Santeri Leppänen

REST-rajapintojen esittely mock-käyttöliittymällä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

21.5.2017

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Santeri Leppänen REST-rajapintojen esittely mock-käyttöliittymällä 25 sivua 21.5.2017
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Lehtori Simo Silander

Insinöörityössä tutustuttiin REST-rajapintoihin. Tavoitteena oli tehdä geneerinen sovellus, jolla pystytään esittämään sidosryhmille rajapinnan kehitystyön tuloksia mock-käyttöliittymällä. Sovelluksen kehityksen pääpainona on, että käyttöliittymän valmistelu on mahdollisimman helppoa, jotta kehitysresursseja ei sidota itse kehitystyön ulkopuolelle liikaa.

Kehitetty sovellus on tehty käyttäen Vaadin-ohjelmistokehystä, joka mahdollistaa sen, että sovellus on kokonaisuudessaan Javalla kirjoitettu. Tämä on tärkeää, koska se mahdollistaa rajapinnan ja mock-käyttöliittymän toteuttamisen samalla ohjelmointikielellä.

Työssä käsitellään REST-arkkitehtuurin alkuperää, määritelmää ja vaatimuksia.
Työn tuotoksena syntyi sovellus, joka täyttää asetetut vaatimukset pääosin.

Avainsanat	REST, REST API, Vaadin-ohjelmistokehys, JSON, Java, mock, Jackson

Author(s) Title Number of Pages Date	Santeri Leppänen Displaying State of REST API Using Mock User Interface 25 pages 21 May 2017
Degree	Bachelor of Engineering
Degree Programme	Information technology
Specialisation option	Software Engineering
Instructor(s)	Simo Silander, Senior Lecturer

This thesis focuses on REST API. The goal was to create a generic software that can display results of interface development work to stakeholders of the system by using a mock user interface. The main aim of the software is to enable the creation of a demo interface as easily as possible. This ensures that as little as possible effort is used outside the main development.

The software developed was made using the Vaadin software framework that enables writing the entire software in Java. This allows the creator of the API backend to write the mock frontend using the same programming language as in the backend.

The thesis takes a look at the origin, definition and requirements of the REST architecture. As an outcome of the study a software was made mostly fulfilling the requirements set.

Keywords	REST, REST API, Vaadin framework, JSON, Java, mock, Jackson

Sisällys

Lyhenteet

REST	REpresentational State Transfer. Roy T. Fielding:in luoma arkkitehtuurimalli.
API	Application Programming Interface. Rajapinta.
POC	Proof-of-concept. Yksinkertainen toteutus, joka näyttää, miten jokin toimii tai että se edes voidaan tehdä
JSON	JavaScript Object Notation. Alun perin JavaScriptin tapa esittää olio tekstinä. Nykyään itsenäinen dataformaatti
XML	eXtensible Markup Language. Määrittelykieli, jossa tiedon selitys on itse tiedon seassa. Käytetään tiedonsiirtoformaattina järjestelmien välillä.
SOAP	Simple Object Access Protocol. Microsoftin luoma integraatiostandardi, jonka viestiformaatti perustuu XML: ään.
JIT	Just in Time. Koodin käännöstapa, jossa koodi käännetään käyttövalmiiksi vasta, kun sitä tarvitaan ensimmäistä kertaa.

1 Johdanto

Erilaiset ketterän kehityksen tyylilajit ovat suuressa osassa nykypäivän ohjelmistokehitystä. Keskeisessä osassa ketterän kehityksen ideologiaa on jakaa kehitystyö useaan pienempään vaiheeseen. Näistä käytetään usein nimitystä sprintti. Jakamalla prosessi useaan osaan tehdystä työstä saadaan nopeammin palautetta ja kehityssuuntaa voidaan ohjata helpommin. Tärkeää tässä nopeassa palautesyklissä on esitellä kaikille sidosryhmille, mitä vaiheen aikana on saatu aikaiseksi, jotta sidosryhmillä on tarvittavat tiedot työn priorisointiin ja mahdollisiin suunnan muutoksiin.

Kun ohjelmaan kuuluu järjestelmäpino käyttöliittymään asti, esittely on yksinkertaisinta toteuttaa käyttöliittymällä itsellään. Aina koko sovellus ei ole kuitenkaan yhden tiimin tai edes organisaation kehittämä, mikä voi johtaa päänvaivaan esittelyssä. Tällöin käyttöliittymän toteuttava taho voi käyttää erilaisia mock-rajapintoja. Yleisesti mock on jonkinlainen korvike oikealle toiminnolle, jota voidaan käyttää muun muassa helpottamaan testaamista, kun oikeaa asiaa ei voida käyttää (1). Yksinkertaisimmillaan mock-rajapinta sisältää kaikki rajapinnan kutsut, mutta palauttaa niistä aina saman kovakoodatun vastauksen. Mockin tarkoituksesta riippuen se saattaa jopa tallentaa käyttäjän syötteet talteen ja käyttää sitä myöhemmin, mutta vain uudelleenkäynnistykseen asti. Jotkut käyttävät eri nimiä eri laajuisista mock-rajapinnoista (2), mutta tämän työn kannalta ne ovat kaikki mock-rajapintoja. Rajapinnan toteuttavalle taholle ei ole mitään näin valmista ratkaisua käyttöliittymän osalta.

REST-rajapinnat on tarkoitettu siirtämään koneluettavaa materiaalia ohjelmien välillä. Vastaanottava pääteohjelma käsittelee tiedon, eli yleensä muokkaa saadun tiedon esitettävään muotoon tai antaa sen eteenpäin toiselle ohjelman osalle. Vaikka XML/JSON-muotoisesta rakenteesta pystyy ihminen lukemaan sisällön, se vaatii joko perehtyneisyyttä tietorakenteeseen tai paljon vaivaa. Ohjelmiston kehitysvaiheessa tarvittava tieto löytyy vain sovelluksen kehittäjiltä. Myöhemmin integraatioita tekevät henkilöt saavat lähes yhtä hyvän pohjan. Bisnespuolen henkilöt eivät normaalisti tule taustajärjestelmiä niin hyvin oppimaan, eikä heidän tulisi joutuakaan.

Siirtoformaatin käyttäminen esittelytyökaluna ei tue minkäänlaista kokonaiskuvan muodostamista, jolloin vähemmän teknisten sidosryhmien tilannetaju kärsii. Huono luetta-

vuus ja kokonaiskuvan puute johtavat siihen, että raaka XML/JSON-esittely ei ole mielekästä millekään osapuolelle. Tästä ajatusketjusta muodostui insinöörityön tavoite, eli tehdä ohjelma, jolla pystytään esittelemään ei-tekniselle henkilölle REST-tyylisen rajapinnan senhetkistä tilaa ja sitä kautta rajapintaan tehtyjä muutoksia. Käytännössä on kyse ,ock-rajapinnan käyttöliittymäversio.

2 Backend-kehityksen demoaminen sidosryhmille

Kokemukseni on, että REST-rajapinnan kohdalla sidosryhmille esittely tapahtuu kirjoittamalla tekstitiedostoon valmiiksi sarja komentoja, joita lähetetään rajapinnalle esittelyssä ja näytetään paluuviestinä tullutta siirtoformaattia. Tähän on enemmän ja vähemmän automatisoituja työkaluja, mutta loppujen lopuksi ne operoivat raa'alla siirtoformaatilla.

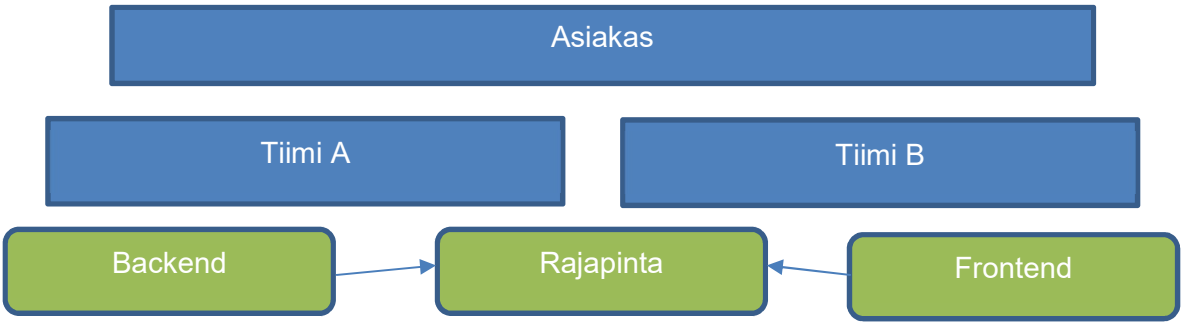
```
{
   userId: "1234",
   name: "custmer1",
   email: "example@example.com",
   country: "fi",
   currency: "eur",
   language: "fi"
}
```

Koodi 1. Yksinkertainen esimerkki JSON-siirtoformaatista.

Kun asiakkaalle näyttää esittelytilaisuudessa edellä olevan esimerkkikoodin 1 JSON:in ja kertoo, että siihen on nyt lisätty kenttä "currency", on demo hyvinkin mitäänsanomaton. Tällaisien esitysten seuraajille ei myöskään muodostu mitään kokonaiskuvaa rajapinnan tilasta, kun huomio kiinnittyy aina yksittäisten rajapintakomentojen suorittamiseen. Yksinkertainenkin käyttöliittymä toisi ei-tekniselle henkilölle muutoksen paljon luontevammin esiin ja helpottaisi kokonaiskuvan muodostamista. Aikaisempi esimerkki voitaisiin esitellä näyttämällä, että mock-käyttöliittymässä pystyy nyt näkemään käytetyn valuutan, ja jos rajapinta mahdollistaa sen, vaihtamaan valuutan.

