Wiljam Rautiainen

Haasteet REST-arkkitehtuurityylin määrittelyssä

Tietotekniikan kandidaatintutkielma

5. tammikuuta 2020

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Wiljam Rautiainen

Yhteystiedot: wirautia@student.jyu.fi

Työn nimi: Haasteet REST-arkkitehtuurityylin määrittelyssä

Title in English: Challenges in determination of REST architectural style

Työ: Kandidaatintutkielma

Sivumäärä: 22+0

Tiivistelmä: Tässä tutkielmassa selvitetään, miksi REST-arkkitehtuurityylin määrittely on haasteellista. Tutkielmassa tarkastellaan, miten REST-arkkitehtuurityyli on syntynyt, miten se on määritelty alkuperäisessä lähteessä ja miten sen määritelmä on tarkentunut.

Lisäksi tutkielmassa selvitetään, mitä käytännön ongelmia REST-arkkitehtuurityylin toteuttaminen on aiheuttanut ja miten termiä on yritetty yksinkertaistaa. Tutkielmassa tullaan johtopäätökseen, että kehittäjät eivät täysin ymmärrä REST-arkkitehtuurityylin rajoituksia, ja termi on muuttunut yleistermiksi, mikä aiheuttaa ongelmia sen tulkinnassa.

Avainsanat: REST, RESTful, www-sovelluspalvelu

Abstract: The paper explains why determination of REST architectural style is challenging. The paper will review, how the REST architectural style has born, how it is defined in the original dissertation and how the definition has changed.

This paper will also clarify, what practical problems there are with the definition, how there has been an effort to make definition more straightforward. Conclusion of this paper is that developers lack the knowledge of REST architectural style constrains and the REST architectural style definition has become a buzzword which causes problems when the definiton is being interpreted.

Keywords: REST, RESTful, Web service

Kuviot

Sisältö

1 Johdanto

Verkkopalvelut ovat saavuttaneet suuren suosion ihmisten keskuudessa, ja internet on osa ihmisten jokapäiväistä elämää, joten näiden palveluiden kehittäminen on noussut tärkeäksi teemaksi. Tärkeänä osana verkkopalveluiden kehittämisessä on myös, miten www-sovelluspalveluita (Web service) tulisi kehittää. Tässä tutkielmassa www-sovelluspalveluilla tarkoitetaan verkkopalvelimissa toimivia ohjelmia, jotka tarjoavat palveluja sovelluksille.

REST (Representational State Transfer) tarjoaa standardin, miten www-sovelluspalveluita voidaan kehittää käyttämällä hyväksi sen arkkitehtuurista tyyliä. REST-arkkitehtuurityyli ei ole protokolla, jonka avulla www-sovelluspalveluita voidaan kehittää, vaan se tarjoaa tyylin tuottaa RESTful-palveluita (RESTful services). REST-arkkitehtuurityylin avulla voidaan ymmärtää, miten WWW (world wide web) on tarkoitettu toimivan, ja näin voidaan helpottaa laajamittaisten ohjelmistojen kehittämistä (R. T. Fielding ym. 2017).

Tämän kirjallisuuskatsaustutkimuksen päämääränä on vastata tutkimuskysymykseenn, miksi REST-arkkitehtuurityylin määrittely on haasteellista. Jotta tutkimusongelma voidaan saavuttaa, tutkielmassa analysoidaan, miten REST-arkkitehtuurityyli on luotu ja miten se elää nykyhetkessä. Tämän jälkeen tutkielmassa tarkastellaan, kuinka REST-arkkitehtuurityyli on määritelty alkuperäisessä lähteessä. Tutkielmassa esitetään myös, miten REST-arkkitehtuurin määrittely on muuttunut ja mitä ongelmia siitä on seurannut. Lopuksi tutkielmassa esitetään yhteenveto tuloksista.

Tutkielman tarkoitus ei ole ottaa kantaa, onko REST-arkkitehtuurityyli paras mahdollinen tapa toteuttaa www-sovelluspalveluita tai mitä hyötyjä-haittoja siinä on, koska tästä aiheesta on löydettävissä jo paljon tutkimuksia.

2 REST-arkkitehtuurityylin syntyminen ja nykyhetki

REST-arkkitehtuurityyli on yksi suosituimpia tapoja kehittää www-sovelluspalveluita, mutta näin ei ole aina ollut (Petrie 2016). Ennen REST-arkkitehtuurityylin luomista ei www-sovelluspalveluiden luomiseen ollut yhtenevää linjaa, ja usein näiden palveluiden kehittämiseen käytettiin tapoja kuten SOAP-protokollaa (Simple Object Access Protocol) ja CORBA-standardia (Common Object Request Broker Architecture). Molemmilla tavoilla oli kuitenkin ongelmansa, ja ne vaativat kehittäjiltä paljon tietoa niiden kehittämistavasta.

2.1 Internet ja World Wide Web

Nykyisessä internetissä on jo yli 1.2 miljardia palvelinimeä 25.11.2019 (hostname), kun taas vuonna 1995 tämä lukema oli vain noin 20 000 (Netcraft 2019). Internet kasvaa myös jatkuvasti, ja samalla se on mullistanut nykymaailman toiminnan ja ajattelun (Leiner ym. 2009).

