verkkosovellusten ollessa hyvin palvelinkeskeisiä, on asiakaspuolella toteutettu toiminnallisuus hyvin kevyttä ja esimerkiksi JavaScriptiä käytetään pääosin vain pienten käyttöliittymämuutoksien tai animaatioiden luomiseen. [8, s. 6-7.]

SPA-sovelluksissa sen sijaan muuttuvan sisällön esittämiseksi käytetään sovelluksen alussa ladatun HTML-sivun rakenteesta löytyvää yksittäistä HTML-elementtiä. Tämän HTML-elementin sisältö voidaan korvata uudella näkymällä eli uudella katkelmalla HTML-koodia. Kuvassa 6 esitellään SPA-sovelluksen elinkaarta.



Kuva 6. SPA-sovelluksen elinkaari [9].

Verrattuna perinteiseen verkkosovellukseen voidaan huomata, että alustavan HTTP-pyynnön jälkeen palvelimelle tehdäänkin AJAX (Asynchronous JavaScript And XML) -pyyntöjä. AJAX-pyynnöt ovat käytännössä JavaScriptillä asynkronisesti toteutettuja HTTP-pyyntöjä, jotka palauttavat tietoa esimerkiksi XML- tai JSON-muodossa. Asynkroniset operaatiot eivät estä käyttöliittymän toimintaa, jolloin käyttäjän näkökulmasta sovellus pysyy koko ajan responsiivisena eli reagoivana käyttäjän toimenpiteille. Operaation jälkeen voidaan näkymää tarvittaessa päivittää uudella tiedolla. Toinen selkeä ero perinteisen- ja SPA-sovelluksen välillä on, kuinka sovelluksen tilaa hallitaan ja säilytetään. SPA-sovelluksessa palvelimella ei ole tietoa sovelluksen tilasta ja tietoa tilasta voidaankin säilyttää esimerkiksi selaimen muistissa. [8, s.4, 11-12.]

SPA-sovellusten ansioista käyttäjille voidaan tarjota perinteisiä työpöytäsovelluksia muistuttavia verkkosovelluksia. SPA-sovellukset yhdistävät verkkosovellusten tarjoaman alustariippumattomuuden ja työpöytäsovellusten responsiivisuuden. SPA-sovelluksen ollessa verkkosovellus on sen käyttöönotto ja käyttäminen mutkatonta. Käyttäjien ei tarvitse asentaa sovellusta omalle tietokoneelleen ja päivitysten saaminen ei vaadi käyttäjältä mitään toimenpiteitä. [8, s. 13.]

3.5 Spring

Spring on avoimen lähdekoodin sovelluskehys, joka kehitettiin aikoinaan helpottamaan Java EE -sovellusten luontia tarjoten kevyemmän ja joustavamman vaihtoehdon raskaille Java EE -teknologioille. Spring on kevyempi, koska se on suunniteltu niin, että sillä kehitetyt sovellukset ovat mahdollisimman vähän riippuvaisia Springin omasta ohjelmointirajapinnasta. Tämän ansiosta sovelluksiin tarvitsee tehdä hyvin vähän muutoksia, jotta voidaan hyödyntää Springin tarjoamia ominaisuuksia. [10, s. 1.]

Springin toiminta perustuu Inversion of Control (IoC) -malliin, jossa eri komponenttien välisten suhteiden luonti ja hallinta ulkoistetaan. Perinteisesti esimerkiksi, jos luokka A on riippuvainen luokasta B, niin luokka A luo ilmentymän luokasta B. IoC-mallissa luokka B välitetään luokalle A ajon aikana, jonkin ulkoisen prosessin toimesta. Tällaista toimintaperiaatetta kutsutaan myös tarkemmin kuvaavalla termillä Dependecy Injection (DI) eli riippuvuuksien injektointi. [10, s. 5.]

Riippuvuuksien injektoinnilla saavutetaan perinteisempään lähestymistapaan verrattuna ainakin seuraavia etuja:

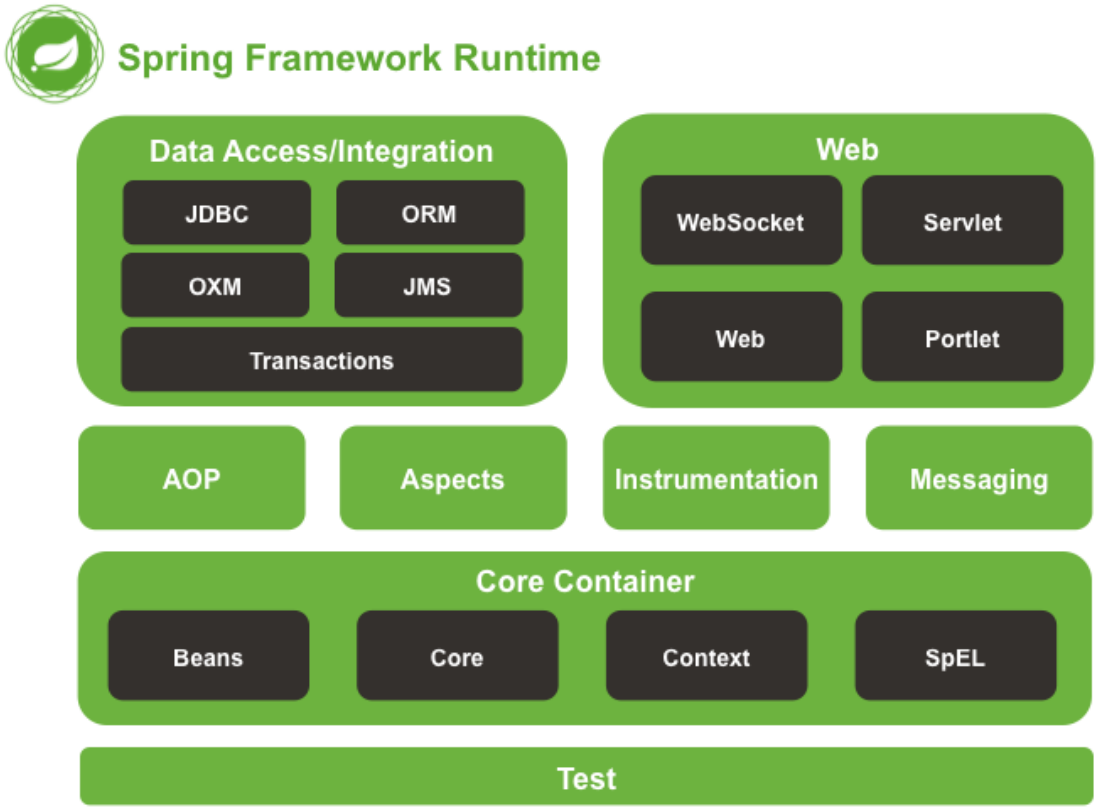
- Riippuvuuksien vähentyminen: Riippuvuuksien injektoinnilla voidaan poistaa tai ainakin vähentää komponenttien tarpeettomia riippuvuuksia. Komponentit ovat alttiitta riippuvuuksien muutoksille, jolloin riippuvuudessa tapahtuvan muutoksen yhteydessä täytyy mahdollisesti myös komponentin mukautua muutoksiin. Samalla vähennetään huomattavasti myös komponenttien yhdistämiseen tarvittavan koodin kirjoittamista.
- Uudelleen käytettävämpää koodia: Vähentämällä komponenttien riippuvuuksia on niiden uudelleenkäyttö eri ympäristössä huomattavasti helpompaa. Kun riippuvuudet voidaan injektoida, voidaan ne myös konfiguroida ulkoisesti ilman tarvetta komponentin koodin muutokseen.

- Testattavampaa koodia: Myös komponenttien testattavuus parantuu riippuvuuksien injektoinnilla. Käytettäessä injektoituja riippuvuuksia voidaan komponentille helposti injektoida kyseistä riippuvuutta matkiva toteutus.

- Luettavampaa koodia: Riippuvuuksien injektointi siirtää riippuvuuksien määrittelyn komponenttien rajapintaan, joka helpottaa komponentin riippuvuuksien hahmottamista. Koodi on luettavampaa kun ei tarvitse käydä läpi komponentin toteutusta löytääkseen komponentin riippuvuudet.

- Riippuvuuksien ”kantamisen” vähentyminen: Riippuvuuksien kantamisella tarkoitetaan tilannetta, jossa komponentti ottaa parametrin, jota se ei suoranaisesti itse tarvitse, mutta jokin komponentin käyttämistä komponenteista tarvitsee kyseistä parametria. Käytettäessä riippuvuuksien injektointia voidaan parametri injektoida suoraan sitä tarvitseville komponenteille. [11.]

Spring-sovelluskehys tarjoaa kehittäjille erittäin paljon erilaisia ominaisuuksia ja valmiita tominnallisuuksien toteutuksia, jotka ovat organisoitu noin kahteenkymmeneen moduuliin, jotka ovat ryhmitelty kuvan 7 mukaisesti.



Kuva 7. Spring-sovelluskehyksen moduulit [12].

Pääryhmien ominaisuudet jakautuvat seuraavasti:

- Core Container: Core Container koostuu Beans-, Core-, Context- ja SpEL-moduuleista. Core- ja Beans-moduulit tarjoavat Spring-sovelluskehyksen keskeiset ominaisuudet kuten riippuvuuden injektoinnin. Context-moduuli rakentuu näiden kahden moduulin päälle tarjoten muun muassa tuen lokalisaatiolle. SpEL (Spring Expression Language) -moduuli mahdollistaa objektien kuvauksien ajonaikaisen tiedustelun ja muokkauksen.

- AOP (Aspect Oriented Programming) ja Aspects: AOP-moduuli tarjoaa tuen aspektipohjaiselle ohjelmoinnille. Aspektipohjaisessa ohjelmoinnissa pyritään kapseloimaan ohjelmiston sellaiset usein käytetyt ominaisuudet, jotka kuuluisi erottaa komponenttien toteutuksista kuten virheenkäsittely tai lokitus. Aspects-moduuli tarjoaa tuen AspectJ:lle, joka on aspektipohjaisen ohjelmoinnin tarjoava lisäosa Java-ohjelmointikieleen.

- Instrumentation: Instrumentation-moduuli tarjoaa muun muassa laajat suorituskykyyn ja resurssien käyttöön liittyvät tilastot sekä tarjoaa ajonaikaisen resurssien hallinnan.

- Messaging: Messaging-moduuli tarjoaa pohjan ja työkalut viestitykseen perustuvien ohjelmistojen luontiin.

- Data Access/Integration: Data Access/Integration-ryhmän tarjoamat moduulit antavat laajat työkalut tiedonhallintaan tarjoten tuen JDBC-, ORM-, OXM- ja JMS-teknologioille. Moduuli tarjoaa muun muassa abstraktio-kerroksen JBDC:hen ja integraatio-kerrokset JPA-, JDO- ja Hibernate ORM-ohjelmointirajapinnoille.