Konkreettisempi esitystapa ja parempi kokonaiskuva auttavat ihmisiä näkemään puuttuvat tiedot rajapinnan rakenteessa helpommin, kuin sarja rajapintavastauksia raakana tiedonsiirtoformaattina kykenee. Kehittäjälle esimerkin userId olisi luultavasti se kenttä, jolla siirrytään rajapinnan rakenteessa syvemmälle, ja täten on se kenttä, mihin ohjelmoijan huomio kiinnittyy. Joku toinen voisi esimerkiksi huomata, että syntymäpäivä puuttuu

viestistä, koska se kuuluu asioihin, joita hän tarvitsee, tai liittyy johonkin toiminnallisuuteen, joka hänellä on mielessä.



Kuva 1. Yhden organisaation hallussa olevan rajapintapohjaisen sovelluksen toteutuksen vastuunjako. Tiimi A toteuttaa taustajärjestelmän, B käyttöliittymän. Tiimit tekevät yhteistyötä rajapinnan suunnittelussa.

Tärkeää on myös muistaa, että ohjelmiston toiminnallisuus on erikseen jaettu taustajärjestelmään ja erilliseen käyttöliittymään (kuva 1) joko bisnesstrategian tai arkkitehtuurillisen syyn vuoksi. Tällaisessa tilanteessa REST-rajapinnan tekijän ei ole mielekästä käyttää suurta määrää aikaa luodakseen esittelykäyttöliittymiä, oman testaustarpeensa ulkopuolella. Kuitenkin parempi esitystapa helpottaisi esimerkiksi rajapinnan puutteiden huomaamista ajoissa, paljolti kokonaiskuvan muodostumisen kautta. Kun toiminnallisuus on sidottu johonkin näkyvään, osaavat sidosryhmät helpommin hahmottaa, mitä tietoa rajapinta vaatii ja mitä se palauttaa. POC-käyttöliittymän myyminen asiakkaalle voi olla vaikeaa, jos lopullisen käyttöliittymän on tarkoitus valmistaa joku toinen organisaatio.

Tällaisen esittelykäyttöliittymän valmistelu ei kuitenkaan saa viedä paljoa kehittäjän aikaa, joten käyttöliittymän luovan järjestelmän on sisällettävä hyviä pohjia esitystä varten. Käyttöliittymän osalta painopiste on oltava helpolla muokkaamisella. Jossain määrin rujo ulkoasu on oletettavaa ja jopa melkein toivottavaa, jotta asiakas muistaa, että kyseessä on vain esittelytyökalu eikä lopullinen käyttöliittymä.

Työssä tehdyn sovelluksen tarkoitus ei ole avustaa kehittäjää oman testaamisensa kanssa, kehitysvaiheen kokeiluissa tai automaatiotestauksen toteutuksessa. Vaikka ohjelma saattaa auttaa puutteiden löydössä, se ei ole testaamisen työkalu.

3 API

API, interface tai rajapinta viittaa verkkopalveluiden yhteydessä ohjeistukseen, joka kertoo, miten jotain tiettyä palvelua kutsutaan niin, että saadaan halutun tyyppinen odotettu vastaus takaisin tai tapahtumaketju alkamaan palvelimen päässä. Puhuttaessa saatetaan viitata myös palvelua tarjoavan pään toteutukseen.

Käsitteenä API on laaja sisältäen monia alatarkoituksia. Monet näistä ovat keskeisiä osia ohjelmistojen kehittämisessä, joten lähes kaikki ohjelmointia tekevät tunnistavat osan niistä, ja vähintään käyttävät niitä, vaikkeivat olisikaan termistä tietoisia. Työssä tehdyn ohjelman kannalta tärkeimmät rajapinnat ovat ne, jotka siirtävät tietoa verkon yli, mutta muitakin selitetään lyhyesti.

3.1 Lyhyt historia

Yhden tietokoneen sisällä erilaiset rajapinnat ovat olleet olemassa siitä lähtien, kuin käyttöjärjestelmät ovat olleet olemassa. Ne antavat ensimmäistä kertaa mahdollisuuden sille, että jokaisen sovellutuksen ei tarvinnut enää sisältää koodia koko laitteiston hallintaan. Erilaiset kirjastot ovat olleet seuraava askel kehityksessä. Näissä ilmenee rajapintojen yksi tärkeimmistä käytöistä, sillä olemassa olevat kirjastot ratkaisevat monia ongelmia, joihin törmätään usein ohjelmistokehityksessä.

Julkiset verkkorajapinnat ovat melko uusi asia. Ne ovat saaneet alkunsa mm. eBay:n rajapinnoista, Googlen kartta API:sta, Twitterin API:sta ja Amazonin rajapinnoista (3).

3.2 Erilaiset APIt

Normaalissa ohjelmistokehityksessä erilaiset kirjastot ovat yleisimmin kohdattu rajapintatyyppi. Kirjasto toteuttaa jonkin tietyn toiminallisuuden osittain tai kokonaan, mikä helpottaa ohjelmiston kehittämistä. Kirjasto ei ole itsessään suoritettava ohjelma eikä niiden yleensä ole tarkoitus keskustella sovelluksen ulkopuolelle, mutta ne voivat helpottaa prosessia. Verkko-API:n käyttöön voi olla kirjastototeutus.

Käyttöjärjestelmät tarjoavat rajapintoja systeemikutsujen muodossa. Korkeamman abstraktiotason ohjelmoinnissa ohjelmoija ei joudu ajattelemaan näitä yleensä itse. Tästä huolimatta järjestelmäkutsut ovat kriittisiä nykypäiväisessä ohjelmistopinossa.

Verkon yli toimivat rajapinnat jakautuvat kolmeen eri ryhmään: privaatti -, partneri - ja julkinen verkkorajapinta (4). Privaatti verkko-API on rajapinta, joka tarjoaa entiteetin (firma, valtio tai yms.) sisällä toiminnallisuuksia mahdollistaen henkilöiden ja järjestelmien hyötyä valmiista toiminnallisuudesta ja varmistaa, että tietoa ei tarvitsisi tallentaa useaan paikkaan ja että samaa toiminnallisuutta ei tarvitse toteuttaa useaan kertaan. Rajapinnan toteuttaa oma erillinen sovellus. Käyttäville sovelluksille voidaan antaa melko laajoja oikeuksia ja voidaan luottaa sovelluksien varmistavan itse, että niiden käyttäjällä on oikeus tehdä asia, ennen kuin rajapintaa kutsutaan.

Täysin julkinen API tarjoaa palvelujansa vapaasti internetissä. Jos osa tai kaikki informaatiosta on luottamuksellista, tulee järjestelmän kyetä itse varmentamaan, että käyttävällä taholla on oikeus saada tieto ja että käyttäjä se, kuka väittää olevansa. Se sisältää sekä vapaasti saatavia palveluja kuten virastojen järjestelmiä että maksullisia palveluita, joita yritykset myyvät eteenpäin. Partnerirajapinta on näiden välimaastossa. Rajapintaan pääsee entiteetin ulkopuolisia toimijoita, mutta vain tarkasti valitut toimijat. Tarkoituksena on helpottaa tai mahdollistaa yhteistyötä entiteettien välillä.

Tämän työn kannalta privaatti verkkorajapinta on tärkein rajapintatyyppi, vaikkakin konsepteja pystyy melko pienellä vaivalla siirtämään sekä partneri- että julkiseen rajapintaan.

4 REST

4.1 Historia

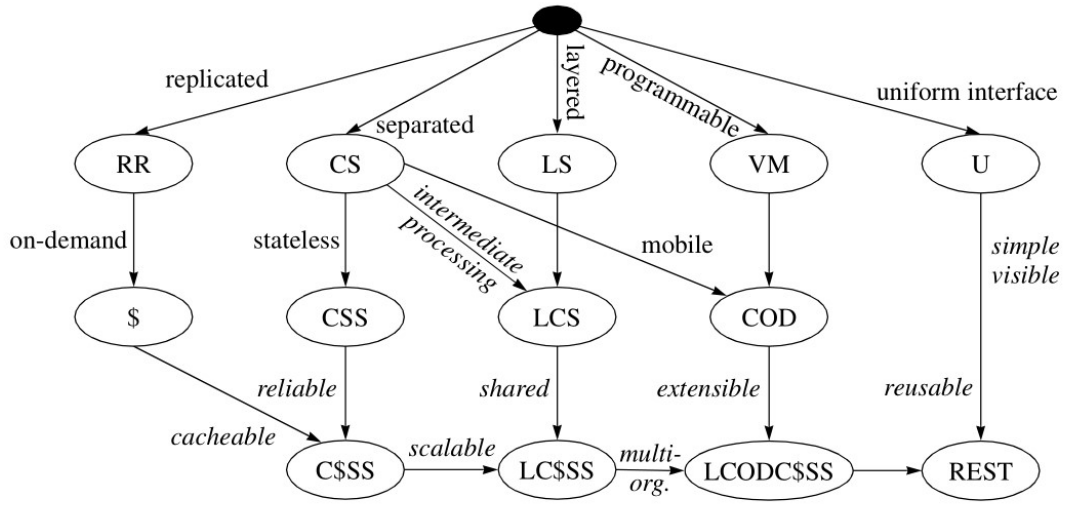
REST on terminä Roy Fieldingsin luoma. Hän loi termin alkuperäisen määritelmän käsitellessään varhaisen WWW-arkkitehtuurin senhetkistä tilaa, ja suunnitellessaan parempaa mallia, jonka päälle internet kykenisi kasvamaan hyvin. Suunnittelun lopputuloksia on käytetty HTTP 1.1 -standardin suunnittelussa. Fieldings käsittelee suunnitteluproses-

sia väitöskirjassaan (5). Nykyinen internet ei joltain osin enää vastaa Fieldingsin alkuperäistä määritelmää, samoin kuin REST-termiä käytettäessä tarkoitettavat rakenteet eivät enää vastaa määritelmää.

