Joni Parpala

Web-sovellusohjelmointi Scala-ohjelmointikielellä

Tietotekniikan kandidaatintutkielma

4. elokuuta 2021

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Joni Parpala

Yhteystiedot: joni.j.parpala@student.jyu.fi

Ohjaaja: Sampsa Kiiskinen

Työn nimi: Web-sovellusohjelmointi Scala-ohjelmointikielellä

Title in English: Programming web applications in Scala

Työ: Kandidaatintutkielma

Sivumäärä: 23+0

Tiivistelmä: Nykyisten web-sovellusten kasvavat vaatimukset painostavat etsimään ratkaisuja tavanomaisten ohjelmointiparadigmojen ulkopuolelta. Tässä tutkielmassa selvitetään, miten funktio-ohjelmoinnin ja olio-ohjelmoinnin perusperiaatteita yhdistävä Scala-ohjelmointikieli soveltuu web-sovellusten ohjelmointiin. Tutkimuskysymyksenä on ”Miten Scala-ohjelmointikieltä voi käyttää web-sovellusohjelmoinnissa?”. Tuloksena saadaan, että Scalalle on kehitetty hyviä web-sovelluskehitykseen soveltuvia sovellusviitekehyksiä, ja että kielen ydintoiminnallisuuksia, kuten tehokasta moniajoa ja tiivistä syntaksia, sekä kielen laajaa ohjelmakirjastotarjontaa voidaan hyödyntää web-soveluskehityksessä tehokkasti.

Avainsanat: Funktio-ohjelmointi, Scala, web-sovellus, REST

Abstract: The increasing demands posed on modern web applications put pressure on utilizing solutions outside of conventional programming paradigms. This thesis investigates how the Scala programming language, which incorporates principles from both functional and object-oriented programming models, can be utilized to program web applications. The research question is ”How can the Scala programming language be utilized for programming web applications?”. The result is that Scala has a wide variety of application frameworks well suited for web application programming, and that the core features of the language, such as the efficient approaches to multithreading, the compact syntax and the language’s vast ecosystem can be efficiently utilized in programming web applications.

Keywords: Functional programming, Scala, web, application, REST

Kuviot

Sisällys

1 Johdanto

Suuri osa ihmisten tietokoneella käyttämästä ajasta kuluu erinäisillä web-sivuistoilla. Esimerkiksi sosiaalisen median vakiintuminen tavanomaiseksi kommunikointiväyläksi 2010-luvulta lähtien on kasvattanut web-sivustojen kysyntää räjähdysmäisesti. Samanaikaisesti yhä useammat oppilaitokset, yritykset ja järjestöt digitalisoivat infrastruktuuriaan ja siirtävät toimintaansa yhä enemmissä määrin internettiin. Räjähdysmäisen kysynnän mukana ovat tulleet myös alati kasvavat vaatimukset. Perinteiset, ainoastaan staattista sisältöä esittävät web-sivustot eivät enää vastaa kaikkiin nykypäivän tarpeisiin. Käytännössä ihmisille on syntynyt tarve käyttää selaimen välityksellä sellaisia sovelluksia, joilla on joskus enemmän ominaisuuksia kuin tyypillisellä käyttöjärjestelmäkohtaisilla työpöytäsovelluksilla – on syntynyt tarve web-sovelluksille.

Kasvaneiden toiminnallisuusvaatimusten lisäksi web-sovellukset voivat joutua tietyissä tapauksissa hallinnoimaan satojatuhansia tai jopa miljoonia käyttäjiä, joista suuri osa käyttää sovellusta yhtäaikaisesti ja ympäri maailmaa. Nämä vaatimukset ovat aiheuttaneet muutospaineita monille perinteisille ohjelmistonkehityksen paradigmoille. On syntynyt tarve kehittää ennalta arvattavia, sivuvaikutuksettomia tietokoneohjelmia, jotka hyvän suorituskyvyn lisäksi pystyvät vastaamaan erilaisiin tietoturvahaasteisiin, nopeaan jatkokehitykseen ja ylläpitoon paremmin kuin aiemmat, pelkästään käyttöjärjestelmien työpöytäympäristöissä käytetyt sovellukset.

Vastausta modernien web-sovellusten haasteisiin on etsitty tyypillisten teollisuudessa käytettyjen käytänteiden ulkopuolelta. Osaksi vastausta on muodostunut jo pitkään tietojenkäsittelytieteessä tunnettu funktionaalinen ohjelmointiparadigma. Funktio-ohjelmoinnin uskotaan osaksi vastaavaan juuri sellaisiin ohjelmoinnin haasteisiin, joita modernissa web-sovelluskehityksessä esintyy. Scala-ohjelmointikieli on kehitetty ottamaan tarvittavat hyödyt lujatekoisesta ja tulikastetusta olio-ohjelmointiparadigmasta ja yhdistämään ne funktionaalisen ohjelmoinnin periaatteiden kanssa.