Internet ei olisi voinut kuitenkaan saavuttaa menestystään ilman, että se toisi jotain uutta ja merkityksellistä ihmisten elämään. WWW:n laaja kasvu ja internetin kaupallistuminen mahdollistivat kuluttajien pääsyn globaaleille markkinoille, mikä edesauttoi internetin kasvua merkittävästi (Leiner ym. 2009).

Nykyisen WWW:n voidaan sanoa syntyneen Tim Bernin kirjoituksesta ”Information Management: A Proposal”, joka julkaistiin ensimmäisen kerran marraskuussa 1989 (Berners-Lee 1989). WWW:llä tarkoitetaan palvelujärjestelmää, jonka avulla voidaan julkaista ja hyödyntää verkkosivuja.

Ajatus WWW:stä ei ollut kuitenkaan täysin uusi, sillä jo vuonna 1945 Vannevar Bush oli esitellyt teorian linkkien avulla siirrettävästä datasta, ja vuonna 1962 J.C.R. Licklider muotoili ensimmäisen järjestelmän, joka pystyisi ylläpitämään sosiaalisia suhteita verkon avulla (Connolly 2000; Leiner ym. 2009). Tim Bernin määrittelyn avulla WWW on kuitenkin muotoutunut siihen muotoon, miten se nykypäivänä tunnetaan. Lokakuussa 1990 Tim Bern ohjelmoi ensimmäisen nettiselaimen nimeltään ”WorldWideWeb” sekä määritteli kolme ominaisuutta, mitä selaimen tulee sisältää. Nämä kolme ominaisuutta ovat,

- HTML (Hypertext Markup Language),
- URI (Uniform Resource Identifier),
- HTTP (Hypertext Transfer Protocol).

WWW sisälsi aluksi kuitenkin vain verkkosivuja, jotka olivat staattisia HTML-sivuja, ja niiden ainoa ominaisuus oli säilyttää tietty dokumentti sillä kuuluvassa paikassa. 1990-luvun alussa nämä staattiset sivut alkoivat muuttua kokonaisvaltaisemmiksi verkkopalveluiksi. Kun verkkopalvelut muuttuivat yhä kokonaisvaltaisemmiksi, niiden avulla voitiin luoda yhä monimutkaisempia palveluita, jotka pitivät sisällään muun muassa tietokantojen yhdistämistä sekä ohjelmistokirjastoja (Fowler ja Stanwick 2004).

2.2 REST-arkkitehtuurityylin historia

Vuonna 1993 internetin kasvoi nopeaa vauhtia, sillä web-palvelimien määrä tuplaantui joka kolmas kuukausi (R. T. Fielding ym. 2017). Internetin jatkuva kasvu aiheutti myös ongelmia, sillä uusia internetin lisäosia esiteltiin jatkuvalla tahdilla, joka aiheutti web-kehittäjien yhteisössä erimielisyyksiä (R. T. Fielding ym. 2017).

Kun Roy Fielding työskenteli projektissa, jonka tarkoituksena oli spesifioida HTTP/1.1 -standardia, hän alkoi kehittää myös http-oliomallia (Hypertext Transfer Protocol object model) (R. T. Fielding ym. 2017). R. T. Fielding ym. (2017) esittää, että mallin tarkoituksena oli spesifioida, miten erityyppisten web-applikaatioiden tulisi käyttäytyä sekä miten protokollien muutokset tulisivat vaikuttamaan näihin applikaatioihin.

Vuonna 1995 Roy Fielding liittyi Apachen HTTP-palvelinprojektiin, joka kasvoi nopeasti internetin suosituimmaksi palvelimeksi. Tämän projektin ansiosta HTTP/1.1 popularisoitui huomattavasti, ja se loi samalla pohjan jatkaa Fieldingin tutkimusta HTTP-oliomallista. Mallin nimi muuttui myöhemmin nimeksi "Representational State Transfer", ja se julkaistiin ensimmäisen kerran syyskuussa vuonna 2000 (R. T. Fielding ym. 2017).

2.3 REST-arkkitehtuurityyli nykypäivänä

REST-arkkitehtuurityylin suosioon kehittämisessä ei voida sanoa olevan tiettyä syytä, mutta esimerkiksi Fensel ym. (2011, s.68) toteavat, että REST-arkkitehtuurityyli rajaa käytettävissä olevia teknologioita positiivisella tavalla ja edesauttaa uusien web-kehittäjien oppimiskäyrää. Myöskin Renzel, Schlebusch ja Klamma (2012) toteavat, että RESTful-palvelut sopivat paremmin ad hoc -tyylisten yritysten tarpeisiin, sillä ne tarjoavat paremman käyttöiän verrattuina muihin teknologioihin.

REST-arkkitehtuurityylin kilpailevana teknologiana on voitu nähdä SOA (Service Oriented Architecture), joka on johtanut keskusteluun, kumpi on parempi tapa kehittää www-sovelluspalveluita. Kuitenkin SOA:n kehittämisen ongelmana on usein nähty, että se vaatii enemmän suunnittelua ja on vaativampi toteuttaa käytännössä (Renzel, Schlebusch ja Klamma 2012). Toteuttamisen helppous on voinut edesauttaa REST-arkkitehtuurityylin suosiota nykypäivän kehittämisessä.