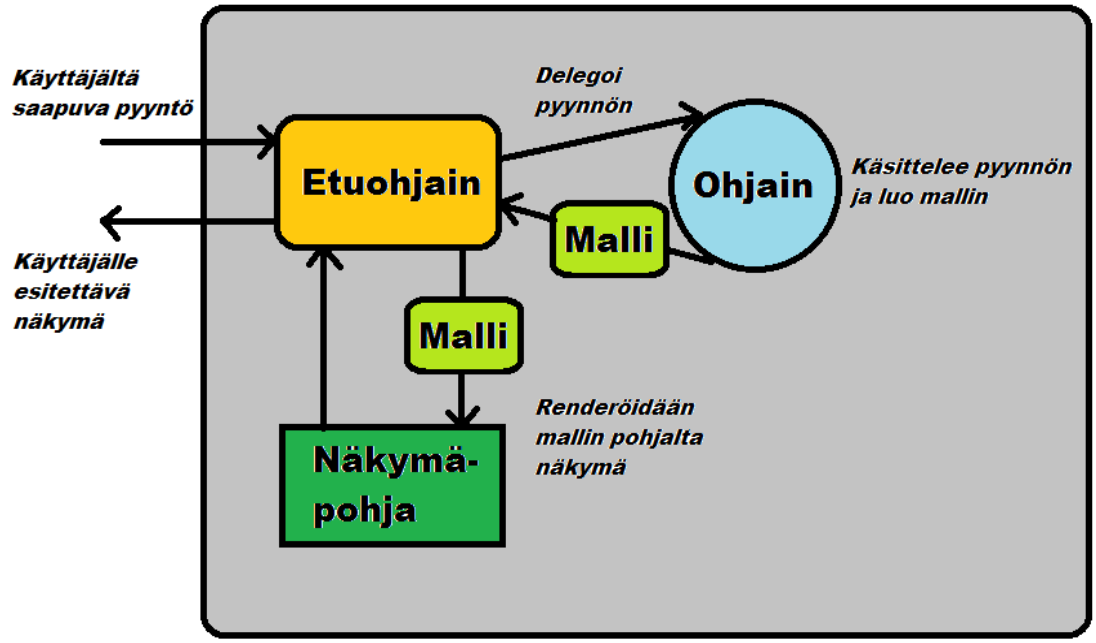
- Web: Web-ryhmä tarjoaa työkalut verkkosovellusten ja REST-rajapintojen kehitykseen Java:lla MVC-mallin mukaisesti.

- Test: Test-moduuli tukee Spring-komponenttien yksikkö- ja integraatiotestausta. Moduuli tarjoaa myös matkija-objektit, joiden avulla toisista komponenteista riippuvaa koodia voidaan testata eristetysti. [12.]

3.5.1 Spring MVC:n arkkitehtuuri

Verkkosovelluksissa perinteiseen MVC-arkkitehtuuriin täytyy ottaa hieman erilainen lähestymistapa HTTP-protokollan luonteen vuoksi. Perinteisessä MVC-arkkitehtuurin toteutuksessa käyttäjän toimenpiteen johdosta ohjain (controller) päivittää mallin (model), joka vuorostaan puskee muutokset näkymään (view). Tämän jälkeen näkymä päivittää itsensä mallin uudella datalla. Verkkosovelluksissa sen sijaan tyypillisesti käyttäjän suorittaessa toimenpiteen lähtee sovellukselle pyyntö, jonka seurauksena näkymä päivitetään ja palautetaan vastauksena käyttäjälle. Verkkosovelluksessa täytyykin muutokset

hakea palvelimelta sen sijaan, että muutokset puskettaisiin näkymään. Erona perinteiseen MVC-arkkitehtuurimalliin Spring sisällyttää siihen myös etuohjaimen (Front Controller), kuten kuvassa 8 voidaan huomata. [13, s. 51-52.]



Kuva 8. MVC-malli verkkosovellukselle [13, s. 53].

Etuohjain käsittelee ja delegoi käyttäjältä tulevat pyynnöt eteenpäin sopiville ohjaimille. Kun valittu ohjain on saanut prosessoitua ja päivitettyä mallin, etuohjain määrittää mallista, mikä näkymä tulee renderöidä. Etuohjaimessa siis käsitellään asiakkaan pyyntö, valitaan sopiva ohjain mallin käsittelyyn ja päätetään, mikä näkymä käyttäjälle esitetään. [13, s. 53.]

3.5.2 Spring Boot

Spring Boot on sovelluskehys, joka on luotu Springin päälle helpottamaan ja nopeuttamaan Spring-pohjaisten sovellusten luontia vähentäen tai poistaen useita turhia toimenpiteitä ja konfiguraatioita. [14.]

Spring Boot-kehys helpottaa Spring-sovellusten kehitystä seuraavilla tavoilla

- Nopeuttaa Spring-sovellusten luomista lisäten näin tuottavuutta.

- Vähentää tarvittavaa boilerplate-koodia, annotaatioita ja konfiguraatioita.
- Yksinkertaistaa integraatioita muiden Spring-sovelluskehysten, kuten Spring JDBC, ORM, Data, Security jne. kanssa.
- Tarjoaa sisäänrakennetun HTTP-palvelimen, kuten Tomcatin tai Jettyn.
- Sisältää komentorivi-työkalun helpottamaan sovelluksen kehitystä ja testausta komentoriviltä käsin.
- Tarjoaa paljon liitännäisiä helpottamaan sovelluksen kehitystä ja testausta. [14.]

3.5.3 Spring Data JPA

Spring Data JPA on sovelluskehys, jonka tarkoituksena on helpottaa JPA (Java Persistence API) -pohjaisten tiedonhallintakerrosten luomista. Spring Data JPA:n avulla kehittäjät voivat esimerkiksi määrittää tiedonhallintakerroksia käyttäen hyväksi Spring Data JPA:ssa määriteltyjä rajapintoja. Spring Data JPA automatisoi kerrosten konkreettisten toteutuksen luonnin vähentäen näin ollen tarvittavan koodin kirjoitusta. [15.]

3.6 AngularJS

AngularJS on pääosin Googlen kehittämä ja ylläpitämä dynaamisten verkkosovellusten kehitykseen tarkoitettu avoimen lähdekoodin JavaScript-sovelluskehys. Sen tavoitteena on mahdollistaa pienempien, kevyempien, yksinkertaisempien, laajennettavampien ja testattavampien SPA-sovellusten kehitys. Seuravaaksi esitellään lyhyesti muutamia AngularJS:n tärkeimpiä konsepteja ja komponentteja. [16.]

Asiakaspuolen mallipohjat (Client-Side Templates)

Perinteisesti verkkosovellukset kasaavat HTML-pohjan ja liittävät tarvittavan datan siihen palvelimen puolella, jonka jälkeen valmis sivu palautetaan selaimelle. Myös monet SPA-sovellukset tekevät tätä ainakin tiettyyn pisteeseen asti. AngularJS hoitaa tämän

hieman eri tavalla; HTML-pohja sekä data lähetään selaimelle ja kasataan siellä. Palvelimen tehtäväksi jää näin toimiminen vain staattisena resurssina HTML-pohjille, joka tarjoaa niille niiden tarvitseman datan. [16, s. 2.]

AngularJS ja MVC-malli

AngularJS-sovelluskehyksen pohjatuu käytännössä MVC-mallin mukaiseen arkkitehtuuriin. AngularJS sovellukset voidaan jakaa MVC-mallin mukaisiin komponentteihin seuraavasti:

Malli koostuu käytännössä JavaScript-objekteista ja niiden sisältämästä datasta.

Näkymänä toimii DOM (Document Object Model).

Ohjaimina toimivat JavaScript-luokat, jotka sisältävät sovelluksen toimintalogiikan. [16, s. 3.]

AngularJS:n yhteydessä voidaan myös puhua MVVM (Model-View-View-Model) -mallista, jossa MVC-mallin ohjain korvataan näkymä-mallilla (View-Model). Perinteisessä MVC-mallissa näkymään tehdyt muutokset eivät aiheuta muutosta mallissa, mutta AngularJS:n kahdensuuntaisen datan sitomisen vuoksi näkymän muutokset heijastuvat myös malliin. MVVM-mallin perusajatuksena on siis yhdistää malli ja näkymä näkymämalli -komponentilla, joka osaa tarvittaessa peilata komponenteissa tapahtuneet muutokset toisiinsa. [17.]

Datan sitominen (Data binding)

Datan sitomisessa tarkoituksena on määritellä, mitkä näkymän osat vastaavat mitäkin mallin osaa. Toisin sanoen luodaan ennalta määritetty yhteys sovelluksen käyttöliittymän ja sen tietorakenteen välille. AngularJS mahdollistaa kahdensuuntaisen datan sitomisen (two-way data binding) automaattisesti, jonka ansioista muutokset näkymässä peilautuvat suoraan malliin ja toisinpäin. Tämän ansioista kehittäjän ei tarvitse turhaan luoda itse mitään erillistä koodia saavuttaakseen tämänkaltaisen toiminnallisuuden. [16, s. 3-4.]

Riippuvuuksien injektointi (Dependency Injection)

AngularJS:ään on sisäänrakennettu riippuvuuksien injektointi mekanismi, jonka ansiosta luokkien välille ei tarvitse luoda riippuvuuksia, vaan luokat voivat suoraan pyytää konstruktorissa tarvitsemiaan komponentteja. AngularJS huolehtii automaattisesti riippuvuuksien luomisesta ja hallinnasta. [16, s. 5.]

Lausekkeet (Expressions)

AngularJS mahdollistaa pääsyn sovelluksen JavaScript-koodiin suoraan HTML-pohjilta lausekkeiden avulla. Lausekkeet muodostuvat kahdesta sisäkkäisestä aaltosulkeesta ja niiden sisällä voidaan esimerkiksi suorittaa laskutoimenpiteitä kuvan 9 mukaisesti.

```
<div>1 + 1 = {{1 + 1}}</div>
```

Kuva 9. Esimerkki AngularJS-lausekkeen käytöstä.

Käyttäjälle kuvan esimerkki tulostuisi seuraavasti: 1 + 1 = 2. Lausekkeita käytetään yleensä pieniin toimenpiteisiin, kuten juurikin muuttujien välisiin laskutoimituksiin tai mallissa sijaitsevan datan esittämiseen. [18.]

Filtterit (Filters)

Filttereiden eli suotimien avulla voidaan alustaa lausekkeiden sisällä olevaa käyttäjälle näytettävää dataa. Filttereitä voidaan käyttää suoraan HTML-pohjissa tai vaihtoehtoisesti ohjaimissa ja palveluissa.

```
<div>{{"tekstiä" | uppercase}}</div>
```

Kuva 10. Esimerkki AngularJS-filtterin käytöstä.

Kuvan 10 esimerkissä käytetään filtteriä, joka alustaa lausekkeen sisällä olevan muuttujan sisällön näkymään isoilla kirjaimilla kirjoitettuna riippumatta sen määritetystä muodosta. Käyttäjälle kuvan esimerkki tulostuisi seuraavasti: TEKSTIÄ. AngularJS sisältää paljon valmiita filttereitä ja niitä on myös mahdollista luoda itse. [18; 19.]

Direktiivit

AngularJS mahdollistaa HTML-syntaksin laajentamisen direktiiveillä, jotka ovat DOM-elementteihin toiminnallisuutta liittäviä niin sanottuja merkitsijöitä. AngularJS tarjoaa laajan valikoiman valmiita direktiivejä, mutta direktiivejä on myös mahdollista kirjoittaa itse.

```
<!-- tämä elementti näytetään vain, jos 1 + 1 = 2 -->
<div ng-show="(1 + 1) == 2"></div>
```

Kuva 11. Esimerkki AngularJS:n ng-show -direktiivin käytöstä.

Kuvan 11 esimerkissä on lisätty valmiiksi AngularJS:ssä määritetty ng-show -direktiivi div-elementtiin. Direktiiviin määritetyn ehtolausekkeen mukaisesti voidaan, joko näyttää tai jättää näyttämättä kyseinen elementti. Kuvan esimerkissä oleva ehtolause on tosi ja näin ollen elementti näytetään käyttäjälle. [20.]

Moduulit

AngularJS-sovellukset rakennetaan moduuleihin, jotka ovat sovelluksen eri osat sisältäviä säiliöitä. Moduulit voivat riippua toisistaan, ja niissä voidaan määrittää ohjaimia, palveluja, direktiivejä ja filttereitä. Moduuleita käytetään, koska AngularJS sovelluksilla ei ole yleisesti ohjelmoinnista tuttua main-metodia, joka alustaisi ja yhdistäisi kaikki sovelluksen osat yhteen. Moduulit ovat siis komponentteja joihin sovelluksen tominnallisuudet liitetään. Käyttämällä moduuleja paketoituu koodi myös suoraan helposti uudelleen käytettäviin osiin. [18; 19.]

```
var helloWorldApp = angular.module("helloWorldApp", []);
```

Kuva 12. Esimerkki moduulin määrityksestä AnguarJS-sovelluksessa.