Tutkimuskysymyksenä on "Miten Scala-ohjelmointikieltä voi käyttää web-sovellusohjelmoinnissa?". Luvussa kaksi esitellään tyypillisimmän funktio-ohjelmoinnin periaatteet. Luvussa kolme tu-

tustutaan Scala-ohjelmointikieleen. Luvussa neljä tutkitaan yleisiä modernin web-sovelluksen toimintaperiaatteita ja lopuksi luvussa viisi tutkitaan Scala-ohjelmointikielen soveltuvuutta web-sovellusohjelmointiin sekä toteutetaan pieni yksinkertainen web-palvelimella suoritettava ohjelma. Kuudennessa luvussa kootaan yhteen tärkeimmät johtopäätökset.

2 Funktio-ohjelmointi

Funktio-ohjelmointi on pohjimmiltaan *funktioiden*, *muuttujien* ja *arvojen* toiminnan käsittelyä matemaattista lähestymistapaa käyttäen, mikä on tyypillisesti vähemmän käytetty lähestymistapa useissa muissa ohjelmointikielissä (Wampler 2011a, s. 7). Nykyaikaisissa funktionaalisissa ohjelmointiviitekehyksissä tietokoneohjelmat esitetään algoritmisuuden sijaan mielummin algebrallisesti. Tämä tarkoittaa sitä, että funktionaalisilla kielillä kirjoitetut tietokoneohjelmat ovat yhdistelmä tiettyjä alkeellisia tietotyyppejä ja niiden operaattoreita. (Kmetiuk 2018, s. 8.)

Ensimmäinen funktionaalisen ohjelmoinnin periaatteita käyttävä ohjelmointikieli on 1950-luvun lopulla kehitetty Lisp, joka lukeutuu Fortranin ohella vanhimpiin korkean tason ohjelmointikieliin. Myöhemmin, 1970-luvulla, funktionaalisia periaatteita ryhtyivät soveltamaan myös ML-kieliperheeseen kuuluvat Caml sekä Microsoftin F#. Kenties lähimmäksi funktio-ohjelmoinnin perusperiaatteiden toteuttamista pääsee Haskell-ohjelmointikieli, jonka kehittäminen aloitettiin 1990-luvun alussa. (Wampler 2011a, s. 8.)

Funktionaaliset ohjelmointikielet ja funktionaalinen ohjelmointitapa sisältävät useita alikäsitteitä ja -käytänteitä. Voitanee kuitenkin katsoa, että funktionaalisen ohjelmoinnin ydin redusoituu kahteen periaatteeseen: arvojen muuttumattomuuteen ja deklaratiiviseen ohjelmointitapaan. Käsitellään aluksi muuttumattomuuden periaatetta. Otetaan esimerkiksi Pythagoraan lause:

$x^2 + y^2 = z^2$

Asettamalla muuttujille x ja y arvot $x = 3$ ja $y = 4$, saadaan laskettua, että $z = 5$. Tässä operaatioiden sarjassa avainasemassa on se, että muuttujien arvot eivät koskaan vaihdu. Matemaattisessa asiayhteydessä sellainen operaatio kuin `3++` kuulostaisi erikoiselta – olisi luontevampaa kirjoittaa uusi lauseke, missä muuttujille asetettaisiin eri arvot. (Wampler 2011a, s. 8.)

2.1 Muuttumattomuuden periaate

Useimmat ohjelmointikielet eivät tee selkeää eroa arvon ja arvoon viittaavan muuttujan välille. Esimerkiksi Java-ohjelmointikielessä on käytettävä avainsanaa `final` osoittamaan, ettei arvon saaneelle muuttujalle saa määrittää uutta arvoa, joskin esimerkiksi olioiden sisäisen tilan muuttaminen on yhä mahdollista. Arvojen pitäminen muuttumattomina tuo muutamia käytännön hyötyjä. Muuttuvat arvot hankaloittavat useamman säikeen hyödyntämistä ohjelmakoodissa. Jos useampi säie voi muuttaa samaa arvoa, arvolle suoritettavat toimenpiteet on synkronoitava oikein. Jos arvot taas ovat muuttumattomia, synkronointitoimenpiteitä ei tarvitse tehdä, mikä tekee rinnakkaisoperaatioiden suorittamisesta riskittömämpää. Tämän lisäksi arvojen muuttumattomuus parantaa ohjelmakoodin oikeellisuutta myös muilla tasoilla; muuttuvia arvoja sisältävä ohjelmakoodi on vaikeaselkoisempaa sekä sitä on vaikea testata tyhjentävästi, etenkin jos arvojen muuttamista ei ole lokalisoitu yhteen paikkaan. (Wampler 2011a, s. 8.)

2.2 Deklaratiivinen ohjelmointitapa

Funktio-ohjelmointiin liittyy tiiviisti myös käsite deklaratiivisesta ohjelmoinnista. Deklaratiiviseksi ohjelmoinniksi kutsutaan ohjelmointitapaa, jossa kuvataan haluttu lopputulos, mutta ei tapaa millä se saavutetaan. Tämän voi paremmin ymmärtää, kun vertaa deklaratiivista ohjelmointia sen vastakohtaan imperatiiviseen ohjelmointitapaan. (Kmetiuk 2018, s. 7.)