Vaikka REST-tyyppiset sovellukset ovat suosittuja tapoja toteuttaa nykypäivän www-sovelluspalveluita, ei suosion jatkuminen ole varmaa. Facebookin vuonna 2017 patentoitu GraphQL web-protokollan on ennustettu kasvavan jopa REST arkkitehtuurityylin ohi, sillä se tarjoaa vielä helpomman tavan kehittää www-sovelluspalveluita (Hartina, Lawi ja Panggabean 2018).

3 REST-arkkitehtuurityylin rajoitukset

Kun puhutaan REST-arkkitehtuurityylistä, siitä voidaan käyttää termiä RESTful-palvelu. Tällä tarkoitetaan www-sovelluspalvelua, joka toteuttaa REST-arkkitehtuurityylin kaikki sille asettamat ehdot (Fensel ym. 2011). On myöskin mahdollista käyttää nimeä REST-suuntainen palvelu, jos www-sovelluspalvelua ei täytä kaikkia REST-arkkitehtuurityylin asettamia ehtoja. Tässä luvussa selvennetään, mitkä ovat REST-arkkitehtuurityylin keskeiset rajoittimet ja miten ne ovat määritelty.

REST-arkkitehtuurityylin voidaan sanoa olevan arkkitehtuurisien elementtien hajautettu abstraktio hypermediajärjestelmässä, jossa arkkitehtuuriset elementit voidaan jakaa kolmeen eri luokkaan (Fielding ja Taylor 2000). Nämä luokat ovat,

- data elementit,
- liittimet,
- komponentit.

Fielding ja Taylor (2000) määrittelevät, että asettamalla näihin edellä mainittuihin elementteihin tietyt ehdot, voidaan REST-arkkitehtuurityyli toteuttaa ja määritellä. REST-arkkitehtuurityylissä on kuusi rajoitusta, jotka on määritelty ensimmäisen kerran "Architectural styles and the design of network-based software architectures"väitöskirjassa. Tässä tutkielmassa seurataan ensisijaisesti alkuperäisen lähteen määrittelyä sekä täydennetään määrittelyä muiden lähteiden perusteella.

3.1 Yhtenäinen rajapinta (Uniform interface)

Yhtenäinen rajapinta voidaan jakaa neljään osaan, jotka sovelluksen tulee toteuttaa. Yhtenäinen rajapinta yksinkertaistaa palvelua ja varmistaa komponenttien olevan vaihdettavissa, jotta palvelun toiminta ei häiriintyisi (Fielding ja Taylor 2000). Yhdenmukainen rajapinta myös varmistaa, että asiakas ja palvelin voivat kommunikoida keskenään.

Resurssien identifikaatio (Identification of resources)
Resurssien identifikaation tarkoitus on, että kaikki resurssit voidaan identifioida yhden iden-

tifiointimekanismin avulla (R. T. Fielding ym. 2017). Käytännössä tällä tarkoitetaan, että haluttu informaatio on aina kapseloitu tiettyyn staattiseen tunnisteeseen, ja www-sovelluspalveluiden tapauksessa käytetään URL-osoitetta (Uniform Resource Locator) (Davis 2012). Fielding ja Taylor (2000) toteavat myös, että tunniste ei voi muuttua, vaikka resurssin tila muuttuisikin, vaan reusurssilla tulee aina olla olemassa jokin tunniste.

Resurssien manipulointi (Manipulation of resources through representations)

Fielding ja Taylor (2000) määrittelevät resurssien manipulaation toteamalla, että palvelun lähettämä resurssi asiakkaalle on vain resurssin esitys eikä itse resurssi. Resurssien manipulaation tarkoituksena on estää asiakasta muuttamasta resurssin tilaa suorasti. Asiakas voi siis muuttaa resurssin tilaa, mutta tämä tapahtuu ehdotuksella palvelimelle, esimerkiksi käyttämällä apuna JSON (JavaScript Object Notation) kieltä.

Esitysmuoto ja Mediatyypit (Reprenstations and Media Types)

Esitysmuodon Fielding ja Taylor (2000, s.5.2.1.2) määrittelevät siten, että esitysmuoto on sekvenssi bittejä, ja näin esitysmuodon metadata voi kuvata näitä bittejä. Esitysmuodon avulla REST-komponentit voivat suorittaa toimintoja resursseille, joissa esitysmuoto kuvaa reusurssin tulevaa tai nykyistä tilaa (Fielding ja Taylor 2000). Mediatyypeillä tarkoitetaan datan muotoa esitysmuodossa (Fielding ja Taylor 2000). Jokaisen viestin käsittelyyn käytetään mediatyyppejä, joiden avulla viestit voidaan tulkita. Esitysmuoto ja mediatyypit rajoittimen tarkoitus on, että asiakkaan ja palvelimen välisessä kommunikaatiossa kaikki oleelliset tiedot voidaan esitetään yhdessä viestissä. Davis (2012) toteaa, että tämä tapa vähentää huomattavasti liikennekatkoja liikenteen välillä.