Moduuli voidaan määrittää kuvan 12 mukaisesti. Moduuli-funktion ensimmäisenä argumenttina on moduulin nimi, joka tässä tapauksessa on *helloWorldApp*. Toisena argumenttina annetaan taulukko, joka sisältää moduulien nimet, joista määritetty moduuli on riippuvainen. Esimerkin tapauksessa taulukko on tyhjä eli määritetty moduuli ei ole riippuvainen toisista moduuleista. [21.]

Scope

Scope on objekti, joka viittaa sovelluksen malliin ja yksinkertaisesti ajateltuna sen voidaan ajatella linkittävän ohjaimet ja näkymät keskenään. Scopen voisi suomentaa sanalla näkyvyysalue, joka kuvaakin hyvin sen toimintaa. AngularJS-ohjaimen scope-objektiin eli näkyvyysalueeseen voidaan lisätä muuttujia tai funktioita, jotka ovat sen jälkeen käytettävissä näkymässä johon ohjain on liitetty. [22.]

Ohjaimet

AngularJS:n ohjaimet sisältävät käytännössä sovelluksen toimintalogiikan; ohjaimet ovat JavaScript-funktioita, jotka suorittavat tai käynnistävät suurimman osan käyttöliittymän toiminnallisuuksista. Tyypillisesti ohjainten vastuualueeseen kuuluu:

- Näkymän tarvitseman datan hakeminen palvelimelta.
- Näkymässä näytettävän datan määritys.
- Näkymässä tarvittavan toiminnallisuuden määritys.
- Käyttäjän toimenpiteisiin reagoivien funktioiden määritys.

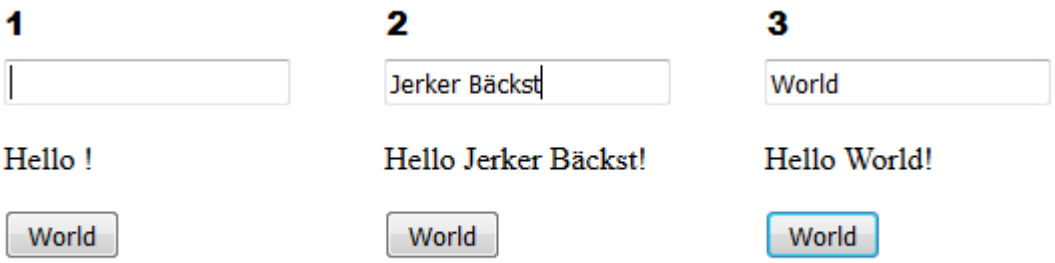
Ohjaimissa määritetään siis sen scopessa käytettävissä ja näkyvissä oleva data, sekä toiminnallisuuden tarjoamat funktiot. Ohjaimet liitetään haluttuihin osiin näkymän DOM-rakenteessa käyttäen direktiiviä ng-controller, jonka seurauksena AngularJS luo uuden instanssin ohjaimesta ja sille luodaan oma scope. [23; 24.]

Kuvassa 13 on esimerkki, kuinka *helloWorldApp* nimiseen moduuliin määritetään *HelloWorldController* niminen ohjain ja kuinka se liitetään näkymässä olevaan div-elementtiin. Käytännössä kyseisen div-elementin sisällä on nyt käytettävissä kaikki *HelloWorldController*-ohjaimen scopeen määritettävät muuttujat ja funktiot.

```
<html ng-app="helloWorldApp">
    <head>
        <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.7/angular.min.js"></script>
        <script>
            var app = angular.module("helloWorldApp", []);
            app.controller("HelloWorldController", ['$scope', function($scope){
                $scope.text = '';
                $scope.world = function() {
                    $scope.text = 'World';
                };
            }]);
        </script>
    </head>
    <body>
        <div ng-controller="HelloWorldController">
            <input ng-model="text">
            <p>Hello {{text}}!</p>
            <button ng-click="world()">World</button>
        </div>
    </body>
```

Kuva 13. Esimerkki ohjaimien määrityksestä ja käytöstä AngularJS-sovelluksessa.

Ohjaimen scopeen alustetaan tyhjä string-muuttuja *text*. Ohjaimessa määritetään myös funktio *world*, jota kutsumalla voidaan asettaa *text*-muuttujan sisällöksi teksti *World*. Määritetty funktio on liitetty painikkeeseen ng-click -direktiivillä. Esimerkissä näkymään on myös lisätty käyttäjän syötettä varten tekstikenttä. Tekstikenttään syötetty data on sidottu *text*-muuttujaan ng-model -direktiivillä. Käyttäjän selaimessa esimerkki näyttää alkutilanteessa kuvassa 14 näkyvän kohdan 1 mukaiselta.



Kuva 14. Esimerkki AngularJS:n kaksi suuntaisesta datan sitomisesta.

Käyttäjän alkaessa kirjoittaa tekstikenttään vaikkapa omaa nimeään voidaan muutos huomata näkymässä kirjain kirjaimelta AngularJS:n kaksi suuntaisen datan sitomisen ansiosta kohdan 2 mukaisesti. Käyttäjän painaessa World-painiketta alustetaan *text*-

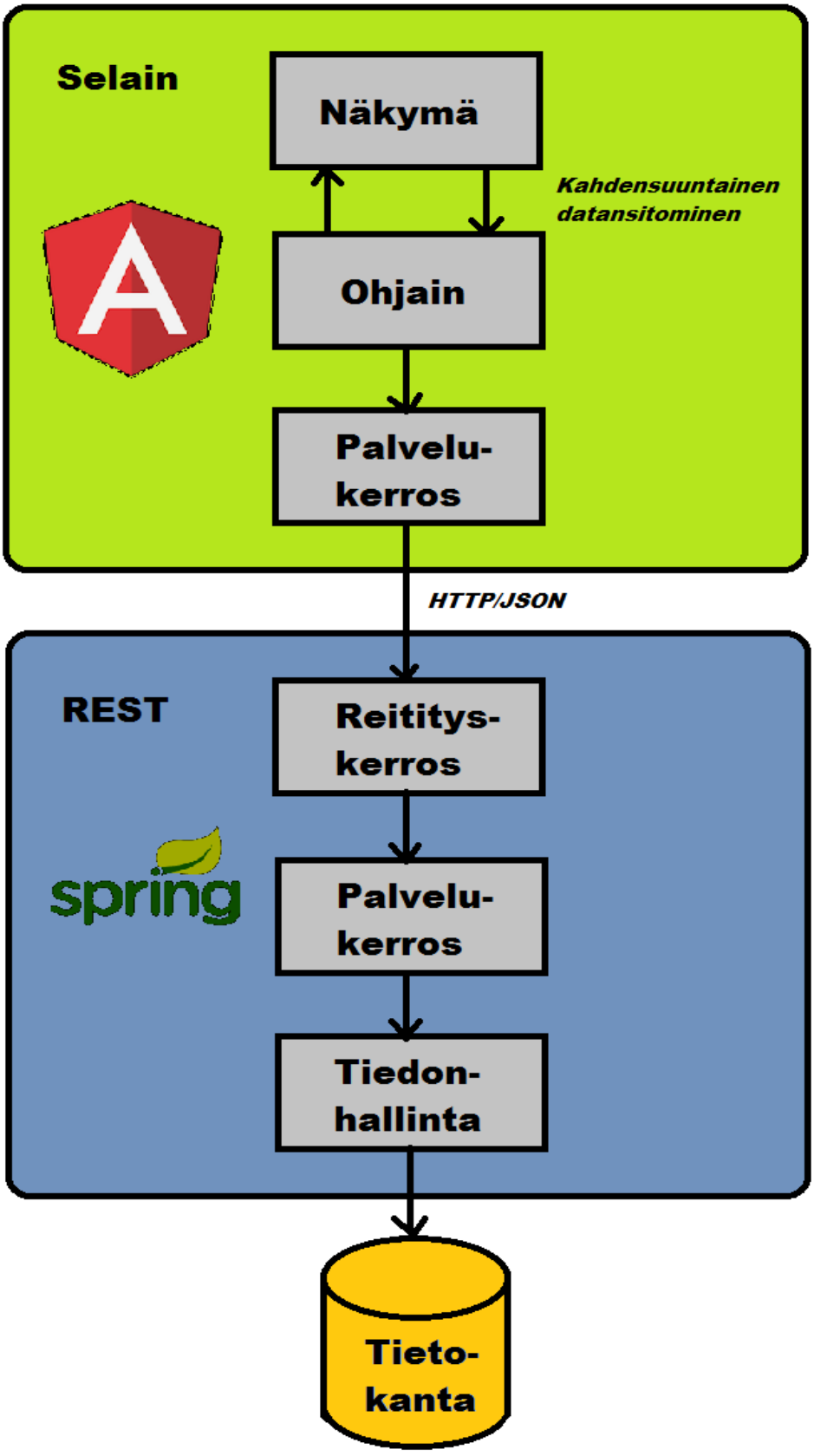
muuttuja määritellyn mukaisesti, jolloin myös tekstikenttä sekä näkymän tuloste päivittyvät automaattisesti.

Palvelut

Palvelut ovat toiminnallisuutta sisältäviä singletoneja eli objekteja, jotka luodaan sovelluksessa vain kerran. AngularJS sisältää paljon hyödyllisiä valmiita palveluita, mutta usein tulee myös tarve kirjoittaa omia palveluita. Palveluihin on hyvä sijoittaa sovelluksen sellainen toiminnallisuus, jota halutaan käyttää useaan otteeseen eri puolilla sovellusta. Palveluita on mahdollista injektoida ohjaimien tai toisten palveluiden käyttöön. [18; 25.]

3.7 Spring MVC ja AngularJS yhdessä

Perinteisissä Spring MVC -sovelluksissa HTML-mallipohjat luodaan palvelinpuolella aiemmin esitetyn prosessin mukaisesti tähän tarkoitetuilla tekniikoilla, kuten JSP:llä (Java Servlet Pages) tai Thymeleafilla. Siirryttäessä käyttämään asiakaspuolella tapahtuvaan mallipohjien luontiin perustuvaa AngularJS:ää, jaetaan verkkosovellus oikeastaan kahdeksi eri sovellukseksi: asiakaspuolen- ja palvelinpuolen-sovelluksiksi. AngularJS sovellus siis toimii itsenäisesti pyörien asiakkaan selaimessa ja kommunikoi Spring MVC -sovelluksen tarjoamien palvelinpuolen palveluiden kanssa. Spring sovellus voi näin toimia REST-rajapintana asiakaspuolen-sovellukselle. [26.]



Kuva 15. Spring MVC + AngularJS -sovelluksen arkkitehtuuri [27].

Kuten kuvasta 15 huomataan, palvelinpuolen rakenne voidaan kuvata käyttäen kerroksia ja näin jakaa toiminta selkeisiin osiin. REST-rajapintana toimivalla Spring MVC -sovelluksella voidaan ajatella olevan seuraavanlaiset kolme kerrosta:

- Reitityskerros (Routing Layer): Reitityskerroksen tarkoituksena on määrittää, mitkä REST-päätepisteet vastaavat mitkäkin URI:a ja miten HTTP-pyynnön mukana tulevat parametrit luetaan.
- Palvelukerros (Service Layer): Palvelukerros sisältää sovelluksen logiikan.
- Tiedonhallintakerros (Persistence Layer): Tiedonhallintakerroksen tehtävänä on kommunikoida sovelluksen tietovaraston kanssa. [27.]

Asiakaspuolen AngularJS sovellus jakautuu myös kerroksiin jo aiemmin esitetyn MVC-arkkitehtuurin tyylisesti ja kommunikoi palvelinpuolen REST-rajapinnan kanssa asiakapuolen palvelukerroksen (Frontend Services Layer) välityksellä. Näin yhdistämällä asiakas- ja palvelinpuoli muodostuu kuvan 15 mukainen arkkitehtuurikokonaisuus.

3.8 Google Charts API

Google Charts on Googlen kehittämä ja ylläpitämä datan visuaaliseen esittämiseen verkkosovelluksissa tarkoitettu ohjelmointirajapinta. Erilaisia kaaviotyyppejä löytyy paljon ja niiden ulkoista olemusta voidaan muokata halutun näköiseksi lukuisilla asetuksilla. Kaaviot määritellään JavaScript luokkien ilmentyminä ja ne ovat hyvin interaktiivisia. Kaaviot renderöidään käyttäen HTML5 ja SVG teknologioita, jonka ansiosta voidaan tarjota hyvä tuki eri selaimille ja alustoille. [28.]