Sekä imperatiivisessa että deklaratiivisessa ohjelmointitavassa käytetään erilaisia annetun ohjelmointikielen perusominaisuuksia ja niitä yhdistelemällä saavutetaan haluttu toiminnallisuus. Nämä ominaisuudet voisivat tässä tapauksessa olla esimerkiksi silmukkarakenteita, tietorakenteita tai tietorakenteiden tapauksessa, tietorakenteille suoritettavia operaatioita, kuten elementtien poistamista, lisäämistä tai hakemista tietorakenteesta. Deklaratiiviselle ohjelmointitavalle on tyypillistä, että ohjelmoija luo uusia, omaan spesifiin käyttötapaukseensa perustuvia ominaisuuksia. Imperatiivisessa ohjelmointitavassa taasen on tapana käyttää enemmän itse kielen tarjoamia perusominaisuuksia. (Kmetiuk 2018, s. 8—9.)

Eräs hyvä tapa havainnollistaa deklaratiivisen ja imperatiivisen ohjelmointitavan eroja on tarkastella, millaisia lähestymistapoja eri ohjelmointikielet käyttävät tietorakenteiden käsit-

telyyn. Tarkastellaan esimerkkinä tilannetta, jossa merkkijonotietorakenteesta pitäisi saada suodatettua pois ne elementit, jotka eivät ala "A" -kirjaimella. Käytetään ohjelmointikielinä enimmäkseen imperatiivista lähestymistapaa käyttävää Javaa sekä enemmän deklaratiivista ohjelmointitapaa noudattavaa Scalaa. (Kmetiuk 2018, s. 13—14.)

```
// Suodata merkkijonolistasta talteen kaikki "A" -kirjaimella alkavat nimet
// Toteutus Java-ohjelmointikielellä
List<String> tyontekijat = new ArrayList<String>();
tyontekijat.add("Aleksi");
tyontekijat.add("Tommi");
tyontekijat.add("Arttu");
tyontekijat.add("Sami");
List<String> tulos = new ArrayList<String>();
for (String t: tyontekijat)
    if (t.charAt(0) == 'A') tulos.add(t);
System.out.println(tulos);

// Vastaava toteutus Scala-ohjelmointikielellä
val tyontekijat = List(
    "Aleksi",
    "Tommi",
    "Arttu",
    "Sami"
)
val tulos = tyontekijat.filter(t => t(0) == 'A')
println(tulos)
```

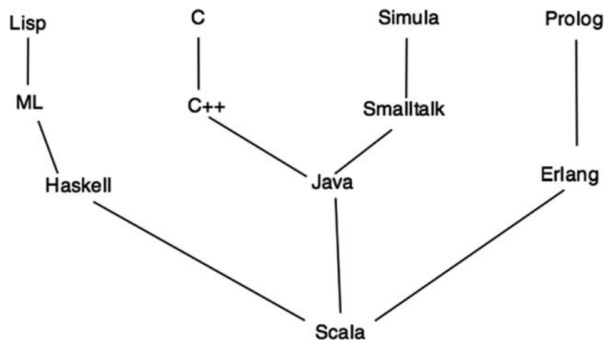
Molemmissa tapauksissa kyseessä on sama operaatio: määritellään uusi tietorakenne ja sisällytetään sen alkioita toiseen tietorakenteeseen jonkin predikaatin perusteella. Javalla tehdyssä ratkaisussa täytyy kuitenkin suorittaa useita ylimääräisiä toimenpiteitä halutun tuloksen saamiseksi, kun taas Scalan deklaratiivisemmassa tavassa tarvitsee ainoastaan määritellä operaation nimi – filter; suodata. Lyhyempään, deklaratiiviseen ohjelmointitapaan lukeutuvat edut liittyvätkin yleensä olenneisesti tiiviimpään esitystapaan, mikä parantaa ohjelmakoodin luettavuutta ja ylläpidettävyyttä. (Kmetiuk 2018, s. 14—16.)

Kun puhutaan deklaratiivisesta ja imperatiivisesta ohjelmointitavasta sekaisin funktio-ohjelmoinnin,

ja myös olio-ohjelmoinnin, kanssa, täytyy myös ymmärtää että deklaratiivinen ohjelmointi ei ole synonyymi funktionaaliselle ohjelmoinnille. Funktio-ohjelmointi on käytännössä aina deklaratiivista, mutta deklaratiivinen ohjelmointi ei ole aina funktionaalista. Esimerkiksi JavaScript-ohjelmointikielessä on olemassa eräille tietorakenteille suoritettava deklaratiivinen `forEach()` -operaatio, joka muokkaa alkuperäistä tietorakennetta, eikä täten noudata muuttumattomuuden periaatetta.