HATEOS (Hypermedia as the engine application state)

Yleisesti tähän rajoittimeen viitataan nimellä HATEOS (Hypermedia as the engine application state). Hypermedialla tarkoitetaan hypertekstin keinoja, erityisesti hypertekstilinkkejä hyödyntävää multimediaa (Multimediasanasto (TSK 28, 1999). Fielding ja Taylor (2000) määrittelevät HATEOS-rajoittimen tarkoittavan, että palvelulla on olemassa tapa liikkua käyttämällä avuksi hypermediaa. Asiakas voi siis päätellä kahden eri resurssin välisen yhteyden sekä minkälaisia metodeja näihin resursseihin voidaan lähettää (Schreier 2011). HATEOS-rajoitin voidaan havainnollistaa helpommin kuvittelemalla palvelu, jossa ehto ei toteudu. Näin palvelussa liikkuminen voisi toteutua vain käyttäjän omilla syötteillä URL-kenttään.

3.2 Asiakas-palvelin (Client server)

Asiakas-palvelin-rajoittimen määritelmä määrittelee palvelun tarvitsevan erotetun palvelimen ja asiakkaan. Fielding ja Taylor (2000) toteavat asiakas-palvelin-rajoittimen tarkoittavan, että erottamalla nämä kaksi komponenttia toisistaan, voivat komponentit kehittyä itsenäisesti. Käytännössä asiakkaan ei tarvitse tietää palvelusta muuta kuin sen omistamat resurssit.

3.3 Tilattomuus (Stateless)

Tilattomuudella Fielding ja Taylor (2000) tarkoittavat, että jokaisen viestin tulee sisältää kaikki oleellinen, jotta palvelimen ja asiakkaan välinen kommunikaatio voidaan tulkita vain yhdellä viestillä, eikä perustua tallennettuun tietoon palvelimella. Tallennetulla tiedolla tarkoitetaan esimerkiksi kirjautumistietoja palveluun tai evästeitä. Tässä tapauksessa asiakas on aina vastuussa oman tilan tallentamisesta, jos asiakas niin päättää.

3.4 Välimuistin tallennus (Cacheable)

Fielding ja Taylor (2000) määrittelevät, että jokaiseen vastaukseen tulee ilmoittaa, onko viesti mahdollista varastoida välimuistiin. Näin välimuistin avulla voidaan kokonaan eliminoida osa asiakkaan ja palvelimen välisestä vuorovaikutuksesta (Fielding ja Taylor 2000). Välimuistin tallennuksen avulla voidaan nopeuttaa liikennettä, sillä ilman tallennusta jokainen kutsu joutuisi kiertämään takaisin alkuperäiselle palvelimille eri reittejä pitkin, mikä kuormittaisi internetin palvelukykyä huomattavasti (Davis 2012).

3.5 Moniosainen järjestelmä (Layered System)

Moniosaisen järjestelmän tarkoitus on, että sovellukset pystyvät mukautumaan internetin vaatimiin vaatimuksiin (Fielding ja Taylor 2000). Lisäksi sen tarkoituksena on, että komponentit eivät voi nähdä kerrosten edelle, eli ne eivät tiedä ovatko ne tekemisessä loppu- vai alku-palvelimen kanssa (Fielding ja Taylor 2000). Kerroksilla tarkoitetaan arkkitehtuurin jakamista pienenpiin kerroksiin, jossa jokaisella kerroksella on omat funktionaalisuudet.

3.6 Koodaus vaadittaessa (Code on demand)

Koodaus vaadittaessa on ainoa vapaaehtoinen rajoitin REST-arkkitehtuurityylissä. Fielding ja Taylor (2000) määrittelevät, että koodaus vaadittaessa mahdollistaa koodien suorittamisen asiakkaan puolella, jos sille on tarvetta. Koodin suorittaminen voi tapahtua esimerkiksi HTML-dokumentin avulla, jossa HTML sisältää suorittavan skriptin, jonka asiakas hakee palvelimelta.

4 Määrittelyn ongelmat

Tässä luvussa tarkastellaan, miksi REST-arkkitehtuurin termin määrittelyssä esiintyy paljon eriäviä mielipiteitä ja mistä nämä eroavaisuudet johtuvat. Lisäksi tarkastellaan Richardsonin kypsyysmallia, jonka tarkoituksena on auttaa määrittelemään, miten hyvin sovellus toteuttaa REST-arkkitehtuurityylin asettamat periaatteet ja miksi käytännön toteutuksessa esiintyy yhä paljon ongelmia.

4.1 REST-arkkitehtuurityylin määrittely

REST-arkkitehtuurityylin määrittelyyn ei ole tiedeyhteisössäkään täysin selkeää linjaa. Määritelmän Wikipedia artikkelilla on yli 4000 editointia, mikä kuvastaa termin muuttuvan ja tarkentuvan jatkuvasti. R. T. Fielding ym. (2017) toteavatkin REST-arkkitehtuurityylin muistuttavan matemaattista kaavaa, joka tarkentuu ajan kuluessa yhä tarkemmaksi määritelmäksi.

Termin määrittelyn ongelmaksi muodostuu REST-arkkitehtuurityylin määrittely alkuperäisessä lähteessä, koska se ei tarjoa oikeaa tapaa toteuttaa RESTful-tyyppisiä palveluita (R. T. Fielding ym. 2017). Fielding ja Taylor (2000) toteavatkin tutkimuksessaan, että he eivät ota kantaa itse toteutukseen tai protokollaan.