Arkkitehtuuri suunniteltiin yksinkertaiseksi ja helpoksi toteuttaa, koska WWW oli pääasiassa vapaaehtoisprojekti (6). Pohjimmiltaan oli ajatus auttaa tekemään järjestelmä, joka on niin helppo ymmärtää ja toteuttaa, että kaikki web-kehityksen haarat alkaisivat käyttää samaa standardia oma-aloitteisesti.

4.2 Ydinkonseptit

Arkkitehtuurin suunnittelun Fielding aloitti niin kutsutusta tyhjästä tyylistä, eli kaiken sallivasta arkkitehtuurista. Tätä sitten rajoitettiin vaatimus kerrallaan.



Kuva 2. Fieldingsin tohtorin väitöskirjan kuva 5-9. Kuvaa REST-arkkitehtuurin kasaamisprosessia

Suunnittelunsa lopuksi Fielding oli valinnut arkkitehtuurimallit, joita kutsui nimillä client-server, stateless, cache, uniform interface, layered system ja code-on-demand esittämään hänen näkemystään HTTP:n tavoitetilasta (kuva 2).

Ensimmäinen määrite on erottaa palvelin ja käyttävä järjestelmä, jotta kumpikin voidaan tehdä yksinkertaisemmaksi ja niitä voidaan kehittää erikseen toisistaan riippumatta.

Kommunikaatio jaetaan useaksi tilattomaksi osaksi, joissa kaikissa lähetetään kaikki tieto, mitä palvelin tarvitsee kyselyn käsittelyyn, jolloin palvelimen ei tarvitse ylläpitää muistissa erillisiä merkintöjä käyttäjien viimeksi tekemistä asioista. Välimuistin käytöllä tuetaan arkkitehtuurin skaalautuvuutta, kun loppukäyttäjä ja/tai välikomponentit saavat säilöä vastauksia ja käyttää uudelleen niitä, jos uskovat, että ne eivät ole muuttuneet vielä.

Yhdenmukainen rajapinta kaikkien komponenttien välillä pohjustaa kerroksittaisen järjestelmän, joka piilottaa mahdolliset välikomponentit sekä palvelimelta että käyttäjältä. Tämä mahdollistaa kuormantasausta, proxy-palvelimia ja prosessin jakamisen moneen eri toteutukseen jne.

Kokonaisuuden viimeistelee Code-on-Demand, jossa ”client”-pään toiminnallisuutta laajennetaan ilman, että sen koodiin tarvitsee koskea. HTML-tekniikan kanssa muun muassa JavaScript suorittaa tämän tarpeen. Tämä vaatimus on Fieldingsin listaamista ainut valinnainen, ja jää myös työssä vähälle käsittelylle.

4.3 REST ei ole API

Oman kokemukseni perusteella REST-termiä käytetään pääasiassa viittaamaan REST-rajapintoihin, vaikka termi ei rajapintaan oikeastaan viittaa. REST on arkkitehtuurimalli, joka suunniteltiin kuvaamaan HTTP-maailman haluttua tilaa, jotta internet skaalautuisi mahdollisimman hyvin (7.). Käsitteenä REST on ajan kanssa korruptoitunut ja muuttunut, ja osaan alkuperäisistä konsepteista ei tarvitse kiinnittää suuremmin huomiota rajapinnasta puhuttaessa. Tämä johtuu siitä, että REST API -ratkaisut ovat lähestulkoon aina HTTP-tekniikkapinon päällä, eli alkuperäisen REST-ratkaisun nykyinen versio hoitaa ne.

4.4 REST API

REST-arkkitehtuuria hyödyntävästä rajapinnasta käytetään usein myös nimitystä RESTful API. Rajapintatyypissä tärkeässä osassa on, että kutsut on jaettu useaan eri resurssiin, joista jokaisesta käsitellään vain tiettyä asiaa. Suurin kilpaileva malli SOAP tarjoaa vain yhden käsittelypisteen, josta tehdään kaikki rajapintaan liittyvät asiat. Yksi erinomai-

nen työkalu ymmärtää, mitä REST-rajapinta tarkalleen tarkoittaa on Richardsonin kypsyysmalli (8), joka mittaa, kuinka lähellä rajapinta on REST API:a. Malli määrittelee neljä tasoa: 0-3.

- 0: Swamp of POX
- 1: Resurssit
- 2: HTTP-verbit
- 3: Hypermediakontrolli

0-taso on HTTP:n päälle tehty API, joka ei hyödynnä HTTP:n mekanismeja millään muulla tapaa kuin datansiirtotyökaluna, mm. SOAP lasketaan tähän luokkaan. Resursseja käyttäessä on API jaettu erillisiin kontaktipisteisiin, jotka kukin käsittelevät tiettyä asiaa. Resurssit on yleensä jaettu URI:lla hierarkkisesti alkaen tietystä juuriosoitteesta. Taso 1 helpottaa rajapinnan hallittavuutta ja laajennettavuutta. Rischardsonin mallin toisella tasolla rajapinta hyödyntää HTTP:n tarjoamia eri kutsu”verbejä” POST, GET, PUT, DELETE jne. määrittelemään, mitä ollaan tekemässä. Tämä mahdollistaa samalla, että samaa URI:a voi käyttää usea resurssi, käsittelemään samaa loogista rakennetta eri tavalla esimerkiksi komennot hae, luo, päivitä ja poista. Viimeisellä tasolla rajapinnan vastaukset sisältävät hypermediassaan linkkejä seuraaviin rajapintoihin, jos sellaisia on. Esimerkiksi asiakkaan haku voisi sisältää linkin sen muokkaamiseen, poistoon, asiakkaan ostoskoreihin ja tilaushistoriaan. Vasta tasolla 3 rajapinta on oikeasti REST-rajapinta, mutta mallin seuraaminen tasoon kaksi asti tuntuu oman kokemuksen mukaan riittävän siihen, että työelämässä aletaan puhua REST-rajapinnasta. Esimerkiksi työn testaamisessa käytetyssä rajapinnassa ei näitä linkkejä ole. Tämä ero määreiden välillä on yksi esimerkkinä siitä, miksi mielestäni termin tarkoitus on korruptoitunut käytössä.

Näiden lisäksi Roy T. Fieldingsillä on vielä oma lista asioista, joita rajapinta ei saa tehdä, jos tekijät haluavat kutsua rajapintaansa REST-rajapinnaksi (9). Hänen listansa on melko korkealla abstraktiotasolla. Listalla on muun muassa protokollageneerisyys (REST API:n ei tarvitse olla HTTP:n päällä, kunhan uusi protokolla tarjoaa tarvittavat työkalut metadatan siirtoon). Rajapinta ei saa muokata protokollaa, jolla se toimii ja että data on tyypitöntä (voidaan esittää ihmiselle sellaisenaan). Huomattava ero on se, että

Richardsonin malli viittaa suoraan http-protokollaan, kun taas Fielding huomauttaa erikseen, että terminä REST ei oikeastaan määrittele protokollaa, vaikka historiallinen yhteys HTTP:n kanssa on olemassa.

5 API integraatiotyökaluna

Rajapinnat ovat yrityksille tärkeä työkalu saada olemassa olevat järjestelmät tarjoamaan toiminnallisuuttaan sekä uusille sovelluksille että olemassa oleville järjestelmille, joihin lisätään jotain uutta. Moni yllättäväkin järjestelmä tulee jossain elämänvaiheessaan tarjoamaan toiminnallisuuksiaan eteenpäin organisaation muille järjestelmille. Sovellukset voivat olla eri tekijöiden, eri aikaan tehtyjä ja eri tahojen hallinnoimia, mutta kaikkia tarvitaan bisnestapahtuman suorittamiseen. Tämä johtuu paljolti siitä, että tietojen tallentaminen moneen eri paikkaan, validointi ja käsittely nostaa kuluja ja johtaa ylläpito-ongelmiin. Tämän takia suurien organisaatioiden suuntaus on kehittää järjestelmiä, jotka toteuttavat kerrosarkkitehtuuria. Kun uusia järjestelmiä/palveluja halutaan lisätä, ne halutaan osaksi olemassa olevaa arkkitehtuuria. SSOT (singe source of truth) on esimerkki kattotermi ratkaisuille, jotka pyrkivät tukemaan tällaista ajattelua (10).

Kytkemällä organisaation eri järjestelmät kiinni keskenään voidaan yhden järjestelmän toiminnallisuutta hyödyntää toisaalla, eikä samaa toiminnallisuutta tarvitse toteuttaa moneen kertaan. Tämä tarkoittaa, että olemassa oleviin järjestelmiin joudutaan kehittämään komponentteja, jotka tarjoavat niiden ydintoiminnallisuutta ulospäin. Itse komponenttien kehitykseen liittyvät ongelmat eivät ole tämän työn piirissä, mutta niiden kehityksen etenemisen esittäminen asiakkaalle on päätavoite.