3 Scala

Scala on Martin Oderskyn kehittämä moniparadigmainen ohjelmointikieli, joka yhdistää olio-ohjelmoinnista tuttuja käytänteitä ja lähestymistapoja funktio-ohjelmoinnin kanssa. Käytännössä tämä tarkoittaa sitä, että Scala-ohjelmakoodia voi kirjoittaa tavallisin olio-ohjelmoinnissa käytetyin työkaluin, kuten luokkien, olioiden ja perinnän avulla. Scalaa voi kuitenkin käyttää myös täysin funktio-ohjelmoinnin perusperiaatteiden mukaisesti, esimerkiksi Haskellin ja Clojuren tavoin. Scala onkin saanut kehityskaarensa aikana vaikutteita useista eri ohjelmointikielistä, kuten Javasta, Haskellista ja Erlangista (Kuvio 1). Scalan kehitys aloitettiin vuonna 2004, ja se saa nimensä englannin kielen sanoista Scalable Language. (Hunt 2018, s. 1—2.)



Kuvio 1. Scalan sukupuu (Hunt 2018, s. 2.)

Scalan käyttämä hybridimallinen ohjelmointiparadigma tuo mukanaan muutamia käytännön hyötyjä. Scala käännetään Java Virtual Machine (JVM) -alustalla ajetulle Javan tavukoodille. Tämä tarkoittaa sitä, että Scalalla kirjoitettu ohjelmakoodi on yhteensopivaa Java-koodin kanssa, joten Scalaa voi käytää esimerkiksi vanhan Javalla kirjoitetun ohjelmistoprojektin laajentamiseen. Scala pystyy myös hyödyntämään suurta osaa Javalle tarjolla olevista ohjelmakirjastoista. (Hunt 2018, s. 2—3.)

Kun Scalaa kehitettiin, tavoitteena oli luoda skaalautuva ohjelmointikieli komponenttipohjaisille sovellusohjelmille, jotka toimivat rinnakkaisuutta vaativissa ympäristöissä. Tämän johdosta Scalassa on useita laajoja ohjelmistoprojekteja hyödyttäviä ominaisuuksia. Esimerkiksi kielen Actor -järjestelmä yksinkertaistaa rinnakkaislaskennan toteuttamista ohjelma-

koodissa. Scalan syntaksi on myös esimerkiksi Javaan verrattuna esitystavaltaan tiiviimpää, mikä nopeuttaa ohjelmakoodin kirjoittamista. (Hunt 2018, s. 3.)

Nykyään Scalaa voi soveltaa useisiin eri käyttötapauksiin. Scala.js mahdollistaa Scalan käytön selainohjelmoinnissa laajalti yhteensopivin JavaScript kirjastoin ja viitekehyksin (Sharma 2018, s. 5). Scalaa voi myös käyttää skriptiohjelmointiin tai jopa Unix-tyyppisen käyttöjärjestelmän shell-rajapintana, Li Haoyin kehittämän Ammoniten avulla (Haoyi 2021b).

3.1 Ohjelmointi Scalalla

Kuten aiemmin on todettu, Scalan ominaisuuksiin kuuluvat olio- ja funktio-ohjelmoinnin periaatteiden ja toimintojen yhdistäminen sekä verrattain yksinkertaistettu syntaksi. Käydään seuraavaksi läpi Scalalla ohjelmointia käytännössä. Luodaan sekä Scala- että Java-ohjelmointikielillä luokka nimeltä `Henkilo`, jolla on attribuutteina etu- ja sukunimi sekä ikä ja näille vastaava aksessorit (Hunt 2018, s. 41).

```
class Henkilo {
 public final String etunimi;
 public final String sukunimi;
 public final int ika;

 public Henkilo(final String etunimi, final String sukunimi, final int ika) {
   this.etunimi = etunimi;
   this.sukunimi = sukunimi;
   this.ika = ika;
 }
}

// Aiempaa vastaava luokka Scala-koodina:
class Henkilo(var etunimi: String, var sukunimi: String, var ika: Int)
```

Scalalla kirjoitettu versio on eittämättä lyhyempi. Useat Java-työkalut, kuten Eclipse, tarjoavatkin automatisoituja toimintoja uusien Java-luokkien luomiseksi, jolloin verbositeetti ei ole ongelma. Kuitenkin suuren koodimäärän vuoksi esimerkiksi ohjelmointivirheiden löytäminen Java-luokasta voi olla vaikeampaa Scalan vastaavaan verrattuna. (Hunt 2018,

s. 42.) Scala-luokkiin ei myöskään tarvitse erikseen määritellä metodeita luokan attribuuttien muokkaamiselle; `var` -avainsana luo sekä asettavat että ottavat aksessorit, kun taas `val` luo ainoastaan ottavat aksessorit (Hunt 2018, s. 81).