3.9 Bootstrap

Bootstrap on avoimen lähdekoodin front-end (HTML, CSS ja JavaScript) -sovelluskehys, jonka tarkoituksena on helpottaa ja nopeuttaa verkkosovelluksien kehitystä. Bootstrap tarjoaa laajan työkaluvalikoiman joustavien ja responsiivisten käyttöliittymien toteuttamiseen. [29.]

4 Toteutus

Työn toteuttaminen alkoi sovelluksen kehitykseen tarvittavien teknologioiden selvitystyöllä ja niiden ominaisuuksiin tutustumisella. Kun halutut teknologiat olivat selvillä, jatkettiin työtä kehitysympäristön laittamisella kuntoon, joka käytännössä tarkoitti projektin luomista kehitystyökaluun ja projektin käyntiin saamisen kannalta oleellisteen kirjasto-, sekä sovelluskehys-riippuvuuksien määritystä. Projektin riippuvuuksien hallintaan käytettiin projektienhallintaan tarkoitettua Mavenia.

Selvitystyön ja kehitysympäristön valmistelun jälkeen lähdettiin sovellusta toteuttamaan sen määrittelyiden mukaisesti valituilla teknologioilla. Varsinaisen ohjelmointityön tuloksena luotiin AngularJS-sovelluskehyksellä SPA (Single Page Application) -sovellus, joka käyttää Spring-sovelluskehyksellä luotua REST-rajapintaa resurssien hallintaan. Sovelluksen käyttöliittymästä voidaan kätevästi hallinnoida sovelluksen eri toimintoja. Tässä luvussa on tarkoitus esitellä toteutetun sovelluksen kannalta keskeisiä toimintoja, sovelluksen rakennetta ja siihen toteutettua käyttöliittymää.

4.1 SSH-yhteydet

Ohjelmointityö alkoi SSH-yhteyksien toteuttamiseksi tarvittavien menetelmien selvityksellä, toteutuksella ja testauksella. SSH-yhteyksien luonti käyttäen JSch-kirjastoa osoittautui hyvin yksinkertaiseksi, kuten kuvan 16 esimerkistä voidaan huomata.

```
jSch = new JSch();
// salaisen avaimen sisältävän tiedoston sijainnin määritys
jSch.addIdentity(privateKeyFilePath);
// SSH-yhteyden muodostaminen määritettyyn kohteeseen(host)
session = jSch.getSession(sshUsername, sshHost, port);
session.connect();
// porttiohjauksen määritys
session.setPortForwardingL(lport, localhost, rport);
```

Kuva 16. Esimerkki SSH-yhteyden luonnista ja porttiohjauksesta JSch-kirjastolla.

Esimerkissä luodaan SSH-yhteys palvelimeen käyttäen julkiseen avaimeen perustuvaa autentikointia määrittämällä salaisen avaimen sisältävän tiedoston sijainti. Esimerkissä

myös näytetään, kuinka voidaan luoda porttiohjaus paikallisen portin (lport) ja etäportin (rport) välille. Porttiohjausta käytettiin sovelluksessa MySQL-tunneleiden luomiseen.

4.2 Tietokanta ja REST-rajapinta

Sovellukseen tarvittiin mahdollisuus tallentaa siinä käytettäviä resursseja, kuten palvelinmäärityksiä, kyselyitä, kaavioita ja tilastoja. Sovellukselle määritettiin tähän tarkoitukseen MySQL-pohjainen MariaDB-relaatiotietokantajärjestelmä. Tietokannan rakenne ja varsinaisten tietokantataulujen luonti hoidettiin Spring Data JPA -sovellukskehyksen avulla käyttäen Hibernatea, jonka avulla Java-luokat ja niiden väliset yhteydet voidaan kuvata tietokantaan käyttäen JPA-annotaatioita.

Resurssien hallintaa varten palvelinpuolelle luotiin Spring MVC sovelluskehyksen puitteissa REST-rajapinta. Osalle sovelluksessa oleville resursseille luotiin REST-päätepisteet, joiden avulla voidaan suorittaa mm. luonti-, haku, päivitys- ja poisto-operaatiot.

Kyseinen toiminallisuus voidaan saavuttaa seuraavien askeleiden avulla

- Luodaan domain-luokka, joka pitää sisällään resurssin tiedot.
- Luodaan tiedonhallintakerros, joka vastaa resurssin tietokantaoperaatioista.
- Luodaan palvelukerros, joka sisältää sovelluksen toimintalogiikan ja eristää tiedonhallinnan reitityskerroksesta.
- Luodaan reitityskerros, joka prosessoi http-pyynnöt ja palauttaa oikeanlaiset vastaukset asiakaspuolelle. [30.]

Seuraavaksi käydään läpi tämänkaltaisen toteutuksen rakennetta käyttäen esimerkkinä sovellukseen toteutettua Server (palvelin) -resurssia. Luvussa näytetään myös, kuinka asiakaspuolelta on toteutettu kommunikointi REST-rajapinnan kanssa.

4.2.1 Domain-luokat

Domain-luokat ovat POJO (Plain Old Java Object) -objekteja, ja ne sisältävät esityksen resurssista kuvan 17 tapaan.

```
public class Server {

    private long id;
    private String name;
    private String sshHost;
    private Integer sshPort;
    private String sshUsername;
    private Integer mysqlRemotePort;
    private String mysqlUsername;
    private String mysqlPassword;
    private Set<Installation> installations;

    public Server() {
    }

    // Getterit ja setterit...

}
```

Kuva 17. Server-resurssin Domain-luokka.

Esimerkistä on poistettu selkeyden vuoksi JPA-annotaatiot, joiden avulla luokan kentät ja yhteydet kuvataan tietokantaan.

4.2.2 Tiedonahallintakerros

Hyödynnettäessä Spring Data JPA:ta, toteuttaakseen tietokantaan kohdistuvat normaalit CRUD-toiminnot eli luonti-, haku-, päivitys- ja poisto-operaatiot resursseille riittää kuvan 18 mukaisen rajapinnan luonti.

```
public interface ServerRepository extends CrudRepository<Server, Long> {
}
```

Kuva 18. Server-tietovaraston rajapinta.

Esimerkissä luotu rajapinta periytyy kuvassa 19 näkyvästä Spring Data JPA:n CrudRepository-rajapinnasta, joka tarjoaa useita hyödyllisiä valmiita metodeja.

```
public interface CrudRepository<T, ID extends Serializable> extends Repository<T, ID> {
    <S extends T> S save(S var1);
    <S extends T> Iterable<S> save(Iterable<S> var1);
    T findOne(ID var1);
    boolean exists(ID var1);
    Iterable<T> findAll();
    Iterable<T> findAll(Iterable<ID> var1);
    long count();
    void delete(ID var1);
    void delete(T var1);
    void delete(Iterable<? extends T> var1);
    void deleteAll();
}
```

Kuva 19. Spring Data JPA CrudRepository-rajapinta.

Halutessa voitaisiin luotuun rajapintaan kuitenkin lisätä esimerkiksi omia hakuehtoja sisältäviä metodeja. Mitään konkreettista toteutusta luodulle rajapinnalle ei tarvitse tehdä, koska Spring Data JPA huolehtii tästä automaattisesti.

4.2.3 Palvelukerros

Palvelukerrosta voidaan pitää sovelluksen ytimenä, koska se pitää sisällään sovelluksen toimintalogiikan. Yleisesti palvelut käyttävät tiedonhallintakerrosta, koska useasti suoritettavat toimenpiteet liittyvät tietokantaan. Palvelut voivat kutsua myös tarvittaessa toisia palveluita, koska ne sijaitsevat samassa kerroksessa. Server-palvelulle luodaan rajapinta ja sen toteuttava luokka kuvan 20 mukaisesti. [31.]

```
public interface ServerService {
    Iterable<Server> findAll();
    Server create(Server server);
    Server update(long id, Server server);
    Server findOne(long id);
    void delete(long id);
}

@Service
@Transactional
public final class ServerServiceImpl implements ServerService {
    @Autowired
    private ServerRepository serverRepository;

    @Override
    @Transactional(readOnly = true)
    public Iterable<Server> findAll() {
        return serverRepository.findAll();
    }

    @Override
    public Server create(Server server) {
        return serverRepository.save(server);
    }

    @Override
    public Server update(long id, Server server) {
        Server updated = serverRepository.findOne(id);
        updated.setSshHost(server.getSshHost());
        updated.setName(server.getName());
        updated.setSshPort(server.getSshPort());
        updated.setSshUsername(server.getSshUsername());
        updated.setMysqlRemotePort(server.getMysqlRemotePort());
        updated.setMysqlUsername(server.getMysqlUsername());
        updated.setMysqlPassword(server.getMysqlPassword());
        return serverRepository.save(updated);
    }

    @Override
    @Transactional(readOnly = true)
    public Server findOne(long id) {
        return serverRepository.findOne(id);
    }
    @Override
    public void delete(long id) {
        serverRepository.delete(id);
    }
}
```

Kuva 20. Server-palvelun rajapinta ja toteutus.

Kuten voidaan huomata, kyseisellä palvelulla ei ole kauheasti varsinaista toimintalogiikkaa, vaan se lähinnä delegoi toimenpiteet tiedonhallintakerrokseen.

4.2.4 Reitityskerros

Reitityskerros määrittää sovellukselle REST-päätepisteet, jolla asiakaspuolelta voidaan päästä käsiksi resursseihin. Tätä varten luotiin kuvan 21 mukainen ohjain (controller) -luokka prosessoimaan tulevat HTTP-pyynnöt.

```
@RestController
@RequestMapping("/servers")
public class ServerController {

    @Autowired
    private ServerService serverService;

    @RequestMapping(method = RequestMethod.GET)
    public Iterable<Server> list() {
        return serverService.findAll();
    }

    @RequestMapping(method = RequestMethod.POST)
    public Server create(@RequestBody @Valid Server server){
        return serverService.create(server);
    }

    @RequestMapping(method = RequestMethod.GET, value = "{id}")
    public ResponseEntity<Server> get(@PathVariable long id){
        Server server = serverService.findOne(id);
        if(server == null){
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
        return new ResponseEntity<>(server, HttpStatus.OK);
    }

    @RequestMapping(method = RequestMethod.PUT, value = "{id}")
    public Server update(@PathVariable long id, @RequestBody @Valid Server server){
        return serverService.update(id, server);
    }

    @RequestMapping(method = RequestMethod.DELETE, value = "{id}")
    public void delete(@PathVariable long id){
        serverService.delete(id);
    }
}
```

Kuva 21. Server-resurssin ohjainluokka.

Taulukossa 1 esitetään luodut REST-päätepisteet ja niiden kuvaukset. Vastaavanlaiset päätepisteet on luotu myös kaikille muille resursseille, joita täytyy voida käsitellä asiakaspuolelta.

Taulukko 1. REST-päätepisteet ja niiden kuvaukset.

Metodi	**URI**	**Kuvaus**
GET	/servers	Hakee kaikki Server-resurssit järjestelmästä
GET	/servers/{id}	Hakee parametrina annetua id:tä vastaavan Server-resurssin
POST	/servers	Luo uuden Server-resurssin, jonka tiedot löytyvät pyynnön rungosta
PUT	/servers/{id}	Päivittää olemassa olevan Server-resurssin tiedot
DELETE	/servers/{id}	Poistaa olemassa olevan Server-resurssin järjetelmästä

4.2.5 Asiakaspuolen palvelukerros

Resursseihin päästän käsiksi todella yksinkertaisesti REST-rajapinnan kanssa kommunikointiin tarkoitetun AngularJS:n ngResource-moduulin avulla.

```
// AngularJS-palvelu, jonka kautta päästään käsiksi REST-rajapinnan Server-resurssiin
saasAnalyzerApp.factory('Server', function ($resource) {
    return $resource('/servers/:id', {id: "@id"}, {
        update: {
            method: 'PUT'
        }
    });
});
// AngularJS-ohjain, johon injektoidaan Server-palvelu
saasAnalyzerApp.controller('ServerCtrl', ['Server', function (Server) {
var server = {
  "id": 1,
  "name": "Testipalvelin",
  // ...
}
// esimerkki Server-palvelun käytöstä
// POST-kutsu /servers/{id}
Server.save({}, server);
// GET-kutsu /servers/{id}
Server.get({id: 5});
// GET-kutsu /servers
Server.query();
// PUT-kutsu /servers/{id}
Server.update({id: 5}, server);
// DELETE-kutsu /servers/{id}
Server.delete({id: 5});
}
```

Kuva 22. Esimerkki AngularJS-palvelusta ja sen käytöstä AngularJS-ohjaimessa.