Koska REST-arkkitehtuurityyli ei tarjoa tarkkoja käytännön toteuttamistapoja, voi tämä aiheuttaa ongelmia. Nämä ongelmat voidaan havainnollistaa HTTP-metodien avulla. HTTP-metodeilla tarkoitetaan HTTP-palvelupyynnön tunnistetta, joka kertoo resurssille suoritettavan metodin (R. Fielding ym. 1999, s. 34). Davis (2012, s.4) esittää taulukon, johon on koottu yhtenäisen käyttöliittymän käyttämät HTTP-metodit. Tässä taulukossa esitetään, että PUT-metodi on vastuussa resurssin päivityksestä. Kuitenkaan mikään ei estäisi käyttämästä resurssin päivitykseen POST-metodia, sillä Fielding ei tekstissään määrittele, mitä HTTP-metodeja luodussa palvelussa tulee käyttää. POST-PUT-metodien erot eivät ole välttämättä kaikille kehittäjille selviä, sillä google-haulla ”RESTful POST vs PUT” saadaan jopa yli 605 miljardia osumaa.

Tutkimalla alan tutkimuksia huomataan, että ei ole olemassa täysin selkeää rajaa, mitkä ovat

REST-arkkitehtuurityylin perusperiaatteet. R. T. Fielding ym. (2017) toteavat, että REST-arkkitehtuurityylin yhdenmukainen rajapinta voidaan jakaa viiteen rajoittimeen, kun taas (Davis 2012) toteaa rajapinnan koostuvan neljästä rajoittimesta.

Näkemyserot REST-arkkitehtuurityylin perusperiaatteista voivat johtuvat ajallisista syistä. Edellisessä luvussa esiteltiin kuusi rajoitinta, jotka löytyvät alkuperäistä lähteestä ”Architectural styles and the design of network-based software architectures”. Näiden kuuden rajoittimien tärkeyttä on korostettu uudessa määrittelyssä 2007, jolloin niistä tehtiin REST-arkkitehtuurityylin kuusi tärkeintä kohtaa (R. T. Fielding ym. 2017).

Uuden määrittelyn tarkoitus oli ratkaista RESTful-tyyppisten sovellusten kehittämisen vaikeutta, jolloin nämä kuusi muuttujaa määritettiin muodostamaan REST-arkkitehtuurityylin kokonaiskuva (R. T. Fielding ym. 2017). Vuonna 2008 Roy Fielding julkaisi blogitekstin, jossa huomautetaan, kuinka monet kehittäjät kutsuvat omia web-rajapintojaan REST-arkkitehtuurityylin täyttäviksi, vaikka ne eivät oikeasti täytä REST-arkkitehtuurityylin määrityksiä (Fielding 2008). Määrittelyn tarkennus ei siis lisännyt REST-arkkitehtuurityylin selkeyttä, sillä esimerkiksi Davis (2012) toteaa, että REST-arkkitehtuurityylin rajoittimien määrässä esiintyy erimielisyyksiä, ja monet tutkimukset ovat yksimielisiä ainoastaan neljästä rajoittimesta.

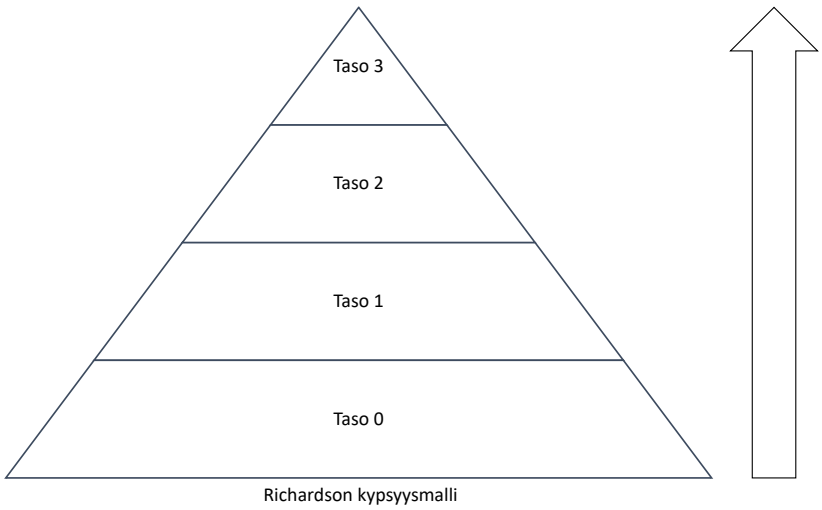
Davis (2012) esittää, että REST-arkkitehtuurityylin tärkeys on kasvanut varsinkin pilviteknologioiden yleistyessä, sillä monet kehittäjät eivät siltikään ymmärrä REST-arkkitehtuurityylin konseptia. Petrie (2016, s.26) kuvaileekin REST-arkkitehtuurityylin muistuttavan kulttia, jossa on löyhästi sovittuja sääntöjä, joiden tulee toteutua.