Scala tarjoaa useita, Javasta poikkeavia tapoja määritellä funktioita. Funktioita voidaan toteuttaa muista ohjelmointikielistä tutuilla metodeilla tai lokaaleilla funktioilla, mutta Scala mahdollistaa myös niin sanottujen ensimmäisen luokan funktioiden käytön. Ensimmäisen luokan funktiot ovat funktioita, joita voi kirjoittaa nimettöminä literaaleina ja esimerkiksi asettaa muuttujiin. (Odersky, Spoon ja Venners 2016, s. 180, 184.)

```
// Funktioliteraalin asettaminen muuttujaan
var korota = (x: Int) => x + 1

// Luokan metodi
class Henkilo(var etunimi: String, var sukunimi: String, var ika: Int) {
    def tulostaTiedot = println(etunimi, sukunimi, ika)
}
```

Koska muuttuja määritellään vaihtelevaksi `var` -avainsanaa käyttäen, se voidaa ylikirjoittaa tavallisen muuttujan tavoin, esimerkiksi kokonaisluvulla. Muuttujaan asetettua funktioita voidaan kutsua samalla tavoin kuin lokaaleja funktioita tai metodeja: `korota(10)`. (Odersky, Spoon ja Venners 2016, s. 184.) Edellämainitusta esimerkistä tulee ilmi myös eräs toinen Sclan syntaksiin liittyvä seikka: muuttujien tyyppejä ei tarvitse välttämättä määritellä eksplisiittisesti, vaan Scalan kääntäjä päättelee ne sijoitettavan elementin tyypin perusteella. (Elahi 2019, s. 52).

4 Web-sovellukset

Web-sovellus on sovellus, jota käytetään web-selaimella (Coleman ja Messenlehner 2019, s. 1). Perinteisen web-sivuston voidaan katsoa koostuvan web-selaimen esittämistä, suoraan HTTP-protokollan avulla palvelimelta ladatuista HTML-, CSS- ja JavaScript-tiedostoista. Vastaavasti modernit web-sovellukset hyödyntävät HTTP-protokollaa huomattavasti enemmän. Web-sovellusten voidaan itse asiassa katsoa koostuvan kahdesta tai useammasta sovellusohjelmakokonaisuudesta, jotka kommunikoivat keskenään tietoliikenneprotokollien avulla. Esimerkiksi monet käyttöliittymän toiminnoista vastaavista selainsovelluksista käyttäytyvät nykyään laajalti samoin tavoin kuin tyypilliset työpöytäsovellukset – ne ovat vastuussa omasta elinkaarestaan, tekevät palvelupyyntöjä tarvitsemalleen datalle automatisoidusti, eikä niiden esittämiä web-sivuja tarvitse ladata erikseen sovelluksen alustavan käynnistyksen jälkeen. Tämän lisäksi tyypilliseen web-sovellukseen voi kuulua muun muassa yksi tai useampi tietokantapalvelin, palvelimet käyttäjien autentikoinnille ja autorisoinnille, yksi tai useampi web-palvelin palvelinohjelmistoineen sekä rajapintatoteutukset sovelluksen eri osien kommunikaatiotarpeisiin. (Hoffman 2020, s. 27—28)

4.1 Web-sovellusviitekehykset

Web-sovellusviitekehykset (web application framework) ovat ohjelmistotyökaluja, joiden avulla web-sovelluksia rakennetaan ja ajetaan. Web-sovellusviitekehykset takaavat sen, että kaikki sitä käyttävät ohjelmistoprojektit noudattavat karkeasti yhtenäisiä lainalaisuuksia ja käytänteitä. Ennen web-sovellusviitekehysten olemassaoloa web-sovellukset olivat pitkälti sidottu kehittäjiinsä, mikä teki niiden toimittamisesta, ylläpidosta ja jatkokehityksestä erittäin työlästä. (Ryabtsev 2021)

Periaatteessa web-sovellusviitekehys on ainoastaan ohjelmakirjasto, joka tarjoaa web-sovellusprojektille eräänlaisen perusinfrastruktuurin syvällisemmälle ohjelmistokehityskelle. Tyypillisesi web-sovellusviitekehys tarjoaa sovellukselle muun muassa URL-reitityksen, HTTP-pyyntöjen jäsentelyn ja rakentamisen, tietokantaliittimet sekä erilaisia tietoturvaominaisuuksia, kuten sisäänrakennetun toiminnallisuuden SQL-injektioiden estämiselle.

(https://www.goodfirms.co/ 2020)

Koska modernit web-sovellukset koostuvat useista osakokonaisuuksista, myös web-sovellusviitekehykset ovat erikoistuneet eri käyttötapauksiin. Käytännössä sovellusviitekehykset ovat jakautuneet tukemaan joko selaimessa tai palvelimella suoritettavien ohjelmien kehitystä, joskin jotkut viitekehykset tukevat sekä palvelin- että selainsovellusten ohjelmointia. Sekä selain-, että palvelinohjelmia voi kirjoittaa lukuisia eri ohjelmointikieliä käyttäen, joskin selainohjelmia kirjoitetaan usein JavaScript-ohjelmointikielellä ja myös muilla ohjelmointikielillä kirjoitetut selainsovellukset kääntyvät JavaScriptiksi. Selainohjelmointiin erikoistuneita viitekehyksiä ovat muun muassa React, Vue.js sekä Angular. Vastaavasti palvelinsovelluskehityksessä voidaan käyttää esimerkiksi Pythonin Django- tai PHP:n Zend-viitekehyksiä (Ryabtsev 2021).