6 Ratkaisun suunnittelu

Tarvitaan siis työkalu, jolla saa käytettyä erilaisia REST-rajapintoja minimivaivalla, tiedon esitettyä ymmärrettävämmässä muodossa. Tällöin saataisiin POC-käyttöliittymä rajapinnalle kasattua nopeasti ja vaivattomasti.

Konkreettisemmin, tavoitteet ovat:

- Konkreettinen graafinen käyttöliittymä

- Helposti muokattavissa uuteen rajapintaan

- Kykenee hyödyntämään aikaisempien kutsujen dataa jatkokutsuissa konfiguroidusti.

- Jatkokehittämiseen mahdollisimman pieni kynnys

Jos rajapinnat olisi toteutettu täysin Fieldingsin määritelmän mukaisesti, voisi yksi geneerinen ratkaisu hoitaa esittämisen kaikkialla pelkällä näytettävien kenttien konfiguroinnilla. Oman kokemukseni pohjalta suuri osa yrityksillä kentällä käytössä olevista rajapinnoista eivät toteuta osia alkuperäisestä määrittelystä, jolloin täysin geneerinen toteutus on huomattavasti vaikeampaa. Täten tavoitteeksi muodostui tehdä järjestelmä, jonka saisi kiinni uuteen rajapintaan minimivaivalla.

6.1 Olemassa olevia työkaluja

Rajapinnat ovat isossa asemassa yritysmaailmassa, joten ongelman parissa ovat olleet muutkin. Osa syntyneistä järjestelmistä täyttää ainakin osan tavoitteista, mutta en ole vielä löytänyt yhtään, joka ratkaisisi kaikki.

Swagger on REST-rajapintojen kuvaamiseen ja suunnitteluun tarkoitettu työkalu. Swaggerin kehittäjät tarjoavat myös ohjelman, joka kytketään ajossa olevaan ohjelmaan, jossa on API, joka on toteutettu Swagger-dokumentaation mukaan. Rajapinnasta haettu tieto tarjoillaan sellaisena kuin se saatiin, eli yleensä joko XML tai JSON. Täten työkalu ei täytä vaatimuksia.

Swagger-dokumentaation pohjalta UI:n rakentaminen on mahdollinen jatkokehitysprojekti.

Kun etsitään suosituksia, miten testata tai esitellä rajapinnan toimintaa, lähes kaikki löytyvät keskustelut aiheesta painottuvat suosittelemaan erilaisia selainliitännäisiä, joilla voi suorittaa HTTP-pyyntöjä. Käytännössä ne ovat selainpohjaisia versioita Linuxin curl-ohjelmasta.

Koska selainliitännäisten esitystapa rajapinnan vastaukselle on siirtoformaatti raakana, vastaavat ne loppujen lopuksi tekstieditorissa olevaa listaa komennoista, joita voi sitten kopioida ja lähettää parametrien muuton jälkeen rajapinnalle curlia käyttäen. Eli selainliitännäiset ovat rinnakkaisratkaisu nykyiselle curlin käytölle.

6.2 Käyttöliittymän toteutusvaihtoehtoja

Käyttöliittymän helppo muokkaaminen uuden rajapinnan kanssa on kriittinen vaatimus ohjelmalle. Käyttöliittymän valmistustavaksi arvioitiin muutamia vaihtoehtoja. Tavoitteena oli etsiä kirjasto tai framework, joka on helppo oppia ja jolle voidaan tehdä muokattavia valmispohjia.

Jquery olisi verkkokauppa-alustoista tuttu työkalu, mutta vaatisi melko paljon työtä HTML:n ja JavaScriptin kanssa tarjota peruskäyttöliittymä, varsinkin jos halutaan enemmän kuin tekstiä ja linkkejä valkoisella taustalla. Olisi iso työ tehdä mitään UI-työtä, mutta toisaalta se soisi vapaat kädet toteuttaa, mitä haluaa. Vaihtoehto tarjoaisi vapaat kädet pääasiassa tosin sen takia että mitään valmista arkkitehtuuria ei ole.

AngularJS, Jqueryn päälle rakennettu framework tarjoaisi paremmat työkalut omatekoisen sivun JavaScript-puoleen kuin pelkkä Jquery. Sivuston rakenne jouduttaisiin yhä tekemään kokonaan itse.

Bootstrap tukisi aikaisempia vaihtoehtoja tarjoamalla valmiita sivurakenteita käytettäväksi. Tämä vaatisi Bootstrapin työkaluketjun käyttöönoton, joka olisi jonkin verran kynnys backend-devaajille lähteä rakentamaan toteutetun ohjelman päälle, vaikka loppujen lopuksi olisi nopeampaa kuin edelliset ilman Bootstrapin tukea.

ZKOSS on Java+XML-pohjainen järjestelmä, joka printtaa ajon aikana sivun HTML- ja JavaScript-osat käyttäjälle lähetettäväksi. ZKOSS on Potix-nimisen yrityksen tuote. Kehys mahdollistaisi sen, että rajapinnan kehittäjät voisivat työstää sivustoa pääasiassa samalla kielellä kuin ohjelmakoodia. Ohjelmiston lisenssi on maksullinen enterprice-käyttäjille.

ZKOSS on käytössä verkkokauppa-alustassa, jota käytän töissä, jota kautta minulla on ollut huonoja kokemuksia. Alustasta ei päästä suoraan ZKOSS:iin käsiksi, eli suoraan käytettynä se saattaa olla vähemmän huono.

GWT on Googlen aloittama avoimen lähdekoodin projekti, joka on Java-pohjainen järjestelmä, joka printtaa ajon aikana sivun HTML- ja JavaScript-osat samaan tyyliin kuin ZKOSS. GWT käsittelee näkymää korkeammalla abstraktiolla kuin ZKOSS, joten se olisi mukavampi käyttää.

Vaadin on ohjelmistokehys, joka hyödyntää GWT:tä, tarkoituksena nopeuttaa kehityssykliä edelleen. Täten Vaadinin pitäisi tarjota kaikki GWT:n edut omiensa lisäksi. Kehyksessä on käytetty paljon saman tyyppistä rakennetta kuin Java Swing -kirjastossa. Lähes kaikki Java-kehittäjät ovat jossain määrin tutustuneet kirjastoon. Toki tekijät ovat pyrkineet vähentämään käytön vaatimaa koodirivien määrää verrattuna Swing-kirjaston käyttöön.

Ilves bootstrap tarjoaa valmiin sivupohjan, käyttäjien hallintaa yms. Tämä luo vahvaa pohjaa siihen, että järjestelmää voidaan käyttää laajemminkin tarvittaessa. Puhtaasti Vaadinin päällä sivuarkkitehtuuri jouduttaisiin tekemään itse. Bootstrapin mukana tulevat näkymät toimivat myös samalla hyvinä esimerkkeinä uusien näkymien tekemiseen.

Ilves paketoi mukaan pienen palvelimen, joten uuden ympäristön voi pystyttää suoraan versionhallinnasta koodin hakemalla.

6.3 Valintana Vaadin + Ilves

Rajapintoja toteuttaa backend-ohjelmoijat, joten on luonnollisinta, että he myös käyttäisivät tätä työkalua. Tällöin mahdollisuus ohjelmoida käyttöliittymä samalla kielellä, kuin pääasiallinen taustasovellus on hyvä. Tämä mahdollistaa myös sen, että kaikki käyttävät henkilöt voivat kehittää järjestelmää edelleen.

Eli lopullinen UI pino on GWT > Vaadin > Ilves.