4.2 Richardsonin kypsyysmalli

Koska REST-arkkitehtuurityylin määrittely voi olla käytännössä vaikeaa, ja sen toteuttaminen täydellisesti käytännössä voi aiheuttaa haasteita, voidaan apuna käyttää Leonard Richardsonin kypsyysmallia (Maturity model). Tästä mallista käytetään myös nimeä RMM (Richardson Maturity Model). RMM-mallin avulla voidaan tutkia luotuja www-sovelluspalveluita ja luokitella, miten hyvin ne toteuttavat REST-arkkitehtuurityylin asettamat rajoitukset.

Mallin avulla voidaan tarkastella luotua www-sovelluspalvelua neljän eri tason kautta ja päätellä, kuinka hyvin se pystyy toteuttamaan REST-arkkitehtuurityylin ideaa (Salvadori ja Siqueira 2015). Kuviossa 1 on esitetty, miten RMM-malli esitetään kaaviona.

Mallin tarkoitus ei ole määritellä REST-arkkitehtuurityylin rajoituksia tai tarjota mekaanista toteutustapaa sen muodostamiseen, vaan sen avulla selvennetään, miten REST-arkkitehtuurityyliä voidaan hahmottaa paremmin (Salvadori ja Siqueira 2015). RMM-malli ei ole ainoa kypsyysmalli, jota voidaan käyttää www-sovelluspalveluiden arvioimiseen. Salvadori ja Siqueira (2015) esittävät, että arvioimiseen voidaan käyttää myös muita kypsyysmalleja, mutta tässä tutkielmassa pysyttelen vain Leonard Richardsonin mallissa, sillä se on profiloitunut juuri REST-arkkitehtuurityylisten www-sovelluspalveluiden arvioimiseen.



Kuvio 1. Esitys kypsyysmallista kaaviona

Taso 0

Taso 0 luokitellaan tasoksi, jossa ei olemassa mitään REST-arkkitehtuurityylin ominaisuuksia, vaan palvelussa käytetään HTTP:tä datan välittämiseen (Salvadori ja Siqueira 2015). Lisäksi tason 0 mukaisissa palveluissa on olemassa vain yksi resurssin paikannin ja yksi HTTP-metodi, jota voidaan käyttää.

Taso 1

Taso 1 esittelee resurssien käytön, missä jokainen resurssi on identifioituvissa (Salvadori ja

Siqueira 2015). Tämän tasoisissa sovelluksissa voi olla monia resurssien paikantajia, mutta edelleen vain yksi HTTP-metodi.

Taso 2

Tasossa 2 vaaditaan HTTP-metodien toteutuvan oikealla tavalla ja kutsuihin vastataan oikealla statuskoodilla (Salvadori ja Siqueira 2015). Lisäksi sovellus voi käyttää kaikkia HTTP-metodeja.

Taso 3

Viimeisin taso vaatii, että on olemassa HATEOS-implementaatio. Palvelulla on olemassa lista eri linkeistä, mitä palvelu sisältää. Lisäksi se pystyy tarjoamaan asiakkaalle tiedon, miten eri resursseja voidaan lähestyä (Salvadori ja Siqueira 2015).

4.3 Kritiikkiä RMM-mallista

RMM-mallin hyötyä on alettu kritisoida lähiaikoina, sillä sen tulkinnassa on ollut epäselvyyksiä. RMM-malli on ollut esillä internetin blogeissa, ja osa tieteellisistä artikkeleista viittaakin Fowler (2010) kirjoittamaan blogiin puhuttaessa RMM-mallista. Kuitenkin RMM-malli on alkuperäisesti lähtöisin Richardson (2008) blogista, ja Martin Fowlerin kirjoittama blogi on vain yleistys alkuperäisestä mallista. Martin Fowlerin kirjoittamassa blogissa esimerkiksi jaotellaan sovellukset, jotka eivät toteuta yhtään tasoa ei kiinnostava XML-toteutukseksi (Swamp of Plain Old XML) ja yli kolmannen tason sovellukset kunniallisesti REST-sovelluksiksi (Glory of REST) (Fowler 2010, kirjoittajan suomennus). Kuitenkaan Richardson ei itse alkuperäisessä kirjoituksessä käytä näin kantaa ottavia termejä.

RMM-mallista on olemassa myös hyvin vähän luotettavaa lähdekirjallisuutta, ja mallin alkuperästä ei ole olemassa muuta lähdettä kuin Richardson (2008) kirjoittama blogi. Steiner ja Algermissen (2011) myöskin toteavat, että RMM-mallin käyttäminen ei ole mielekästä, sillä se olettaa olevan olemassa tietyt kiinteät vaatimukset, miten REST-arkkitehtuurityylin rajoitukset voidaan jakaa toteutettaviin arvoihin.

4.4 Vaikeus käytännön toteutuksessa

Koschel ym. (2019) suorittivat tutkimuksen, jossa tutkittiin 10 erilaista web-rajapintaa. Tutkimuksessa huomattiin, että kuusi kymmenestä eri rajapinnoista eivät toteuttaneet edes RMM-mallin tasoa 3. Tutkimuksessa todettiin myös, että vaikka yli puolet sovelluksista eivät toteuttaneet RMM-mallin tasoa 3, eivät tason 2 sovellukset olleet huonompia sovelluksia kuin tason 3 sovellukset (Koschel ym. 2019).