Kuvassa 22 on esimerkki, jossa on toteutettu ngResource-moduulin avulla AngularJS-palvelu Server-resurssin hallintaan. Kuvassa on myös esimerkki, kuinka luotu palvelu voidaan injektoida AngularJS-ohjaimeen ja kuinka sillä voidaan kutsua REST-rajapinnan päätepisteitä.

4.3 Tilastojen päivittäminen

Sovellukseen kerättävää tietoa täytyy pystyä tarvittaessa päivittämään ja tätä varten luotiin päivitysrutiini, joka vastaa kaikkien asiakkaiden ja yleistilastojen päivityksestä. Päivitysrutiinin voi tarvittaessa käynnistää sovelluksen käyttöliittymästä käsin.

Päivitysrutiini on käytännössä metodi, joka muodostuu seuraavista askelista

- Muodostetaan yhteys palvelimeen.
- Suoritetaan SQL-kyselyt kaikkien asiakkaiden kantoihin.

- Tallennetaan saadut tunnusluvut ja totuusarvot sovelluksen tietokantaan.
- Luodaan taulukkokyselyistä ja tunnuslukutaulukoista määritetyt kaaviot, joista tallennetaan JSON-muotoinen esitys tietokantaan.
- Suoritetaan edelliset toimenpiteet kaikilla muilla palvelimilla.
- Luodaan yhdistetyt tilastot, sekä kaaviot ja tallennetaan ne tietokantaan.

Kaavioista muodostetaan siis jo päivitysrutiinin yhteydessä Google Charts API:n haluamassa muodossa oleva JSON-muotoinen esitys kaavioista. Tämän ansiosta kaavioiden esittäminen käyttäjän pyynnöstä on nopeampaa, koska kaavioita ei tarvitse aina generoida uudelleen tietokannan datasta.

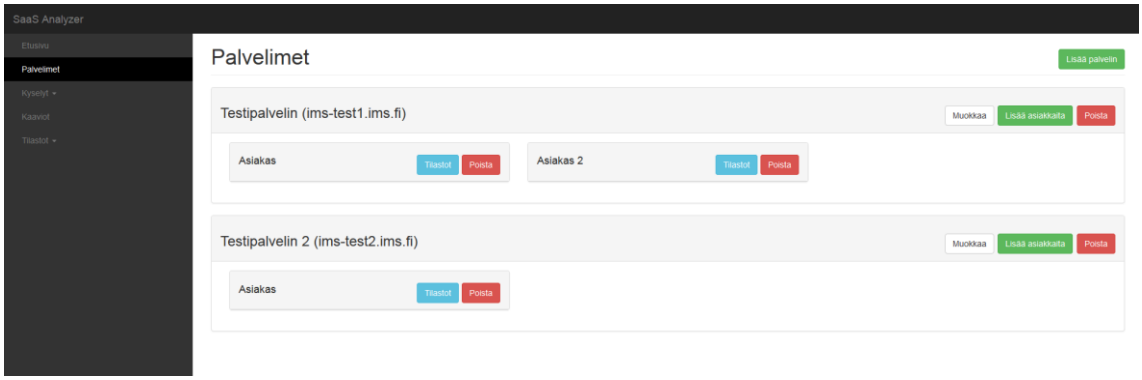
4.4 Käyttöliittymä

Projektissa sovellukselle luotiin asiakaspuolelle selainpohjainen käyttöliittymä, josta voidaan hallita sovelluksen keskeisiä toimintoja ja tarkastella määritettyjä tilastoja. Käyttöliittymän visuaalinen ilme on käytännössä, Google charts -kaavioita lukuun ottamatta, rakennettu Bootstrapin tarjoamilla perustyökaluilla. Seuraavaksi esitellään tarkemmin sovelluksen eri osioiden toiminnallisuutta ja ulkoasua.

4.4.1 Palvelimien ja asiakkaiden hallinta

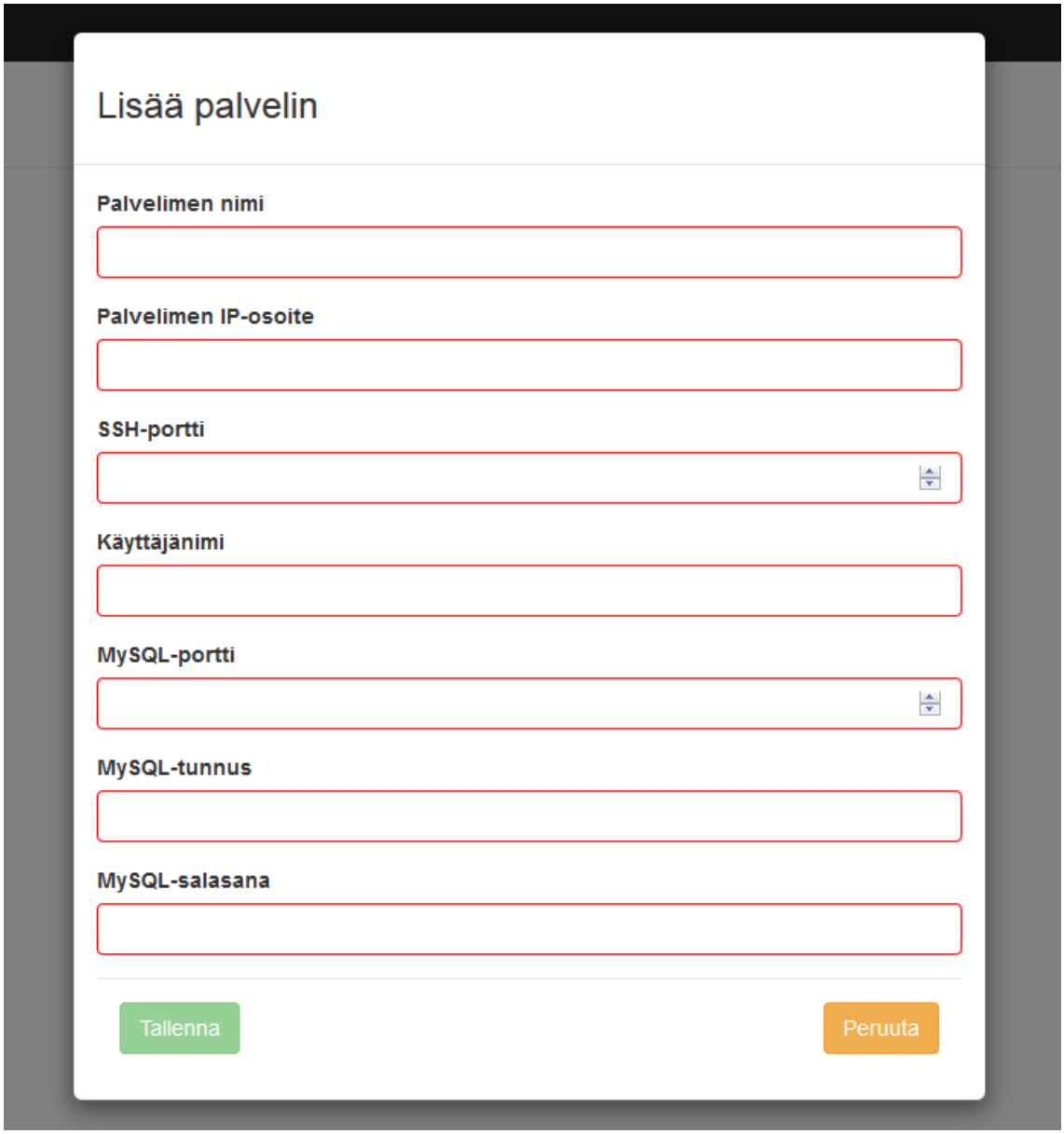
Yrityksen palvelimien ja niiden sisältämien asiakkaiden suuren lukumäärän vuoksi sovellukseen haluttiin toteuttaa helppo tapa hallita analysoinnin kohteena olevia asiakkaita. Analysoitavien tietokantojen määrittäminen on tärkeää myös siksi, että palvelimet saattavat sisältää esimerkiksi tuleville asiakkaille kokeilua varten luotuja asennuksia, joiden tilastoja ei välttämättä haluta turhaan kerätä.

Sovellukseen luotiin kuvan 23 mukainen näkymä, jossa kaikki järjestelmään lisätyt palvelimet ja asiakkaat näkyvät listattuna.



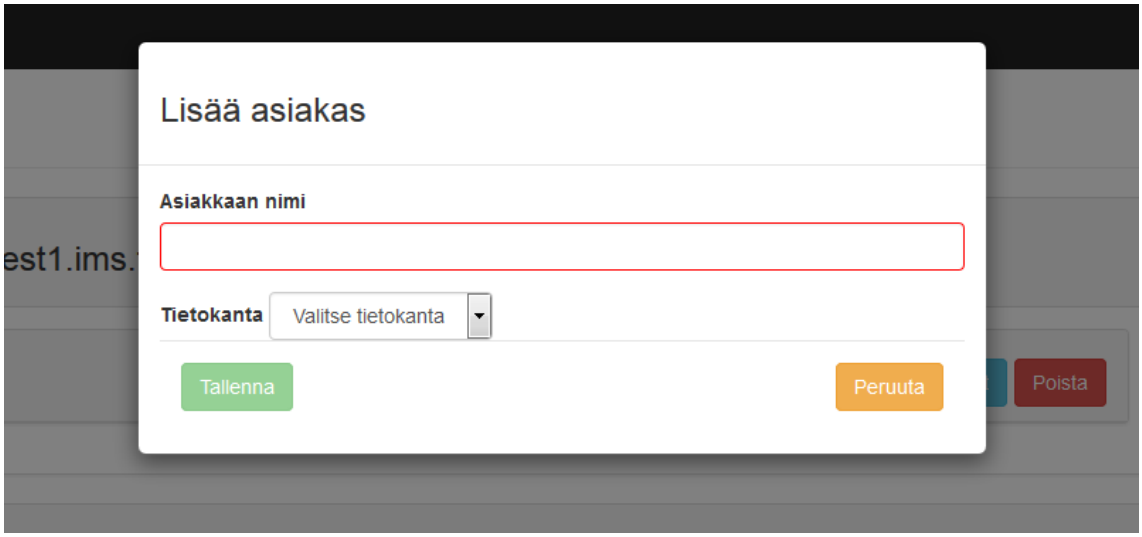
Kuva 23. Palvelinlistaus.

Lisätessä tai muokatessa palvelimia avataan kuvan 24 mukainen modaalinen ikkuna, jossa palvelimelle voidaan määrittää seuraavat tiedot; nimi, ip-osoite, SSH-yhteydessä käytetty tunnus, portti, johon SSH-yhteys luodaan, MySQL-portti, MySQL-tunnus ja MySQL-salasana.



Kuva 24. Palvelinten lisäykseen ja muokkaukseen tarkoitettu modaalinen ikkuna.

Palvelimen lisäyksen jälkeen sille voidaan määrittää asiakkaita painamalle Lisää asiakkaita -painiketta, jolloin avautuu kuvan 25 mukainen modaalinen ikkuna, jossa määritetään asiakkaan nimi ja valitaan sitä vastaava tietokanta palvelimelta löytyvistä tietokannoista.