4.2 REST-rajapinnat

Eräs yleinen web-sovellusten keskinäiseen kommunikointiin käytetty rajapinta on nimeltään Representional State Transfer eli REST. Näiden rajapintojen ainoana tehtävänä on välittää palvelupyyntöjä niihin kytkettyjen sovellusten välillä – esimerkiksi sovelluksen sisäisistä tietomalleista tai -rakenteista ei lähetetä mitään tietoa. Tämänkaltaisen toimintamallin vuoksi REST-rajapintoja voikin kutsua tilattomiksi rajapinnoiksi. (Hoffman 2020, s. 28.)

REST määriteltiin ensimmäisen kerran Roy Fieldingin väitöskirjassa Architectural Styles and the Design of Network-based Software Architecture. REST:iä ei ole millään tavoin standardisoitu – siitä ei esimerkiksi ole olemassa RFC-dokumentaatiota. REST:iä voisi pikemminkin luonnehtia yhdeksi rajapintasuunnittelun tyylisuuntauksista. Varsinaisen standardimäärittelyn sijaan REST esittelee rajapintasuunnitteluun tiettyjä arkkitehtuurillisia rajoitteita, joita useimmat web-sovellukset noudattavat yleensä vain osittain. (Doglio 2015, s. 1—3.)

REST:n määrittelemät suunnittelurajoitteet ovat asiakas-palvelin -mallin käyttö, rajapinnan tilattomuus, selaimen välimuistin tehokas käyttö sekä resurssiorientoitunut ja yhtenäinen palvelupäätepisteiden nimeäminen. Asiakas-palvelin -mallissa asiakkaat ja palvelimet ovat eriytetty toisistaan täysin – esimerkiksi web-käyttöliittymä ei ole kiinnostunut missä muodossa web-palvelin tallentaa sen käsittelemän tiedon ja vastaavasti web-palvelin ei ota kantaa missä

muodossa sen lähettämä data esitetään käyttöliittymässä. (Kanjilal 2013, s. 9—10.)

REST perustuu tilattomaan HTTP-protokollaan. Jokaisen asiakkaan tekemän palvelupyynnön täytyy sisältää riittävästi sellaista tietoa, jonka perusteella palvelin voi vastata pyyntöön. Asiakkaasta ei kuitenkaan tallenneta palvelimelle mitään sovelluksen tilasta identifioivaa dataa. Tämän johdosta palvelimet skaalautuvat paremmin ja niiden suorituskykyä voidaan seurata helpommin. (Kanjilal 2013, s. 9.)

Tyypillisessä REST-arkkitehtuurissa palvelimien vastaukset pitäisi pystyä tallentamaan asiakkaiden välimuistiin. Välimuistin hallinnan kannalta voidaan kuitenkin eritellä mitkä palvelimen vastauksista tallennetaan. Tämä parantaa sovelluksen skaalautuvuutta ja suorituskykyä. (Kanjilal 2013, s. 9.)

REST-arkkitehtuurityyli määrittelee yhtenäisen rajapinnan asiakkaan ja palvelimen välille. Tämän vuoksi palvelupyynnöille ja -vastauksille on määritelty rajattu määrärä HTTP-standardin mukaisia operaatioita, kuten `GET`, `POST`, `PUT` ja `DELETE`. Rajapintaoperaatiden täytyy REST-arkkitehtuurin mukaisesti palauttaa aina jokin URI-tunnisteinen resurssi. Resurssipohjainen lähestymistapa rajapintasuunnitteluun onkin kaikista tärkein osa REST-arkkitehtuuria. On kuitenkin huomioitava, että REST ei määrittele välitettävän datan tallennusformaatteja. Yleisesti käytettyjä tallennusformaatteja ovat esimerkiksi HTML, JSON ja XML. (Kanjilal 2013, s. 10.)

Koska REST-rajapinnat kommunikoivat HTTP-pyynnöillä, toteutuva asiakas-palvelin -kommunikaatio voisi olla esimerkiksi seuraavanlainen:

```
# Pyydetään REST-rajapinnalta kaikki tiettyyn opiskelijaan liittyvät tiedot
GET https://www.web-sovellus.fi/opiskelijat/813482 HTTP/1.1


# Palvelimen vastaus (JSON-muodossa)
{
    nimi: "Aapeli Käki",
    opiskelijatunnus: 813482,
    syntymaAika: "1.5.1997",
    kurssit: [
        {
```

```
            nimi: "Ohjelmoinnin perusteet",
            opettaja: "V.L."
        },
        {
            nimi: "Tietokantojen perusteet",
            opettaja: "T.T."
        }
    ]
}
```

5 Scalan käyttö web-sovellusohjelmoinnissa

Toiseksi viimeisessä luvussa käsitellään Scala-ohjelmointikielen käyttöä web-sovellusohjelmoinnissa. Aluksi käydään läpi muutamia Scalalle kirjoitettuja web-sovellusviitekehyksiä ja niiden eri käyttötapauksia. Lopulta toteutetaan yksinkertainen REST-rajapinnan toteuttava web-palvelinsovellus Scala-ohjelmointikieltä käyttäen.