Samantyyppisiin tuloksiin päätyivät myös Renzel, Schlebusch ja Klamma (2012, s. 354-367), mutta tutkimuksessa käytettiin 17 kriteerin kohtaa arvioimaan sovelluksia. Nämä 17 kriteeriä perustuivat kirjaan "RESTful services described by Richardson and Ruby". Tutkimuksessa huomattiin, että on olemassa suuria eroja, miten tutkitut palvelut ovat omaksuneet REST-arkkitehtuurityylin toteutuksen.

Renzel, Schlebusch ja Klamma (2012, s. 354-367) esittävät, että kehittäjien on vaikea toteuttaa kaikkia REST-arkkitehtuurityylin suuntauksia, sillä ne aiheuttavat paljon ylimääräistä vaivaa. Monesti kehittäjät myös kopioivat isompien yritysten RESTful-palveluiden ratkaisuja, ja näin ei-täydelliset RESTful-palvelut tulevat enemmän esille aiheuttaen termin hämärtymistä (Renzel, Schlebusch ja Klamma 2012, s. 354-367). Renzel, Schlebusch ja Klamma (2012, s. 354-367) toteavatkin, että vastuu REST-arkkitehtuurityylin selventämisessä on tutkijoiden harteilla, jotta termi olisi helpommin ymmärrettävissä.

Molemmissa tutkimuksissa myös huomautettiin, miten RESTful on muuttunut yleistermiksi (buzzword) alalla (Renzel, Schlebusch ja Klamma 2012; Koschel ym. 2019). Myöskin R. T. Fielding ym. (2017) toteaa, miten termiä jatkuvasti käytetään väärin. RESTful-palvelu voi populäärisoituneesti tarkoittaa www-sovelluspalvelua, joka toteuttaa HTTP:tä ja on saavutettavissa URL:n kautta (R. T. Fielding ym. 2017; Renzel, Schlebusch ja Klamma 2012).

5 Yhteenveto

Tässä tutkielmassa tutkittiin REST-arkkitehtuurityyliä, joka on yleisesti käytetty tapa toteuttaa www-sovelluspalveluita. RESTful-palvelut ovat nousseet merkittäväksi tavaksi toteuttaa www-sovelluspalveluita internet-ympäristössä ja tarjonneet mallin minkätyyppisiä näiden palveluiden tulisi olla. Lisäksi RESTful-palvelut ovat myös yksinkertaistaneet www-sovelluspalveluiden toteuttamista, mikä on edesauttanut uusien kehittäjien työmäärää (Fensel ym. 2011). Vaikka REST-arkkitehtuurityyli on vielä suosittu tapa toteuttaa www-sovelluspalveluita, on vaikea todeta, onko näin vielä 10 vuoden päästä. Uudet kehittämistavat esimerkiksi GraphQL on noussut haastamaan RESTful-palveluiden asemaa tarjoamalla vielä helpompia tapoja toteuttaa www-sovelluspalveluita.

REST-arkkitehtuurityylin kuusi rajoitinta muodostavat tyylin perusteet ja säännöt, mitä RESTful-palveluiden tulee noudattaa. REST-arkkitehtuurityylin määrittelyssä on kuitenkin esiintynyt ongelmia, sillä eri tieteellisillä lähteillä on ollut eri näkemys REST-arkkitehtuurin rajoittimista. Alkuperäisessä lähteessä REST-arkkitehtuurityyli on määrittely vain arkkitehtuurisena tyylinä, eikä käytännön toteutukseen ole otettu kantaa, joten arkkitehtuurityylin tulkitseminen on täysin kehittäjien vastuulla. Tutkielmassa huomataan, että rajoittimien seuraaminen ei ole ollut täysin selvää ja kehittäjät ovat jättäneet rajoittimia pois sovelluksistaan ja silti kutsuneet sovelluksia RESTful-palveluiksi. Rajoittimien poisjättämisen syynä on ollut niiden aiheuttama työmäärä kehittäjille sekä, että kehittäjät eivät ole täysin sisäistäneet termiä (Renzel, Schlebusch ja Klamma 2012; Davis 2012, s. 354-367).

Tutkielmassa myös huomataan, että on ollut vaikeuksia todeta, mitkä sovellukset ovat oikeasti RESTful-palveluita. Leonard Richardson kypsyysmallia voidaan käyttää apuna, mutta mallin ongelmana on, että se ei tuota oikeaa vastausta onko sovellus RESTful-palvelu, vaan pikemmin auttaa hahmottamaan toteutusta. Malli on myös saanut osaksi kritiikkiä, sillä se olettaa REST-arkkitehtuurin olevan jaettavissa tiettyihin toteutettaviin arvoihin, joiden avulla voidaan tehdä määritelmä onko sovellus RESTful (Steiner ja Algermissen 2011).

Koska termi on myös popularisoitunut ja muuttunut muotisanaksi, on sen määritelmä hämärtynyt ja näin aiheuttanut ongelmia kehittäjille (Renzel, Schlebusch ja Klamma 2012; R. T.

Fielding ym. 2017, s. 354-367). Usein uudet kehittäjät saattavat törmätä sanaan blogeissa tai ohjelmointi-sivustoilla, jossa termin tulkinnan on tehnyt tekstin kirjoittaja. Lisäksi kuten Renzel, Schlebusch ja Klamma (2012, s. 354-367) toteavat, monet kehittäjät kopioivat suosittujen RESTful-palveluiden ratkaisuja, vaikka nämä palvelut eivät olisikaan oikeasti RESTful-palveluita. Tämä aiheuttaa REST-arkkitehtuurityylin termin hämärtymistä.