7 Ratkaisu

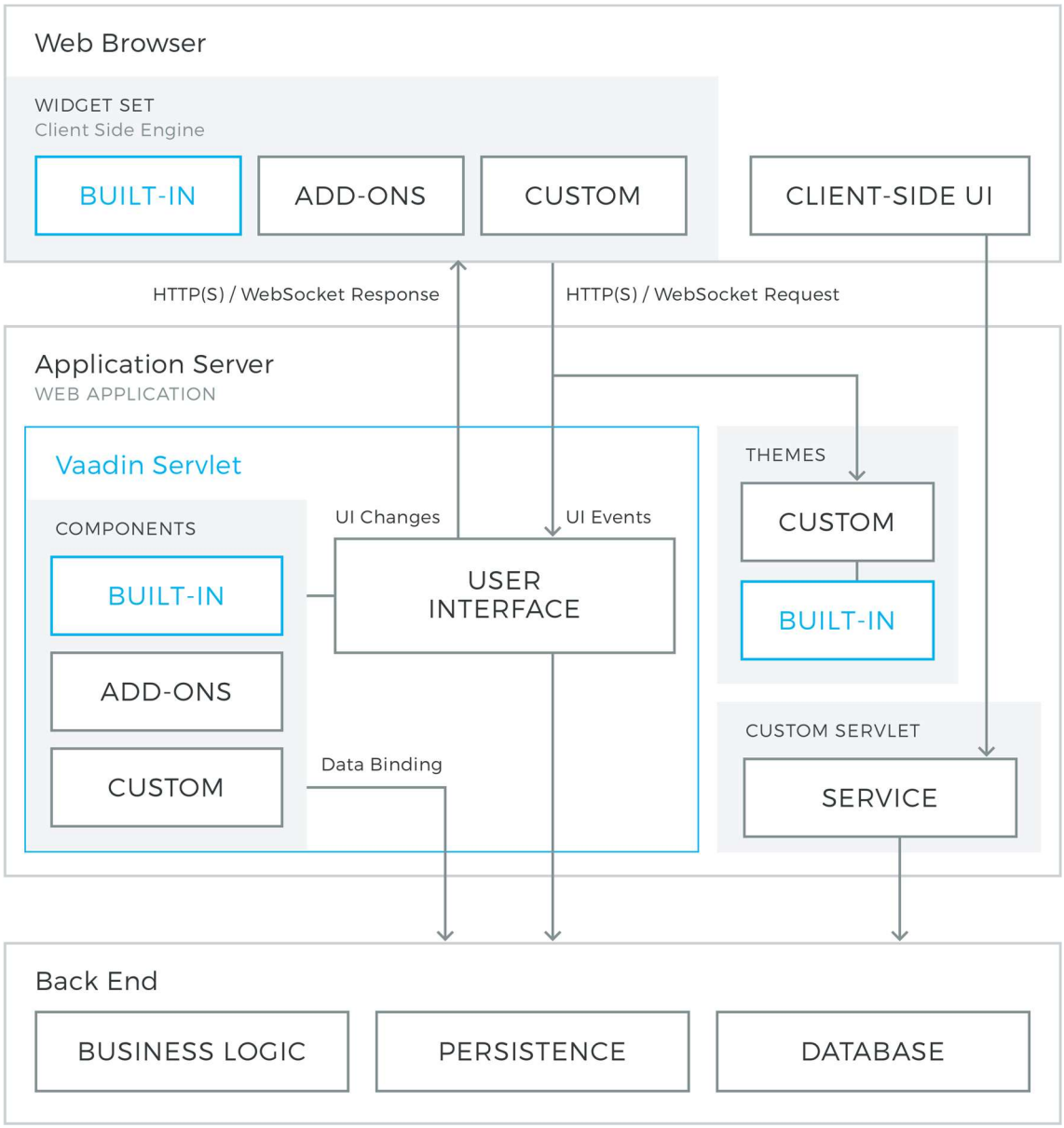
7.1 Teknologiapino kokonaisuudessaan

Ohjelman pääteknologioiksi kasautui Java 8, jetty, Vaadin -> GWT, Jackson fasterXml, ja Apache httpClient. Ilves-paketin mukana tulee myös useita muita kirjastoja, mutta suurin osa niistä ei ole sovelluksen toiminnassa kriittisessä osassa. Yllä listatut on tärkeimmät osat. Listaamattomat kirjastot ja järjestelmät luovat pohjan jatkokehitykselle tarjoamalla mm. tietokannan samassa paketissa ja hoitamaan käyttäjähallintaa, jos sovellus jätetään jollekin palvelimelle ajoon pidemmäksi aikaa. Pääasiallisen tavoitteen kannalta niistä ei ole suoraa etua.

Jetty on Eclipse-säätiön tukema avoimen lähdekoodin palvelintoteutus. Ilves bootstrap paketoi palvelimen mukaansa, palvelin on upotettu Ilveksen koodin, joten sovelluksen käynnistäminen nostaa palvelimen samalla. Tämä mahdollistaa sen, että täyden ympäristön saa pystytettyä pelkästään hakemalla koodin versionhallinnasta.

Nimestään huolimatta fasterXml on ollut alun perin JSON:in käsittelykirjasto. Se käsittelee JSON:ia puurakenteena ja mahdollistaa melko helposti geneerisen oliokäsittelyn. Se pystyy käsittelemään isompia JSON -kokonaisuuksia kuin Oracle JDK:n sisään rakennettu toteutus. Kirjasto pystyisi myös mäppäämään JSON:n rakenteen olioksi. Työn tavoitteelle se ei ole täysin hyödyllistä, mutta staattisissa tyypitetyissä olioissa sitä olisi voinut käyttää. FasterXml:n geneerinen JSON-mäppäys tehdään kirjaston omilla olioilla, jotka ovat käytännöllisempi geneerisessä käsittelyssä kuin Googlen gson-kirjasto.

GWT (Google web toolkit) on Googlen perustama ja ohjaama avoimen lähdekoodin UI:n luontityökalu. Se mahdollistaa html+javascript -pohjaisen käyttöliittymän toteuttamisen Java-koodilla. Riippuen konfiguroinnin tilasta GWT joko kääntää käyttöliittymän koodista JIT-tyyliin muistiin tai tekee valmiiksi kaikki luokat erillisiksi tiedostoiksi, jotka voidaan siirtää tuotantopalvelimelle ilman GWT:n Java-osuutta.



Kuva 3. Vaadin-arkkitehtuuri kuva https://vaadin.com/docs/-/part/framework/architecture/architecture-overview.html

Vaadin on samannimisen yrityksen tekemä ja ylläpitämä, ydinosiltaan vapaan lähdekoodin sovelluskehys, joka rakentaa osittain GWT:n päälle. Kehys tarjoaa työkalut koko UI-työskentelyyn Java-koodilla käyttäen javax.swing-tyylistä rajapintaa, mutta huomattavasti vähemmällä rivimäärällä antaen mahdollisuuden tehdä koko sovellus pelkästään Javalla. Järjestelmä ei käytä GWT:n Java-koodia, vaan sen tarjoamien komponenttien html/JavaScript-käännös on tehty valmiiksi GWT:llä. GWT:n Java-koodi tarvitaan mukaan vain, jos halutaan lisätä uusia komponentteja järjestelmään, ja silloinkin on mahdollista matkia valmista ratkaisua (11.) GWT komponentit ovat kuvan 3 widget set/client side enginen sisällä (12).

Töissäni olen tekemisissä projektin kanssa, joka käyttää JSON:ia REST-rajapinnassaan, joten pääasiallinen käsitelty formaatti on JSON. Järjestelmä on suunniteltu niin, että XML-tuki voidaan lisätä jälkikäteen arkkitehtuuria muuttamatta.

7.2 Kommunikaatiokerros ja datamalli

Sovellus toteutettiin muutamassa vaiheessa. Näistä tärkein ero on staattisen datamallin luomisvaiheen ja myöhemmän dynaamisemman ratkaisun välillä. Muuten kehitys on ollut lineaarista.

Toistuvaa päänvaivaa kommunikaation kanssa on aiheuttanut kehitysympäristön Javan päivittämisen yhteydessä kauppa-alustan lokaaliympäristöissä tarjoama oletus SSL sertifikaatti, joka on vanhentunut, itseallekirjoitettu ja käyttää hash-funktiota, jota Java 7 tai 8 eivät oletuskonfiguraatiollaan hyväksy. Tämän lisäksi Ilves tekee oman sertifikaattinsa niin turvalliseksi, että Java-asennukseen pitää päivittää rajoittamaton kryptografialisenssi, jotta sertifikaatti saadaan generoitua.

REST on sen verran yksinkertainen käyttää, että alustavasti käytin Apachen kirjastojen httpClient-kirjastoa käsittelemään kommunikaatiota. Suurimmin osin kommunikaatio voidaan jakaa palvelimen pysyvään osoitteeseen ja kutsukohtaiseen päätteeseen. On olemassa erilaisia REST client -kirjastoja, mutta niiden tuoma etu arvioitiin niin pieneksi, ettei sellaista otettu käyttöön. Rajapintapohjaisena ratkaisuna nykyisen toteutuksen rinnalle voi tuoda jonkin kirjaston tulevaisuudessa.

Ilves:llä on geneerinen malli esitettävälle datalle, joten työn oma datamalli sidottiin siihen. Tämä varmistaa, että bootstrapin valmiit luokat osaavat käsitellä dataa, jolloin ne toimivat hyvinä esimerkkeinä, miten näkymiä tehdään. Rakenne tukee geneeristä tietuetta ja luokkien extension kautta tehtävää spesifistä rakennetta. Geneeriseessä tapauksessa joudutaan näytettävät kentät määrittelemään esityskerroksessa, spesifissä ratkaisussa item-luokan lapsi määrittelee itse, mitä kenttiä se sisältää. Dataolion luominen koodikantaan vaatii, että keskitetylle tehtaalle on kerrottu, että on olemassa uusi malli ja luoda mallille tapa tunnistaa, onko joku tietty JSON-puu sen kuvaamaa dataa.

```
  public static void init() {
    if (INIT_DONE) {
      return;
    }
    RestItemFactory.registerClass(CustomerItem.class);

    // TODO needs a nicer, likely cofigured, way to do this
    final FieldSetDescriptor customerItemFields = new FieldSetDescriptor(Customer-
  Item.class);
    customerItemFields.setVisibleFieldIds(new String[] { COMPANY_NAME, ERP_ID, LANGUAGE,
  EMAIL });
    FieldSetDescriptorRegister.registerFieldSetDescriptor("customeritem", customerItem-
  Fields);
    for (final FieldDescriptor field : customerItemFields.getFieldDescriptors()) {
      if (ERP_ID.equals(field.getId())) {
        field.setRequired(false);
        field.setReadOnly(true);
      }
      SiteFields.add(CustomerItem.class, field);
    }
    FieldSetDescriptorRegister.registerFieldSetDescriptor("customeritem", customerItem-
  Fields);
    INIT_DONE = true;
  }

  public static boolean isThisType(JsonNode node) {
    final JsonNode typeNode = node.get("type");
    if (typeNode != null && TYPE.equals(typeNode.asText())) {
      return true;
    }
    return false;
  }
```