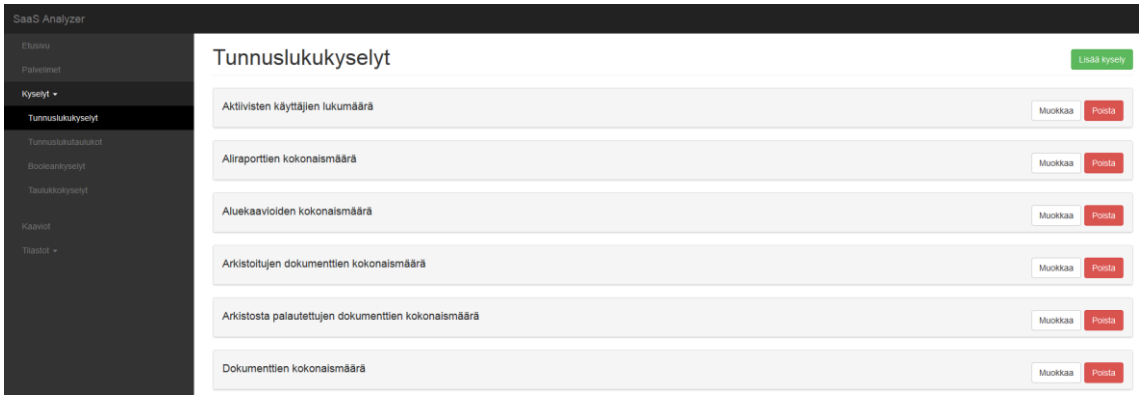


Kuva 25. Asiakkaiden lisäykseen tarkoitettu modaalinen ikkuna.

Asiakkaan tilastoja pääsee tarkastelemaan painamalla asiakkaan nimen vierestä löytyvää Tilastot-painiketta, ja asiakkaan voi halutessaan poistaa tilastoinnista painamalla Poista-painiketta. Poistamalla palvelimen myös kaikki sille määritetyt asiakkaat poistuvat tilastoinnista.

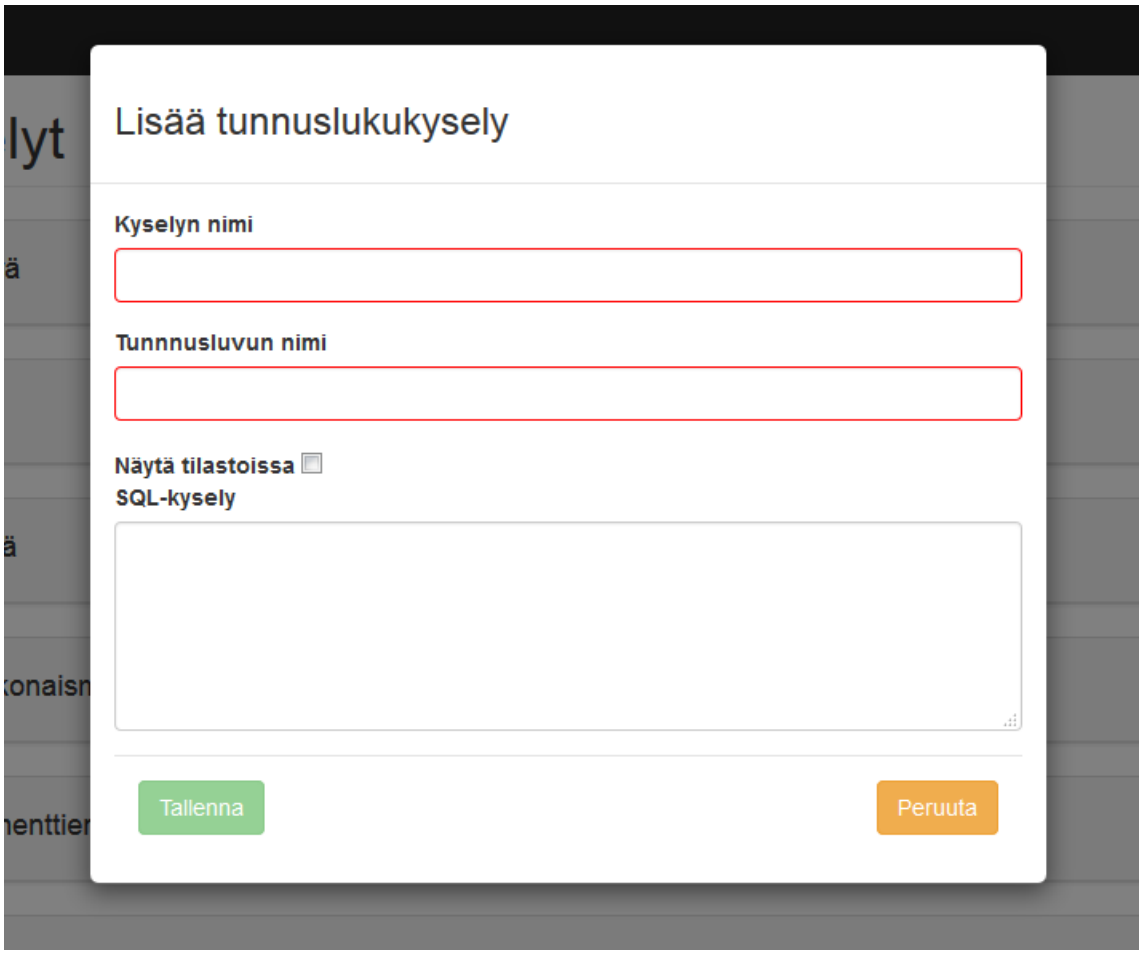
4.4.2 Kyselyiden hallinta

Sovelluksessa tuli olla mahdollista hallinnoida kolmea eri tyyppiä olevia kyselyitä; tunnusluku-, boolean- ja taulukkokyselyitä. Kaikille kyseltytyypeille luotiin omat hallinnointinäkymät, jotta ne olisivat selkeästi erotettavissa toisistaan.



Kuva 26. Tunnuslukukyselyjen listaus.

Kaikki kuvan 26 mukaiset kyselynäkymät ovat hyvin samankaltaisia ja niiden toimintoja ovat kyselyiden listaus, uusien kyselyiden lisäys sekä jo olemassa olevien muokkaus ja poisto.

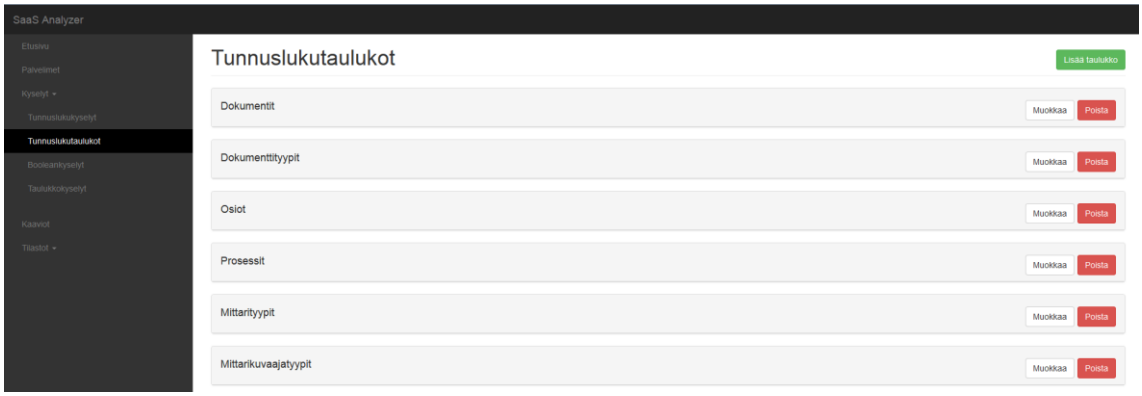


Kuva 27. Tunnuslukukyselyiden lisäykseen tarkoitettu modaalinen ikkuna.

Lisätessä tai muokatessa kyselyä avataan kuvan 27 mukainen modaalinen ikkuna, jossa voidaan määrittää kyselylle nimi, SQL-lauseke. Tunnusluku- ja booleankyselyille voidaan myös määrittää valinta, että näytetäänkö kyseisen kyselyn tuottama tulos tilastonäkymässä omana informaatiolaatikkona.

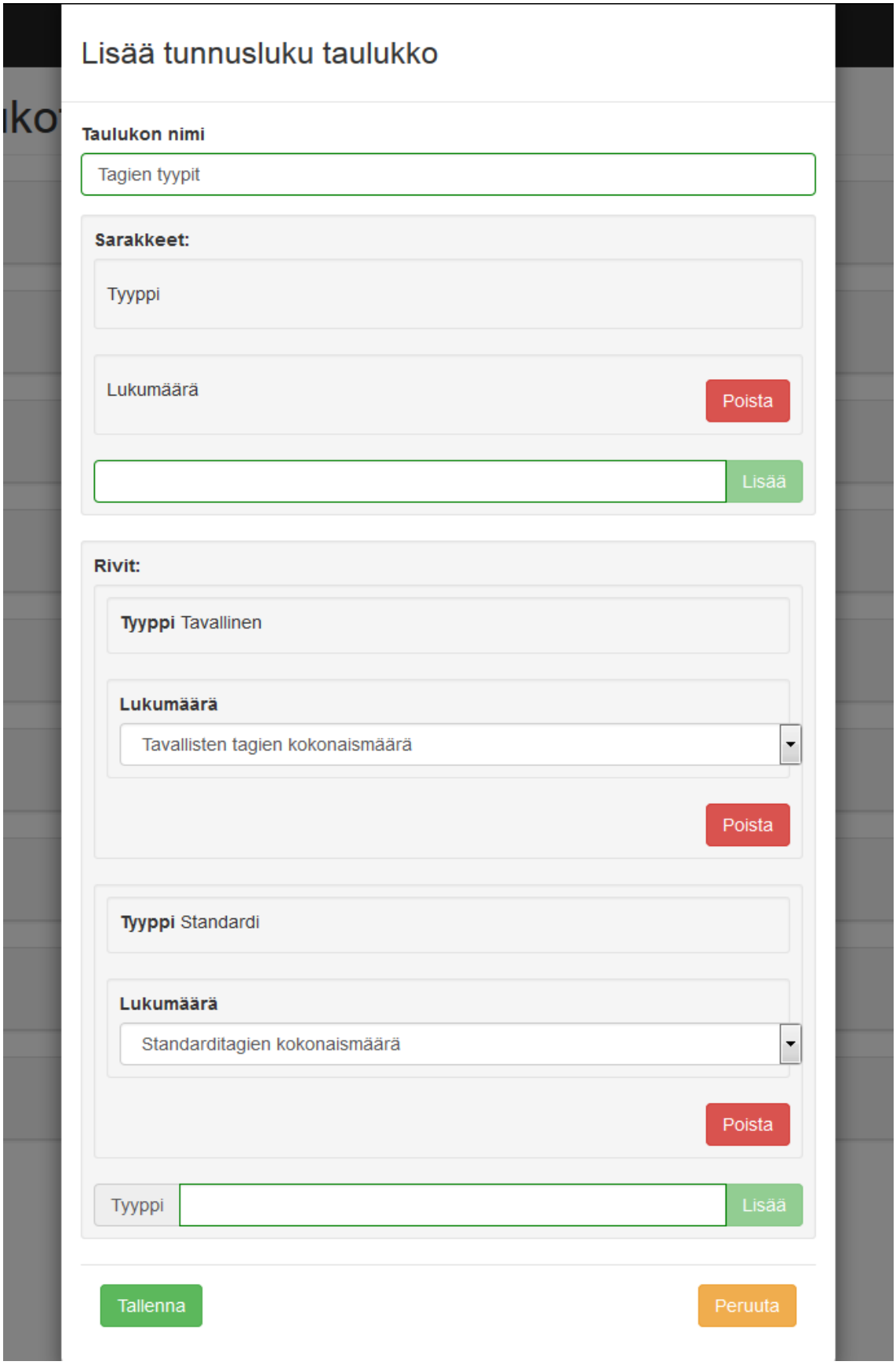
4.4.3 Tunnuslukutaulukoiden hallinta

Tunnuslukukyselyiden tuottamista yksittäisistä tunnusluvuista koostuvien tunnuslukutaulukoiden hallinnointia varten toteutettiin sovellukseen oma kuvan 28 mukainen näkymä.