5.1 Scalan web-sovellusviitekehykset

Scalan ympärille on rakennettu useita web-sovellusviitekehyksiä. Näistä suosituin on myös Java-ohjelmointikieltä tukeva Play-sovellusviitekehys (www.scala.libhunt.com 2021). Playn tavoite on tarjota helppokäyttöinen, vaivattoman jatkokehityksen takaava web-sovellusviitekehys, jonka keskiössä ovat muun muassa hyvä tuki REST-rajapinnoille sekä tehokas resurssien käyttö ja hyvä suorituskyky (www.playframework.com 2021).

Muita Scalaa tukevia web-sovelluskehyksiä ovat muun muassa Scalatra sekä Twitterin kehittämä Finagle. Scalatra on erikoistunut pienille, helposti hallittaville web-sovellusprojekteille, joille ei ole Play-sovelluksen tavoin tarkoitus omistaa suuria määriä jatkokehitystä. Vastaavasti Scalatran käyttö vaatii Play-sovelluskehykseen verrattuna huomattavasti vähemmän ohjelmakoodia esimerkiksi yksinkertaisten REST -perustoimintojen saavuttamiseksi. (Wampler 2011b.)

Sekä Play- että Scalatra-viitekehyksillä pystyy tarvittaessa rakentamaan web-sovelluksen selain- ja palvelinsovelluksen toiminnallisuudet – ne eivät tarvitse mitään kolmannen osapuolen tarjoamaa selainohjelmointiin erikoistunutta viitekehystä. Vastaavasti Finagle on erikoistunut tukemaan palvelinsovellusten ohjelmointia – sen kehityspäämäärä oli alunalkaen pystyä vastaamaan Twitterin vaatimiin valtaviin suorituskyky- ja jatkokehitysvaatimuksiin. Finagle esimerkiksi tarjoaa sovellukselle kuormantasausta, automatisoitua uudelleenkäynnistystä, edistynyttä virheenhallintaa sekä lukuisia tietoturvaominaisuuksia. (Wampler 2011b.)

Finaglea vastaava viitekehys selainohjelmien tekoon on Scala.js. Scala.js:n avulla selainoh-

jelmia voidaan kirjoittaa suoraan Scala-ohjelmointikieltä käyttäen, jolloin Scalaa on mahdollista käyttää sekä selaimella että palvelimella ajetuissa ohjelmissa. Käytännössä Scala.js:llä kirjoitettu ohjelma käännetään erittäin tehokkaaksi JavaScript ohjelmakoodiksi. Scala.js:ää on myös mahdollista käyttää yhdessä muiden selainohjelmille tarkoitettujen viitekehysten kanssa. (https://www.scala-js.org/ 2021.)

5.2 REST-rajapinnan toteuttaminen Scalalla

Tarkastellaan vielä yhtä Scalalle kirjoitettua web-sovellusviitekehystä ja toteutetaan sillä yksinkertainen REST-rajapinta. Li Haoyin Cask-sovellusviitekehys on yksinkertainen, Python-ohjelmointikielen Flask-sovellusviitekehyksen inspiroima web-sovelluskehityskirjasto, jossa HTTP-palvelupäätepisteet sekä niiden tarjoamat resurssit määritellään yksinkertaisin funktioin ja annotaatioin. Ohjelma siis muodostuu käytännössä yhdestä oliosta, jonka metodit määrittävät sovelluksen tarjoamat resurssit. Cask on todella yksinkertainen ja helppokäyttöinen web-sovellusviitekehys, REST-rajapintojen toteutukseen, mutta se ei esimerkiksi Scalatrasta ja Finaglesta poiketen tarjoa suoraa tukea joillekin Scalan suorituskykyä parantaville ominaisuuksille, kuten moniajolle. (Haoyi 2021a.) Koska Scalaa käytetään tyypillisesti palvelinsovellusten ohjelmointiin, toteutetaan kokonaisten selain- ja palvelinsovellusten sijaan ainoastaan varsinaiset REST-rajapinnan vaatimat palvelinohjelman komponentit.

```
package app
object TodoMvcApi extends cask.MainRoutes {
  case class Todo(text: String)

  object Todo {
    implicit def todoRW: upickle.default.ReadWriter[Todo]
      = upickle.default.macroRW[Todo]
  }

  var todos = List(
    Todo("Get started with Cask"),
    Todo("Profit!")
  )
```

```
  @cask.get("/list")
  def list() =
    todos

  @cask.post("/add")
  def add(request: cask.Request)
    = todos = List(Todo(request.text())) ++ todos

  @cask.post("/delete/:index")
  def delete(index: Int)
    = todos = todos.patch(index, Nil, 1)

  initialize()
}
```

Käynnistetty sovellus tarjoilee kolme resurssia HTTP-palvelupisteisiin, joista kukin manipuloi sovelluksen muistissa sijaitsevaa muistiinpanojen listaa. GET-operaattorilla saadaan haettua kaikki olemassaolevat muistiinpanot, kun taas kahdella määritellyllä POST-operaatiolla voidaan joko luoda tai poistaa muistiinpanoja. (Haoyi 2021c.)