Koodi 2. Customer-olion alustusfunktio ja JsonNode:n tunnistusfunktio

Tuntui sinänsä oikealta, että mallin tarkistusfunktiot ovat staattisia, mutta tämä aiheuttaa sen, että tehdas hakee funktiot reflektiolla. Kokonaisuudessaan ratkaisu on toimiva mutta kömpelö ja vaatii harmittavan määrän täytekoodia per datamalli. Kehitys alkoi Java 7 päällä, jolloin staattinen interface ei ollut mahdollinen, mutta sekään ei ongelmaa ratkaise kokonaan, koska staattisia rajapintoja ei voi ylikirjoittaa.

```
@Override
protected void integratedPreInit()
{
              final List<String> variableList = new LinkedList<>();
              variableList.add("email");
              variableList.add("language;name");
              variableList.add("companyName");
              variableList.add("ERPId");
              super.preInit(variableList, "Customer", "trusted_client",
              secret", SmilehouseConstants.HYBRIS_REST_URL,
              "store/users/", SmilehouseConstants.REST_TOKEN_URL);
}
```

Koodi 3. Asiakasdatatyyppinäkymän konfigurointi ilman nappeja

Yllä on vastaava konfiguraatio geneeriseen malliin uudemmassa ratkaisutavassa. Koodi on näkymäluokassa itsessään, jossa sillä alustetaan taulukkonäkymä. Tämä ja muutama muu parannus ensimmäiseen ratkaisuun teki uusien näkymien luomisesta huomattavasti helpompaa. Muun muassa taulukko-JSON:in tunnistusta piti parantaa tätä tehdessä.

Vaiheen 1 toteutuksessa geneerinen JSON-malli oletti, että sen lapsimallit hoitavat taulukkotunnistuksen, joten geneerinen malli sanoi aina, ettei ole taulukko ja logitti error-tason ilmoituksen. Kenttien nimen valinnassa on toiminto, joka auttaa oliokenttien käsittelyssä. Konfiguroitava merkki kertoo logiikalle, että siirryttiin oliokentän sisällä olevaan kenttään, oletuksena ”;”.

7.3 Datan säilytys

Vaadinin puolesta on valmiina muutamia tiedon säilöntäluokkia, mutta ne kaikki olettavat puhuvansa korkeintaan tietokannan kanssa. Demoamiskäytössä on miltei paras ratkaisu, että dataa säilövä ja hallinoiva kerros hakee datan uudestaan vähintään käynnistyksen jälkeen. Alustavasti Ilves-paketin mukana tulevaa tietokantaa ei käytetä edes olioiden cachena.

Säilytyskerros hoitaa keskustelun kommunikaation kanssa, esityskerrokseen jää pääasiassa tarve kertoa rajapinnan kontaktipisteiden osoitteet ja käyttötunnukset. Tunnukset voisi paketoida properties-tiedostoon ja antaa esityskerroksessa vain tietueen avain, mutta sille ei ole tällä hetkellä ollut vielä tarvetta. Säilytysolio on Vaadinin rajapintoja toteuttava, joten esityskerrokselle ei näy alustuksen jälkeen millään tapaa, että se puhuu rajapinnan kanssa eikä paikallisen tietokannan.

7.4 Esitys ja logiikka

Työssä on kaksi esimerkkinäkymäryhmää: oliopohjaiset näkymät ja geneerisiä malleja käyttävät näkymät, mutta ne sisältävät pääosin saman toiminnallisuuden. Oliopohjaiset näkymät tehtiin ensin ja ne ovat pääasiassa klooneja Ilveksen oletusnäkymistä, joista on karsittu toiminnan kannalta turhat osat pois. Pääosa tarvituista näkymistä on listoja tai editoreja. Näille tehtiin abstrakti luokka, joka hoitaa näkymän luomisen, kunhan sille vain kerrotaan, mitä ja mistä kontaktipisteestä se hakee ja minkä nimisiä kenttiä näytetään.

Eri näkymien välillä pystyy siirtymään niin, että listanäkymän valittua rivin tietoja käytetään seuraavan näkymän URI:n rakenteessa tai kutsun parametrina. Valittu tietue voi-

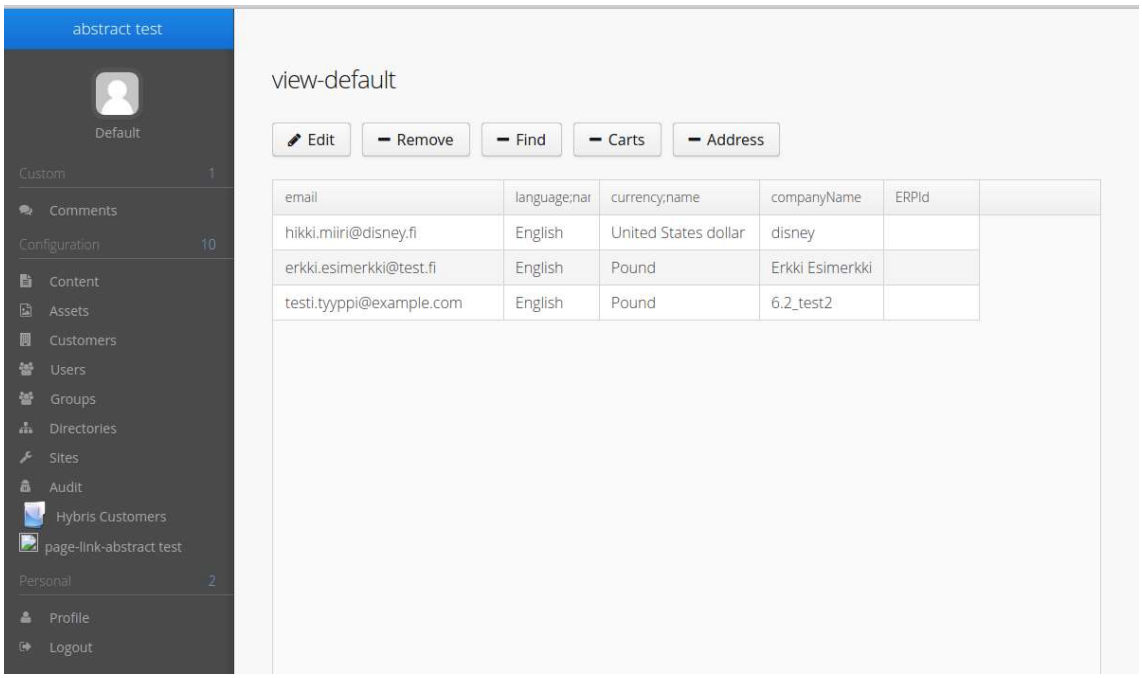
daan myös siirtää sellaisenaan editorin käsiteltäväksi. Nämä mahdollistavat REST-rajapinnan rakenteessa syvemmälle liikkumisen ja pienen lisätyön kanssa rajapinnan operoinnin yksinkertaisen web-sovelluksen tyylisesti.

7.5 Sovelluksen esittely

Tämä luku esittelee sovelluksen toimintaa lyhyesti, käyttäen erään verkkokaupan kehitysympäristön asiakasrajapintaa. Esimerkkikokoonpano esittää asiakkaiden tietoja ja mahdollistaa osan niistä muokkaamisen.


```
"users" : [ {
  "name" : "6.2_test2",
  "companyName" : "6.2_test2",
  "currency" : {
    "active" : false,
    "isocode" : "GBP",
    "name" : "Pound",
    "symbol" : "£"
    },
  "defaultAddress" : {
    "country" : {
      "isocode" : "gb"
    },
    "firstName" : "Mikki",
    "id" : "8816933502999",
    "lastName" : "Hiiri",
    "line1" : "Mannerheimintie 1",
    "line2" : "",
    "postalCode" : "00200",
      "town" : "Helsinki"
    },
    "email" : "testi.tyyppi@exam-
ple.com",
    "firstName" : "6.2_test2",
    "language" : {
      "active" : true,
      "isocode" : "en",
      "name" : "English",
      "nativeName" : "English"
    },
    "lastName" : "",
}, {
  "name" : "Erkki Esimerkki",
  "companyName" : "Erkki Esimerkki",
  "currency" : {
      "active" : false,
      "isocode" : "GBP",
      "name" : "Pound",
      "symbol" : "£"
   },
   "defaultAddress" : {
      "country" : {
          "isocode" : "gb"
      },
      "id" : "8816573054999",
      "line1" : "London street",
      "line2" : "110",
      "postalCode" : "123456",
      "town" : "London"
   },
   "email" : "erkki.esimerkki@test.fi",
   "firstName" : "Erkki",
   "language" : {
      "active" : true,
      "isocode" : "en",
      "name" : "English",
      "nativeName" : "English"
   },
   "lastName" : "Esimerkki",
}, {
   "name" : "disney",
   "companyName" : "disney",
   "currency" : {
      "active" : true,
      "isocode" : "USD",
      "name" : "United States dollar",
      "symbol" : "$"
   },
   "defaultAddress" : {
      "country" : {
         "isocode" : "us"
      },
      "firstName" : "Name",
      "id" : "8816605954071",
      "lastName" : "Last",
      "line1" : "1234 K Some Way",
      "line2" : "123",
      "postalCode" : "00000",
      "region" : {
         "isocode" : "AZ"
      },
      "town" : "Town"
   },
   "email" : "hikki.miiri@disney.fi",
   "firstName" : "disney",
   "language" : {
      "active" : true,
      "isocode" : "en",
      "name" : "English",
      "nativeName" : "English"
   },
   "lastName" : "",
} ]
}
```


Kuva 4. Raaka JSON-vaste palvelimelta, kun kutsuttu asiakaslistan palauttavaa toimintoa.