Lähteet

Berners-Lee, Tim. 1989. *Information Management: A Proposal.* Saatavilla WWW-muodossa, https://www.w3.org/History/1989/proposal.html. Viitattu 15.11.2019.

Connolly, Dan. 2000. *A Little History of the World Wide Web.* Saatavilla WWW-muodossa, https://www.w3.org/History.html. Viitattu 15.11.2019.

Davis, Cornelia. 2012. "What if the Web Were Not RESTful?" Teoksessa *Proceedings of the Third International Workshop on RESTful Design,* 3–10. WS-REST '12. Lyon, France: ACM. ISBN: 978-1-4503-1190-8. http://doi.acm.org/10.1145/2307819.2307823.

Fensel, Dieter, Federico Michele Facca, Elena Simperl ja Ioan Toma. 2011. *Semantic web services.* Springer Science & Business Media.

Fielding, Roy T., Richard N. Taylor, Justin R. Erenkrantz, Michael M. Gorlick, Jim Whitehead, Rohit Khare ja Peyman Oreizy. 2017. "Reflections on the REST Architectural Style and "Principled Design of the Modern Web Architecture"(Impact Paper Award)". Teoksessa *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering,* 4–14. ESEC/FSE 2017. Paderborn, Germany: ACM. ISBN: 978-1-4503-5105-8. http://doi.acm.org.ezproxy.jyu.fi/10.1145/3106237.3121282.

Fielding, Roy T, ja Richard N Taylor. 2000. *Architectural styles and the design of network-based software architectures.* Nide 7. University of California, Irvine Doctoral dissertation.

Fielding, Roy Thomas. 2008. *REST APIs must be hypertext-driven.* Saatavilla WWW-muodossa, https://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven. Viitattu 20.11.2019.

Fielding, Roy, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach ja Tim Berners-Lee. 1999. *Hypertext transfer protocol–HTTP/1.1.* Saatavilla WWW-muodossa, http://www.hjp.at/doc/rfc/rfc2068.html. Viitattu 18.12.2019.

Fowler, Martin. 2010. *Richardson Maturity Model.* Saatavilla WWW-muodossa, https://martinfowler.com/articles/richardsonMaturityModel.html. Viitattu 20.11.2019.

Fowler, Susan, ja Victor Stanwick. 2004. *Web application design handbook: Best practices for web-based software.* Morgan Kaufmann.

Hartina, D. A., A. Lawi ja B. L. E. Panggabean. 2018. "Performance Analysis of GraphQL and RESTful in SIM LP2M of the Hasanuddin University". Teoksessa *2018 2nd East Indonesia Conference on Computer and Information Technology (EIConCIT),* 237–240. Marraskuu. doi:10.1109/EIConCIT.2018.8878524.

Koschel, A., M. Blankschyn, K. Schulze, D. Schöner, I. Astrova ja I. Astrov. 2019. "RESTfulness of APIs in the Wild". Teoksessa *2019 IEEE World Congress on Services (SERVICES),* 2642-939X:382–383. Heinäkuu.

Leiner, Barry M., Vinton G. Cerf, David D. Clark, Robert E. Kahn, Leonard Kleinrock, Daniel C. Lynch, Jon Postel, Larry G. Roberts ja Stephen Wolff. 2009. "A Brief History of the Internet". *SIGCOMM Comput. Commun. Rev.* (New York, NY, USA) 39, numero 5 (lokakuu): 22–31. ISSN: 0146-4833. http://doi.acm.org.ezproxy.jyu.fi/10.1145/1629607.1629613.

Netcraft. 2019. *Netcraft.* Saatavilla WWW-muodossa, https://news.netcraft.com/archives/category/web-server-survey/. Viitattu 11.11.2019.

Petrie, Charles J. kirjoittaja. 2016. *Web Service Composition.* 82. Cham: Springer International Publishing. http://dx.doi.org/10.1007/978-3-319-32833-1.

Renzel, Dominik, Patrick Schlebusch ja Ralf Klamma. 2012. "Today's top "RESTful" services and why they are not RESTful". Teoksessa *International Conference on Web Information Systems Engineering,* 354–367. Springer.

Richardson, Leonard. 2008. *Act Three: The Maturity Heuristic.* Saatavilla WWW-muodossa, https://www.crummy.com/writing/speaking/2008-QCon/act3.html. Viitattu 20.11.2019.

Salvadori, I., ja F. Siqueira. 2015. “A Maturity Model for Semantic RESTful Web APIs”. Teoksessa *2015 IEEE International Conference on Web Services,* 703–710. Kesäkuu.

Schreier, Silvia. 2011. “Modeling restful applications”. Teoksessa *Proceedings of the second international workshop on restful design,* 15–21. ACM.

Steiner, Thomas, ja Jan Algermissen. 2011. “Fulfilling the hypermedia constraint via HTTP OPTIONS, the HTTP vocabulary in RDF, and link headers”. Teoksessa *Proceedings of the second international workshop on RESTful design,* 11–14. ACM.