6 Yhteenveto

Web-sovellukset pyrkivät vastaamaan nykypäivän tarpeisiin laajoista, toisiinsa kytketyistä web-ympäristöistä, tietoturva-asiat, suorituskyvyn ja helppokäyttöisyyden samalla huomioon ottaen. Nykyaikaiset rajapintatoteutukset pitävät huolen siitä, että sovelluksia voidaan tarvittaessa laajentaa ja parantaa nopeasti muuttuvien vaatimusten mukaan. Funktio-ohjelmoinnin muuttumattomuuden sekä deklaratiivisuuden soveltaminen pitävät suuretkin sovelluskokonaisuudet ennalta-arvattavina ja hallittavina.

Yleisesti ottaen Scala soveltuu web-sovellusten ohjelmointiin hyvin. Scala vahvistaa monia hyväksi todettuja olio-ohjelmoinnin käytäntöjä tuomalla mukaan funktionaalisuuden ja deklaratiivisuuden periaatteita. JVM-alustalla ajetut sovellukset lukuisine viitekehyksineen vastaavat myös hyvin teollisuuden tarpeisiin – Scalalle kehitettyjä web-sovellusviitekehyksiä voidaan käyttää niin pienten ja yksinkertaisten kuin suurten ja monimutkaistenkin web-sovellusten ohjelmointiin sekä selaimessa että palvelimella.

Mahdollisia jatkotutkimuksen aiheita aiheeseen liittyen voisivat olla suorituskykyvertailut Scalan ja muiden suosittujen web-sovellusohjelmointiin käytettyjen ohjelmointikielten välillä. Luonnollisesti samankaltaista vertailua voitaisiin suorittaa myös Scalan eri web-sovellusviitekehysten välillä. Voisi olla myös mielenkiintoista selvittää, mitkä ovat pitkäikäisten Scalalla kirjoitettujen web-sovellusprojektien haasteet, ja tutkia syitä miksi Scalaa valitaan tai vastaavasti ei valita web-sovellusprojektien ohjelmointikieleksi.

Lähteet

Coleman, Jason, ja Brian Messenlehner. 2019. *Building Web Apps with WordPress: WordPress as an Application Framework.* Sebastopol, CA: O'Reilly Media, Inc.

Doglio, Fernando. 2015. *Pro REST API Development with Node.js.* La Paz, Canelones, Uruguay: Apress.

Elahi, Irfan. 2019. *Scala Programming for Big Data Analytics.* Notting Hill, VIC, Australia: Apress.

Haoyi, Li. 2021a. *About Cask.* Saatavilla WWW-muodossa, https://com-lihaoyi.github.io/cask/page/about-cask.html, viitattu 1.5.2021.

———. 2021b. *Ammonite.* Saatavilla WWW-muodossa, https://ammonite.io, viitattu 17.4.2021.

———. 2021c. *Cask: a Scala HTTP micro-framework.* Saatavilla WWW-muodossa, https://com-lihaoyi.github.io/cask/index.html, viitattu 1.5.2021.

Hoffman, Andrew. 2020. *Web Application Security: Exploitation and Modern Security Countermeasures for Web Applications.* Sebastopol, CA: O'Reilly Media, Inc.

https://www.goodfirms.co/. 2020. *What is a Web Framework?* Saatavilla WWW-muodossa, https://www.goodfirms.co/glossary/web-framework/, viitattu 3.5.2021.

https://www.scala-js.org/. 2021. *Scala.js.* Saatavilla WWW-muodossa, https://www.scala-js.org/, viitattu 8.5.2021.

Hunt, John. 2018. *A Beginner's Guide to Scala, Object-Orientation and Functional Programming.* Bath, Wiltshire, UK: Springer.

Kanjilal, Joydip. 2013. *ASP.NET Web API.* Birmingham, UK: Packt Publishing.

Kmetiuk, Anatolii. 2018. *Mastering Functional programming.* Birmingham, UK: Packt Publishing.

Odersky, Martin, Lex Spoon ja Bill Venners. 2016. *Programming in Scala.* Mountain View, California: Artima Press.

Ryabtsev, Alexander. 2021. *Web Frameworks: How To Get Started.* Saatavilla WWW-muodossa, https://djangostars.com/blog/what-is-a-web-framework/, viitattu 3.5.2021.

Sharma, Vikash. 2018. *Learning Scala Programming.* Birmingham, UK: Packt Publishing.

Wampler, Dean. 2011a. *Functional Programming for Java Developers.* Sebastopol, CA: O'Reilly Media, Inc.

———. 2011b. "Scala Web Frameworks: Looking Beyond Lift". *The Functional Web* 15:87–94.

www.playframework.com. 2021. *What is Play?* Saatavilla WWW-muodossa, https://www.playframework.com/documentation/2.8.x/Introduction/, viitattu 3.5.2021.

www.scala.libhunt.com. 2021. *Scala Web Frameworks.* Saatavilla WWW-muodossa, https://scala.libhunt.com/categories/585-web-frameworks, viitattu 1.5.2021.