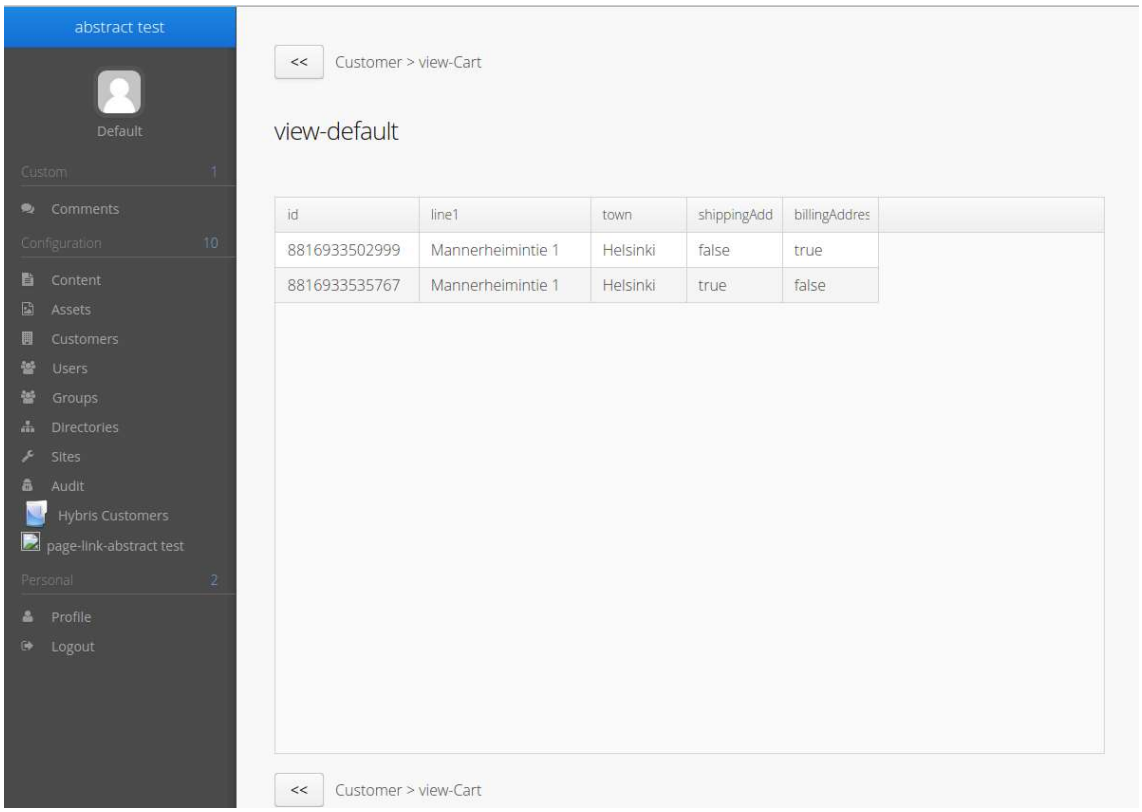
Kuva 5. Konfiguroitu asiakasnäkymä.

Kuvassa 4 on asiakashaun vaste palvelimelta raakana. Kuva 5 on asiakasnäkymä samoista asiakkaista, osa tiedoista on jätetty pois. Näkymän konfiguraatio on koodiotoksessa 2. Osoitteisiin liittyvät tiedot ovat oma näkymänsä.

Kuvan 5 vasemmassa reunassa on Ilves:n navigaatiopalkki, Hybris Customers- ja Abstract test näkymiä lukuun ottamatta kaikki linkit ovat bootstrap-paketin mukana tulleita toimintoja. ”Abstract test” -näkymältä puuttuu käännös.

Napit Edit, Carts ja Address siirtävät käyttöliittymän toiseen näkymään, valitun asiakkaan tietoja käyttäen. Kuva on esimerkkinäkymä listasta, jota voisi käyttää demossa, jossa järjestelmästä saadaan interaktiivisuuden kautta parempi kokonaiskuva. Uusi kenttä listan oliossa lisättäisiin kuvan listaan uutena sarakkeena ja uudet rajapinnat uusina nappeina.

Kuvaa 5 lukuun ottamatta kaikki esittelyn kuvat ovat näkymistä, joissa on siirrytty jonkin tietyn asiakkaan tietoihin.



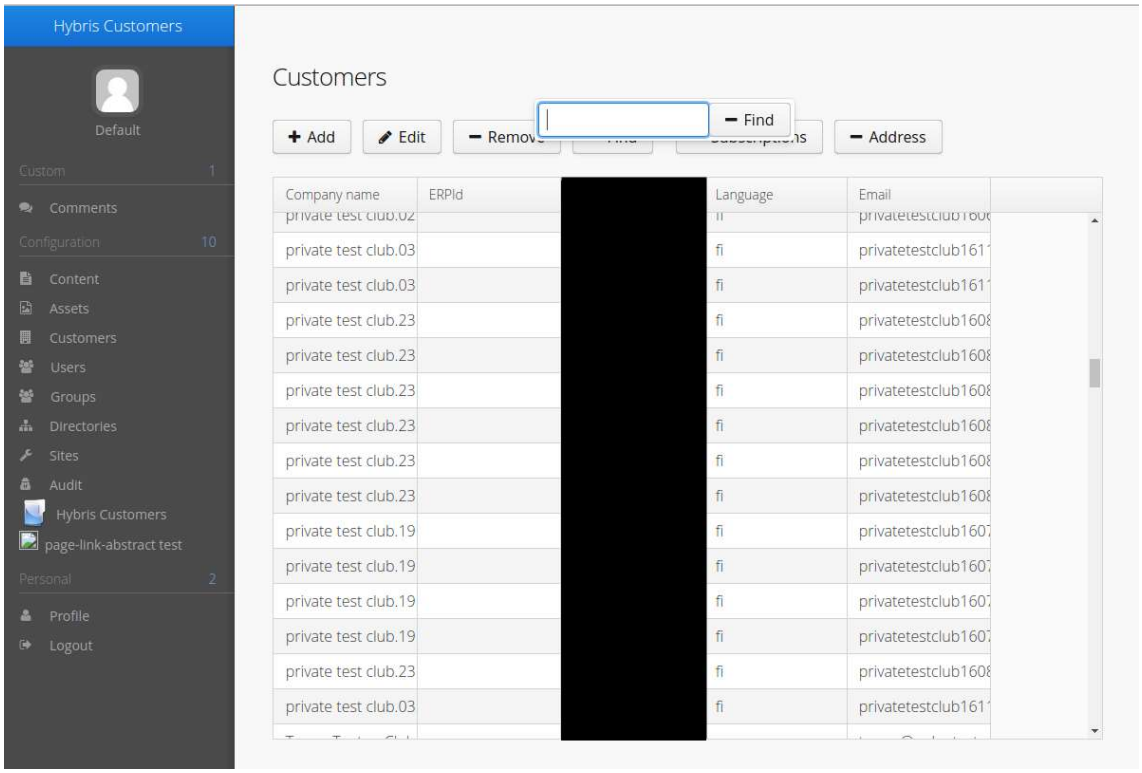
Kuva 6. Osoitenäkymä


```
"addresses" : [ {
  "billingAddress" : true,
  "country" : {
    "isocode" : "gb"
  },
  "defaultAddress" : false,
  "firstName" : "Mikki",
  "id" : "8816933502999",
  "lastName" : "Hiiri",
  "line1" : "Mannerheimintie 1",
  "line2" : "",
  "postalCode" : "00200",
  "shippingAddress" : false,
  "town" : "Helsinki"
}, {
  "billingAddress" : false,
  "country" : {
    "isocode" : "gb"
  },
  "defaultAddress" : false,
  "firstName" : "Mikki",
  "id" : "8816933535767",
  "lastName" : "Hiiri",
  "line1" : "Mannerheimintie 1",
  "line2" : "",
  "postalCode" : "00200",
  "shippingAddress" : true,
  "town" : "Helsinki"
} ]
}
```