Kuva 28. Tunnuslukutaulukoiden listaus.

Näkymässä listataan kaikki järjestelmästä löytyvät tunnuslukutaulukot ja niitä voidaan lisätä, muokata tai poistaa. Lisätessä tai muokatessa kyselyä avataan kuvan 29 mukainen modaalinen ikkuna, jossa voidaan määrittää tunnuslukutaulukolle nimi sekä määrittää siihen uusia sarakkeita ja rivejä.


Lisää tunnusluku taulukko
Taulukon nimi
Tagien tyypit
Sarakkeet:
Tyyppi
Lukumäärä
Poista
Lisää
Rivit:
Tyyppi Tavallinen
Lukumäärä
Tavallisten tagien kokonaismäärä
Poista
Tyyppi Standardi
Lukumäärä
Standarditagien kokonaismäärä
Poista
Tyyppi
Lisää
Tallenna
Peruuta

Kuva 29. Tunnuslukutaulukoiden rakentamiseen tarkoitettu modaalinen ikkuna.

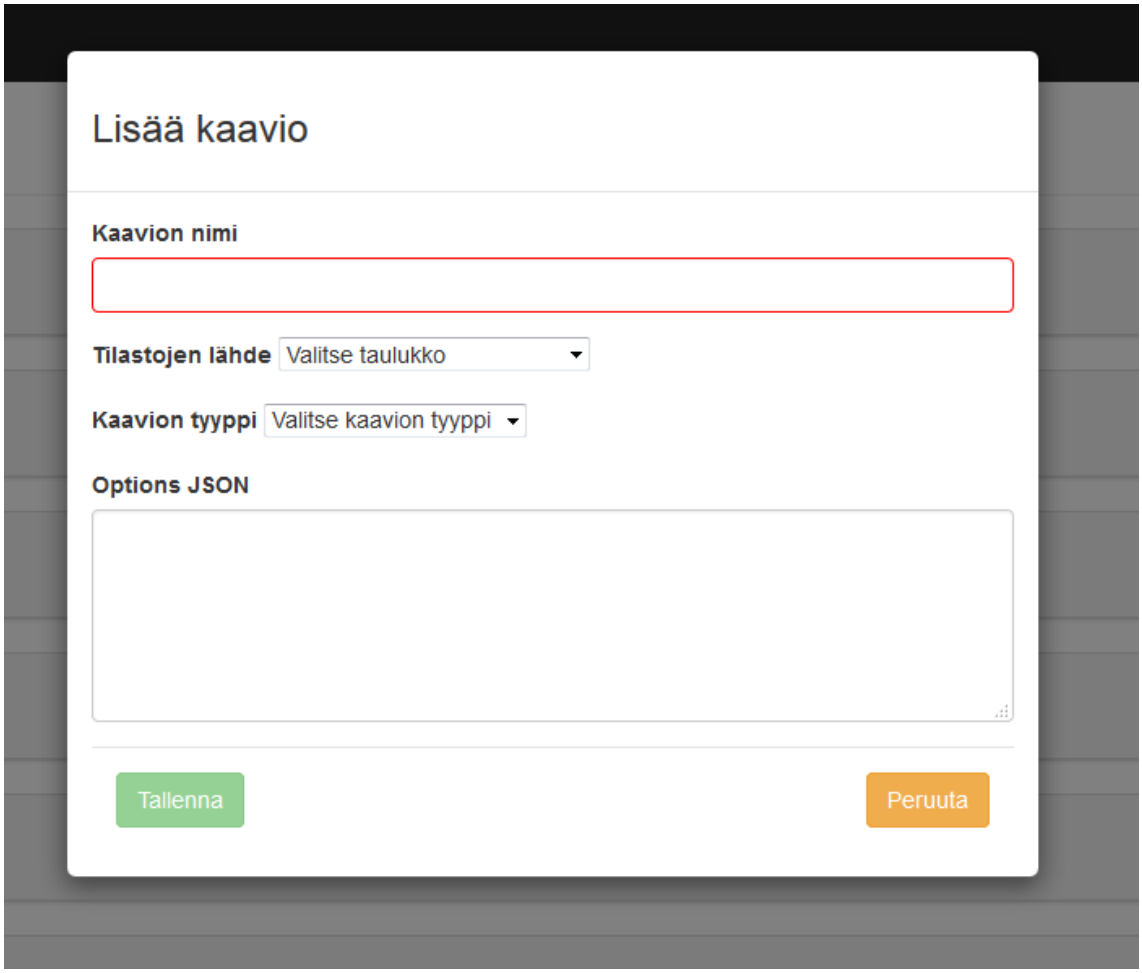
Ensimmäiseksi tunnuslukutaulukkoa muodostettaessa tulee määritellä ns. otsikkosarake, joka määrittää kaavion x-akselilla esiintyvien datasarjojen nimikkeet ja on tekstimuotoinen. Otsikkosarakkeen lisäyksen jälkeen voidaan tunnuslukutaulukkoon lisätä uusia datasarjoja eli rivejä määrittelemällä datasarjalle nimike. Otsikkosarakkeen jälkeen lisättävät sarakkeet ovat ns. tunnuslukusarakkeita, joihin voidaan liittää datasarjakohtaisesti jokin jo luoduista tunnuslukukyselyistä, kuten kuvassa 29 on tehty.

4.4.4 Kaaviomääritysten hallinta

Sovellukseen lisätyistä tunnuslukutaulukoista ja valmiin taulukon tuottavista taulukkokyselyistä voidaan kuvan 30 mukaisesta kaavioiden hallintanäkymästä määrittää erityyppisiä Google Chart -kaavioita esitettäväksi tilastoinnissa. Kaavioiden hallintanäkymässä listataan järjestelmästä löytyvät kaaviomääritykset. Niitä voidaan myös lisätä, muokata ja poistaa.

Kuva 30. Kaaviomääritysten listaus.

Lisätessä tai muokatessa kaaviomääritystä avataan kuvan 31 mukainen modaalinen ikkuna, jossa voidaan asettaa kaaviomääritykselle nimi, datan lähteenä toimiva taulukko, kaaviotyyppi ja lisämäärityksiä Google Chart -kaaviolle JSON-muodossa.



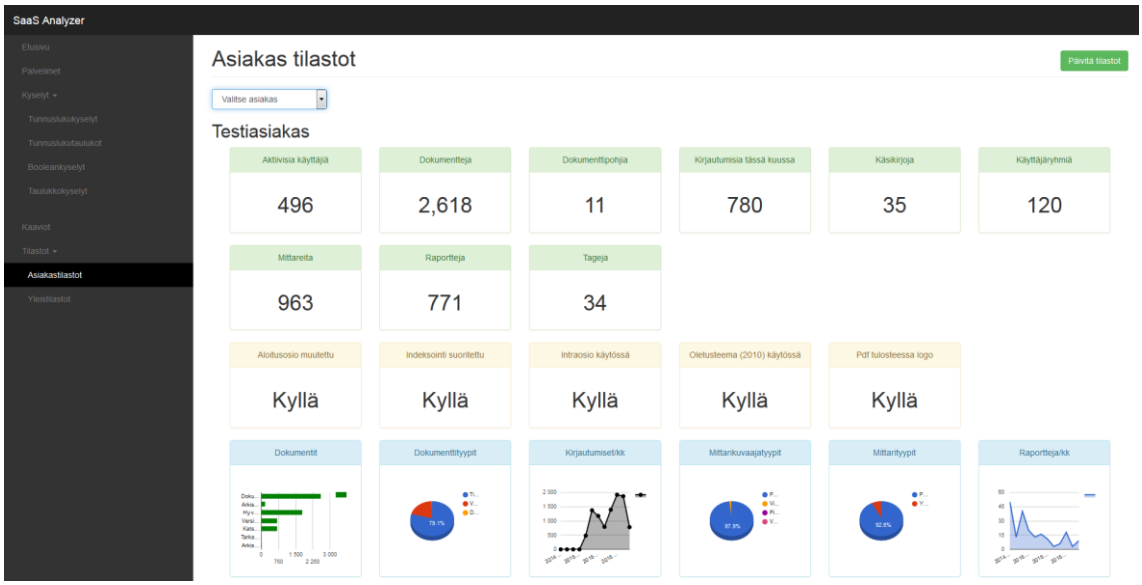
Kuva 31. Kaaviomääritysten lisäykseen ja muokkaukseen tarkoitettu modaalinen ikkuna.

Tunnuslukutaulukkojen rakenteen puitteissa voidaan sovelluksessa tukea seuraavia Google Chart -kaaviotyyppejä; Area Chart (aluekaavio), Bar Chart (palkkikaavio), Column Chart (pylväskaavio), Combo Chart (yhdistelmäkaavio), Gauge Chart (nopeusmittarikaavio), Histogram (histogrammikaavio), Line Chart (viivakaavio), Pie Chart (piirraskaavio) ja Table Chart (taulukkokaavio).

4.4.5 Tilastot

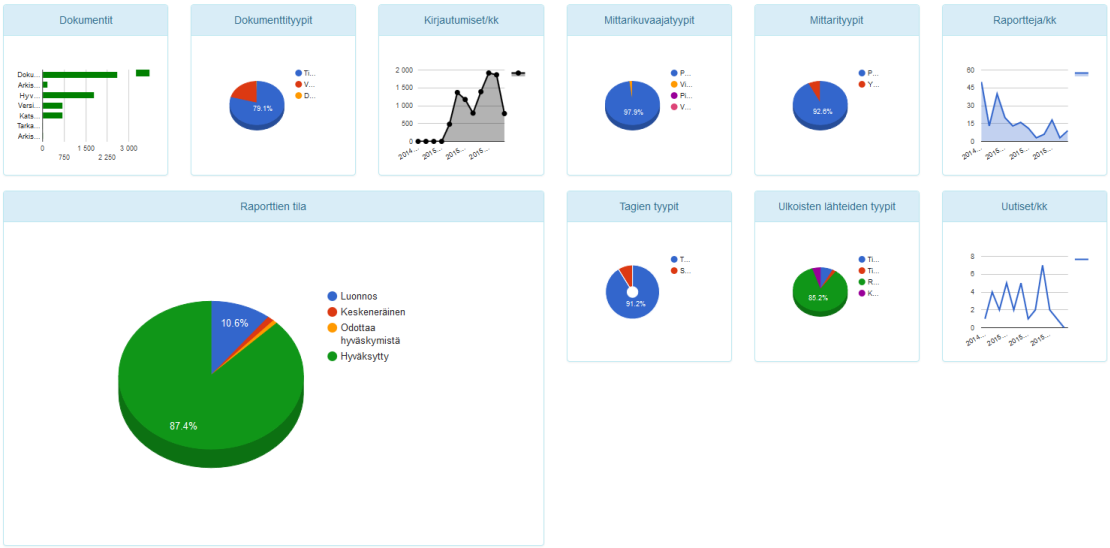
Tilastojen esittämistä varten on sovellukseen toteutettu kaksi eri näkymää; asiakaskohtaiset tilastot ja yleistilastot. Molemmissa näkymissä on pohjalla sama ajatus esittää tulokset tilastoissa näkymään määritetyistä tunnusluvuista, totuusarvoista ja kaaviomäärityksistä. Myös sovelluksen tilastojen päivitysrutiini voidaan käynnistää tilastointinäkymistä.

Kuvan 32 mukaisessa asiakaskohtaisessa tilastonäkymässä esitetään asiakaskohtaisesti tilastoissa näkymään määritetyt tunnusluvut, totuusarvot ja kaaviot. Tunnusluvut ovat kuvassa näkyvissä vihreän otsakkeen, totuusarvot keltaisen otsakkeen ja kaaviot sinisen otsakkeen omaavissa paneeleissa.