Kuva 7. Yllä olevan osoitenäkymän sisältöä vastaava raaka JSON

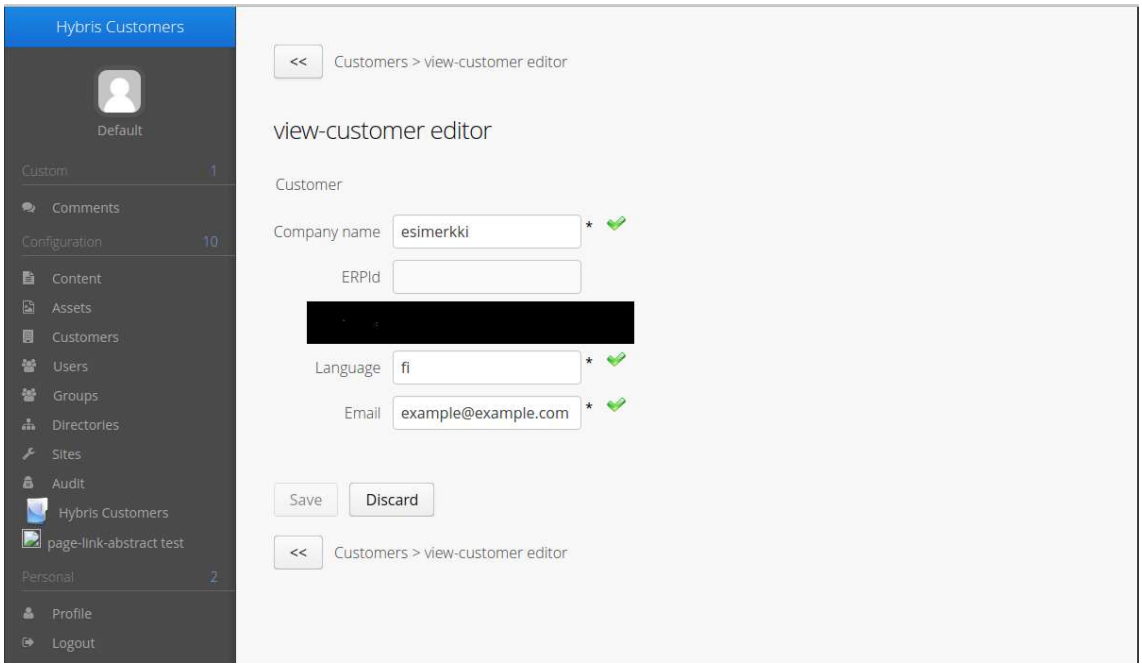
Kuva 6 on kuvan 5 alimman asiakkaan kaikki osoitteet. Kuva 7 on sama vaste raakana JSON:na. Osoitenäkymässä ylälaidassa on esimerkki Ilveksen tarjoamasta mahdollisuudesta pinota näkymiä, näkymään on siirrytty käyttäen edellisen kuvan näkymän ”Address”-napin kautta ja ”<<”-nappi siirtyy pinossa takaisin päin. Rajapinta ei tarjoa mahdollisuutta muokata osoitteita ja oliomalliratkaisun päälle uusien luominen vaati turhan paljon täytekoodia, joten näkymä vain listaa osoitteet.

Jos API sisältäisi linkit REST-määritelmän mukaisesti, tämän tyylinen pinoaminen voisi tapahtua automaattisesti, mutta nyt sen joutuu valmistelemaan käsin etukäteen, osittain sen takia, että Ilves käyttää näkymäkartassa luokkia avaimina.



Kuva 8. Hakunapin toiminnallisuus

Hakutoiminto (kuva 7) avaa oman pienen layoutinsa. Tällä tavalla etsinnän tarvitsema tekstikenttä ei ole esillä häiritsemässä turhaan muuta näkymää. Haku etsii rajapinnan läpi uuden tietueen näkymään käyttäen syötettyä tekstiä päätunnisteena.



Kuva 9. Asiakkaan luontinäkymä, oliomallin päällä. Yksi kenttä poistettu kuvasta.

Editori-näkymässä (kuva 8) voidaan valmistella sekä muokkaus- että luontikomentoja rajapintaa varten. Esimerkissä ajettava rajapintakomento valitaan sen mukaan, onko REST-tietueen oliokuvaus sitä mieltä, että se on uusi tietue vai pelkästään muokattu.

7.6 Toteutuksen puutteet ja jatkokehitys

Sovelluksen REST-määritelmään itseensä liittyvät puutteet liittyvät pääasiallisen testirajapinnan puutteisiin.

Traversal/Hypermedialinkkien tuki puuttuu. Koska pääasiallinen testiprojekti ei käytä ohjautuvaa hierarkiaa, ei toteutus tätä tue. Tämän tukeminen mahdollistaisi automaattigeneroituja sivuja eri toiminnoille. Linkkien hyödyntäminen nopeuttaisi pystyyn laittamista rajapinnoille, jotka linkittävät komentonsa toisiinsa niin kuin REST oikeasti vaatii. Tämä vaatisi muutoksia Ilveksen koodiin.

Code-on-demand on jäänyt pois samasta syystä kuin edellinenkin. Code-on-demand on muutenkin vapaaehtoinen toteuttaa, joten sen puute ei ole niin huomattava. Myöskään yrityksen sisäisissä rajapinnoissa ei pitäisi olla yleensä tarvetta sille. JavaScript code-

on-demand -järjestelmän lisääminen ei pitäisi olla vaikeaa, koska Ilves injektoi jo muutenkin luomaansa JavaScriptiä sivuille.

Nykyisellään järjestelmä ei kykene luomaan dynaamisen määritelmän JSON-rakenteita nollasta, tai muokkaamaan niiden sisältöä. Muokkaaminen ja muokkauksen lähetys olisi mahdollista toteuttaa melko nopeasti. Uuden rakenteen luominen nollasta ja lähetys rajapintaan luontikomentona vaatisi isompaa työtä. Nämä toiminnallisuudet olisin toivonut ehtiväni tehdä, mutta töiden puolesta on ollut kiirettä.

Swaggerillä on valmis jäsennin Java-kirjasto, jolla saisi dokumentaation luettua sisään. Yksinkertainen käsittely dokumentaatiosta näkymäksi ei pitäisi olla vaikea, mutta Vaadin ei ole suunniteltu dynaamisiin näkymiin. OOTB-näkymä käsittely ottaa Class-olion map-rakenteen avaimeksi, joten rakennetta joutuisi muuttamaan. Jos ylätason näkymiä halutaan useita, niin muutokset voivat olla melko radikaaleja.

8 Loppusanat

Insinöörityössä tutustuttiin rajapintoihin yleensä, REST:iin käsitteenä ja REST-rajapintoihin tarkemmin. Työn osana ohjelmoitiin sovellus, joka mahdollistaa yksinkertaisten geneeristen näkymien luonnin REST-rajapintojen pohjalta joko valmiisiin pohjiin tai räätälöityihin näkymiin.

Sovelluksen yhtenä tavoitteena oli, että uuden rajapinnan kanssa toimiminen olisi mahdollisimman helppoa, ja koen, että tässä on pääosin onnistuttu. Käyttöliittymäkehityksen työkaluvalintana koen Vaadin+Ilves -kombon olleen hyvä sekä sen tarjoaman UI:n ohjelmointitavan sekä sen, että valinta mahdollisti koko työn koodin kirjoittamisen ja jatkokehittämisen Javalla. Yksi tavoiteista olikin, että käyttäjällä olisi jatkokehittämiseen mahdollisimman pieni kynnys, joka siis edistyi huomattavasti samalla valinnalla.

Sovelluksessa on muutamia selkeitä parannettavia kohtia, joista itse koen abstrakien rajapintatietueiden muokkaamisen ja luonnin tärkeimpänä. Tämä johtuu siitä, että se on puutteista ainut, jonka olisi voinut täyttää käyttäen pääasiallista testirajapintaa. Kaikki muut vaatisivat muun rajapinnan testaamista varten.

Vahvasti geneeriseen tavoitteeseen suunnitellun ohjelman kirjoittaminen oli kiinnostava kokemus, ja kokemattomuus geneerisen koodin kanssa varmasti näkyy tehdyssä arkkitehtuurissa. Staattisen datamallin rekisteröinnin static-funktiot ja reflektiolla toimiva tehdasluokka ovat vain ne osat, jotka olen itse huomannut. Niitä ei korjattu, koska olin jo luopumassa staattisesta mallista dynaamisemman määrittelyn korvatessa sitä. Dynaaminen malli valitettavasti jäi osin kesken.

Pienen viimeistelyn jälkeen ohjelma toivottavasti näkee käyttöä muidenkin kuin itseni käsissä.

Lähteet

1 xUnit Test Patterns, Test Double <http://xunitpatterns.com/Test%20Double.html> Gerard Meszaros 21.5.2007

2 4 Types of APIs and When to Use Them <https://spin.atomicob-ject.com/2016/03/15/api-types/> Eric Shull 15.3.2016, tarkastettu 31.5.2017

3 History of APIs <http://apievangelist.com/2012/12/20/history-of-apis/> 20.12.2012 tarkastettu 2.5.2017.

4 Private, Partner or Public: Which API Strategy Is Best For Business? <https://www.programmableweb.com/news/private-partner-or-public-which-api-strategy-best-business/2014/02/21> 21,2,2014 tarkastettu 2.5.2017.

5 Architectural Styles and the Design of Network-based Software Architectures luku 5, Roy T. Fielding 2000.

6 Architectural Styles and the Design of Network-based Software Architectures luku 4.1.1, Roy T. Fielding 2000.

7 Architectural Styles and the Design of Network-based Software Architectures luku 6.1, Roy T. Fielding 2000.

8 Richardson Maturity Model <https://martinfowler.com/articles/richardsonMaturi-tyModel.html> 18.3.2010, tarkistettu 2.5.2017.

9 REST APIs must be hypertext-driven <http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven> 20.10.2008, tarkistettu 2.5.2017.

10 Views of Record: The Single Source of Truth for the Enterprise <http://business-financemag.com/business-performance-management/views-record-single-source-truth-enterprise> Michael Abbott 1.10.2004 tarkastettu 31.5.2017

11 Should I use Vaadin Framework [closed], Jens Janssonin vastaus <https://stackoverflow.com/questions/1183801/should-i-use-vaadin-framework 4.3.2017> viimeisin muokkaus 19.9.2011, tarkistettu 2.5.2017.

12 Full-stack Java web dev with Vaadin <https://www.ibm.com/developerworks/li-brary/j-full-stack-java-web-dev-vaadin/index.html> 30.9.2015.