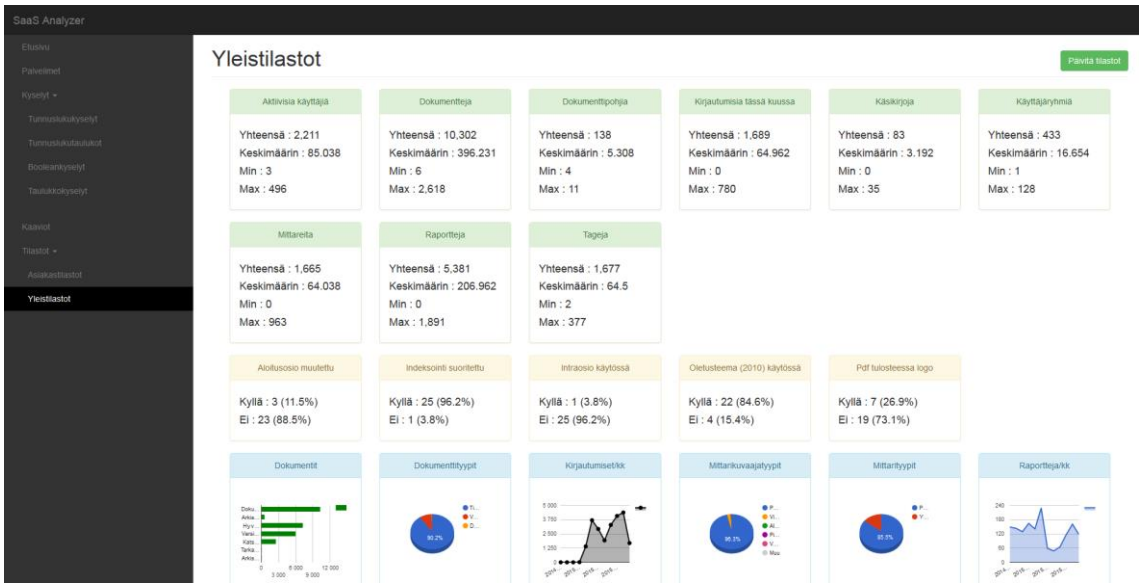
Kuva 32. Asiakaskohtainen tilastonäkymä.

Kaaviot ovat normaalitilassa pienessä muodossa, mutta haluttaessa niitä voidaan kuvan 33 mukaisesti suurentaa tai pienentää klikkaamalla niiden otsaketta. Asiakasta voidaan vaihtaa näkymästä löytyvästä pudotusvalikosta.



Kuva 33. Esimerkki suurennetusta kaaviosta.

Kuvan 34 mukaisessa yleistilastonäkymässä esitetään ensin kaikista asiakaskohtaisista tunnusluvuista seuraavat koostetut tiedot: minimiarvo, maksimiarvo, kokonaissumma ja keskiarvo. Seuraavaksi esitetään asiakaskohtaisten totuusarvojen tuloksista koostetut kyllä ja ei kokonaismäärät, sekä niitä vastaavat prosenttiosuudet. Tämän jälkeen näytetään vielä kaavioista tehdyt koosteet.



Kuva 34. Yleisten tilastojen näkymä.

Yleistilastoista löytyy myös kaksi taulukkoa, jotka sisältävät kaikkien asiakkaiden kaikki tunnusluvut ja totuusarvot. Taulukoista löytyvät myös aiemmin mainitut koostetut tiedot kaikkia arvoja kohden. Taulukoihin on myös mahdollista suorittaa hakuja taulukoiden niistä löytyvän hakukentän avulla.

5 Yhteenveto

Insinöörityön tavoitteena oli luoda työkalu, jonka avulla saataisiin tilastotietoa siitä, kuinka IMS Business Solutions Oy:n asiakkaille tarjoamaa IMS-ohjelmistoa käytetään. Kerättyä tietoa voitaisiin hyödyntää esimerkiksi ohjelmiston tuotekehityksen tehostamisessa ja koulutuksen parantamisessa. Työlle asetettiin selkeät tavoitteet ja määritettiin halutut toiminnallisuudet. Työkalu päätettiin toteuttaa verkkosovelluksena, jolloin tilastoja pääsisi aina halutessaan helposti tarkastelemaan selaimen kautta. Työssä käytettävien teknologioiden valinta jätettiin tekijän päätettäväksi.

Työ alkoikin siihen soveltuvien teknologioiden selvitystyöllä ja niihin perehtymisellä. Sopivien teknologioiden löydyttyä, työn toteutus eteni hyvin suoraviivaisesti ja suuremmilta ongelmilta vältyttiin. Insinöörityön tuloksena luotiin verkkosovellus, jolla tilastojen keräämistä voidaan hallita selainpohjaisen käyttöliittymän kautta. Sovelluksen palvelinpuolelle toteutettiin Spring-sovelluskehyksellä REST-rajapinta, joka palveli asiakaspuolelle toteutettua AngularJS SPA -sovellusta, joka sallii pääsyn käsiksi sovelluksen resursseihin. Sovelluksen käyttöliittymän kautta onnistuu kätevästi tilastoitavien asiakasasennusten, tilastoja tuottavien kyselyiden sekä tilastoille visuaalisen ilmeen antavien taulukoiden ja kaavioiden lisääminen. Tilastoja voidaan sovelluksessa tarkastella sekä asiakaskohtaisesti, että kaikkien asiakastilastojen koostavina yleistilastoina.

Insinöörityön tuloksena tuotettu sovellus vastaa sille asetettuja tavoitteita. Siksi lopputulokseen voidaan olla tyytyväisiä. Kirjoitushetkellä sovellus on ollut käytössä jo muutaman viikon ja on toiminut moitteetta. Analysaattori on saanut positiivisen vastaanoton yrityksessä ja on tarjonnut paljon mielenkiintoisia sekä hyödyllisiä tilastoja. Työkalusta voi olla hyötyä myös muualla, koska käytännössä sen ei tarvitse tietää mitään analysoitavista tietokannoista, joten tarvittaessa sillä voitaisiin tuottaa tilastoja myös mistä tahansa muista MySQL-pohjaisista tietokannoista.

Tuotteen jatkokehitystä on myös jo jonkin verran pohdittu ja tämän kirjoitushetkellä myös hieman toteutettukin. Sovellukseen on toteutettu käyttäjien autentikointi, jonka avulla voidaan rajoittaa osa sovelluksen toiminnallisuudesta, kuten yhteyksien ja kyselyiden määritys vain ylläpitäville henkilöille. Autentikointi on toteutettu käyttäen Kerberos todennusprotokollaa, joka mahdollistaa Windows-ympäristössä käyttäjien todentamisen jo aiemmin Windows-työympäristöön kirjauduttaessa syötettyjen tunnuksien perusteella. Tämän ansiosta sovellusta käyttävien henkilöiden ei siis tarvitse syöttää sovellukseen uudestaan tunnuksia ja salasanoja. Hyvä lisäominaisuus voisi myös olla mahdollisuus tuottaa esimerkiksi PDF-muotoisia raportteja sovelluksen luomista tilastoista.

Lähteet

1 Varanasi B., Belida S. 2015. Spring REST. Apress.

2 Freidrich Todd. 2012. RESTful Service Best Practices. Verkkodokumentti. <http://www.restapitutorial.com/media/RESTful_Best_Practices-v1_1.pdf>. Luettu 20.12.2015.

3 MVC Architecture. Verkkodokumentti. <https://devel-oper.chrome.com/apps/app_frameworks>. Luettu 21.12.2015.

4 inum M. & Serneels K. 2012. Pro Spring MVC: With Web Flow. Apress.

5 Barret J. D. & Silverman R. 2001. SSH, The Secure Shell: The Definitive Guide. California: O'Reilly.

6 SSH. Verkkodokumentti. <https://fi.wikibooks.org/wiki/SSH>. Luettu 6.11.2015.

7 Ferbers Daniel. 2008. About JSch open source project. Verkkosdokumentti. <https://techtavern.wordpress.com/2008/09/30/about-jsch-open-source-project/>. Luettu 6.11.2015.

8 Fink G. & Flatow I. 2014. Pro Single Page Application Development. Apress.

9 Wasson Mike. 2013. ASP.NET - Single-Page Applications: Build Moder, Re-sponsive Web Apps with ASP.NET. Verkkodokumentti. < https://msdn.micro-soft.com/en-us/magazine/dn463786.aspx>. Luettu 13.1.2015.

10 Ho C. & Harrop R. 2012. Pro Spring. Apress.

11 Jenkov Jakob. 2014. Dependency Injection Benefits. Verkkodokumentti. < http://tutorials.jenkov.com/dependency-injection/dependency-injection-be-nefits.html>. Luettu 7.2.2015.

12 Indrotuction to the Spring Framework. Verkkodokumentti. <http://docs.spring.io/spring/docs/current/spring-framework-reference/html/over-view.html>. Luettu 7.2.2015.

13 Deinum M. & Serneels K. 2012. Pro Spring MVC: With Web Flow. Apress.

14 Posa Rambau. 2015. Introduction to Spring Boot. Verkkodokumentti. JournalDev. <http://www.journaldev.com/7969/introduction-to-spring-boot>. Luettu 5.11.2015.

15 Spring Data JPA. Verkkodokumentti. <http://projects.spring.io/spring-data-jpa/>. Luettu 12.11.2015.

16 Green B. & Seshadri S. 2013. AngularJS. California: O'Reilly Media, Inc.

17 Malik Namita. 2014. MVC and MVVM with AngularJS. Verkkodokumentti. <http://codechutney.in/blog/javascript/mvc-and-mvvm-with-angularjs/>. Luettu 20.12.2015.

18 Bégaudeau Stéphane. 2013. Everything you need to understand ti start with AngularJS. Verkkodokumentti. <http://stephanebegaudeau.tumblr.com/post/48776908163/everything-you-need-to-understand-to-start-with>. Luettu 11.11.2015.

19 Developer Guide: Filters Verkkodokumentti. <https://docs.angularjs.org/guide/filter>. Luettu 12.11.2015.

20 Pandeep Sanda. 2014. A Practical Guide to AngularJS Directives. Verkkodokumentti. <http://www.sitepoint.com/practical-guide-angularjs-directives/>. Luettu 8.11.2015.

21 Seshadari. S & Green B. 2014. AngularJS: Up And Running. California. O'Reilly Media, Inc.

22 Developer Guide: What are Scopes?. Verkkodokumentti. <https://docs.angularjs.org/guide/scope>. Luettu 12.11.2015.

23 Developer Guide: Understanding Controllers. Verkkodokumentti. <https://docs.angularjs.org/guide/controller>. Luettu 12.11.2015.

24 Seshadari. S & Green B. 2014. AngularJS: Up And Running. California. O'Reilly Media, Inc.

25 Developer Guide: Services. Verkkodokumentti. <https://docs.angularjs.org/guide/services>. Luettu 12.11.2015.

26 Isvy Michael. 2015. Migrating a Spring Web MVC application from JSP to AngularJS. Verkkodokumentti. <https://spring.io/blog/2015/08/19/migrating-a-spring-web-mvc-application-from-jsp-to-angularjs>. Luettu 11.11.2015.

27 Web App Architecture - the Spring MVC - AngularJS stack. 2015. Verkkodokumentti. <http://blog.jhades.org/developing-a-modern-java-8-web-app-with-spring-mvc-and-angularjs/>. Luettu 8.11.2015.

28 Using Google Charts. Verkkodokumentti. <https://developers.google.com/chart/interactive/docs/>. Luettu 10.11.2015.

29 Bootstrap (front-end framework). Verkkodokumentti. <https://en.wikipedia.org/wiki/Bootstrap_%28front-end_framework%29>. Luettu 11.11.2015.

30 Kainulainen Petri. 2014. Creating a REST API With Spring Boot and MongoDB. Verkkodokumentti. <http://www.petrikainulainen.net/programming/spring-framework/creating-a-rest-api-with-spring-boot-and-mongodb/>. Luettu 11.11.2015.

31 2013. A Comprehensive Example of a Spring MVC Application – Part 2. Verkkodokumentti. <http://community.hpe.com/t5/Software-Developers/A-Comprehensive-Example-of-a-Spring-MVC-Application-Part-2/ba-p/6135443#.VkQu1rYLJA>. Luettu 10.11.2015