

Sami Kankaanpää

# **REST-arkkitehtuurityylin käyttö web-rajapinnoissa**

Opinnäytetyö

Kevät 2016

SeAMK Tekniikka

Tietotekniikan tutkinto-ohjelma



SEINÄJOEN AMMATTIKORKEAKOULU  
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

SEINÄJOEN AMMATTIKORKEAKOULU

## Opinnäytetyön tiivistelmä

Koulutusyksikkö: Tekniikan yksikkö

Tutkinto-ohjelma: Tietotekniikka

Suuntautumisvaihtoehto: Ohjelmistotekniikka

Tekijä: Sami Kankaanpää

Työn nimi: REST-arkkitehtuurityylin käyttö web-rajapinnoissa

Ohjaaja: Petteri Mäkelä

Vuosi: 2016

Sivumäärä: 41

Liitteiden lukumäärä: 1

---

REST-rajapinnat ovat yleistyneet viime vuosina huomattavasti, ja niistä on tullut kaikista keskeisin web-rajapintojen toteutustapa. Kuitenkin vain harva toteutus on nimensä veroinen, sillä REST-arkkitehtuurityyliä noudatetaan edelleen vain osittain.

Opinnäytetyön tavoitteena oli antaa yleiskuva REST-arkkitehtuurityyliin perustuvista web-rajapinnoista. Työssä keskityttiin pääasiassa niiden toimintaan ja luokitteluun. Luokittelun yhteydessä selvitettiin lisäksi, millainen on tyypillinen REST-rajapinta ja mitkä ovat REST-arkkitehtuurityylin kiistellyn HATEOAS-rajoitteen vaikutukset. Työhön sisältyi myös Flask-sovelluskehityksen ja sen Flask-RESTful-lisäosan avulla luotu esimerkkisovellus.

Työstä muodostui tietopaketti, jota voidaan hyödyntää aiheeseen perehdyttävänä taustamateriaalina. Työstä saattaa olla hyötyä myös muissa asiayhteyksissä, sillä työssä käytiin läpi asioita, jotka liittyvät moniin verkkosovelluksiin ja -palveluihin.

Avainsanat: web-rajapinta, REST, HTTP, HATEOAS

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

## **Thesis abstract**

Faculty: School of Technology

Degree programme: Information Technology

Specialisation: Software Engineering

Author: Sami Kankaanpää

Title of thesis: Using the REST architectural style in web APIs

Supervisor: Petteri Mäkelä

Year: 2016

Number of pages: 41

Number of appendices: 1

---

REST APIs have become more common in the recent years and these days they are the most popular way of implementing web APIs. However, only a few implementations are worthy of the name, because the REST architectural style continues to be applied only partially.

The goal of this thesis was to provide an overview of the REST architectural style based web APIs. It focused mainly on the operation and classification. While focused on classification, it also examined the typical REST APIs and the effects of HATEOAS, which is a controversial constraint of the REST architectural style. The thesis also included an example application that was created using the Flask micro framework and its Flask-RESTful extension.

The thesis became an informational package, which can be used as background material for the topic. It may also be useful in other contexts, because it covers things that are related to many web applications and services.

Keywords: web API, REST, HTTP, HATEOAS

## SISÄLTÖ

Opinnäytetyön tiivistelmä.....	2
Thesis abstract.....	3
SISÄLTÖ.....	4
Kuvio- ja taulukkoluetelo.....	6
Käytetyt termit ja lyhenteet .....	8
1 JOHDANTO .....	10
1.1 Työn tavoite .....	10
1.2 Työn rakenne .....	10
2 WEB-RAJAPINNAT .....	12
2.1 SOAP .....	12
2.2 REST .....	12
2.3 Nykytila .....	13
3 TOIMINTA.....	14
3.1 Yleistä .....	14
3.2 REST .....	15
3.2.1 Asiakas-palvelin .....	16
3.2.2 Tilattomuus .....	16
3.2.3 Välimuisti.....	17
3.2.4 Yhdenmukainen rajapinta .....	17
3.2.5 Kerroksittainen järjestelmä.....	18
3.2.6 Ladattava koodi.....	18
3.3 HTTP.....	19
3.3.1 Pyyntö .....	20
3.3.2 Vastaus .....	21
3.3.3 Metodit .....	22
3.3.4 Tilakoodit.....	22
3.3.5 Otsikkotiedot .....	23
3.4 URI.....	23
3.4.1 Syntaksi .....	24
3.4.2 Resurssi.....	24

3.4.3 Representaatio.....	25
4 LUOKITTELU.....	26
4.1 Richardsonin kypsyyssmalli.....	26
4.1.1 Taso 0.....	27
4.1.2 Taso 1.....	28
4.1.3 Taso 2.....	29
4.1.4 Taso 3.....	30
4.2 Tyypillinen REST-rajapinta.....	31
4.3 HATEOAS.....	31
4.3.1 Hyödyt.....	32
4.3.2 Haitat.....	32
5 ESIMERKKISOVELLUS.....	33
5.1 Flask.....	33
5.2 Flask-RESTful.....	33
5.3 Toteutus.....	34
6 TULOKSET JA YHTEENVETO.....	38
LÄHTEET.....	39
LIITTEET.....	41

## Kuvio- ja taulukkoluettelo

Kuvio 1. REST-rajapinnan sijainti.....	14
Kuvio 2. HTTP-pyyntö ja -vastaus.....	19
Kuvio 3. HTTP-pyyntöjen rakenne .....	20
Kuvio 4. Esimerkki HTTP-pyyntöstä.....	20
Kuvio 5. HTTP-vastauksen rakenne .....	21
Kuvio 6. Esimerkki HTTP-vastauksesta.....	21
Kuvio 7. URI-tunnuksen syntaksi .....	24
Kuvio 8. Richardsonin kypsyyssmalli.....	26
Kuvio 9. Tason 0 mukainen pyyntö.....	27
Kuvio 10. Tason 0 mukainen vastaus.....	27
Kuvio 11. Tason 1 mukainen pyyntö.....	28
Kuvio 12. Tason 1 mukainen vastaus.....	28
Kuvio 13. Tason 2 mukainen pyyntö.....	29
Kuvio 14. Tason 2 mukainen vastaus.....	29
Kuvio 15. Tason 3 mukainen pyyntö.....	30
Kuvio 16. Tason 3 mukainen vastaus.....	30
Kuvio 17. REST-rajapinnan lähdekoodi, osa 1/3.....	35
Kuvio 18. REST-rajapinnan lähdekoodi, osa 2/3.....	36
Kuvio 19. REST-rajapinnan lähdekoodi, osa 3/3.....	37

Taulukko 1. Yleisimmin käytetyt HTTP-metodit .....	22
Taulukko 2. HTTP-protokollan tilakoodiryhmät .....	22
Taulukko 3. Yleisesti käytettyjä otsikkotietoja .....	23
Taulukko 4. REST-rajapinnan toimintaperiaate.....	34

## Käytetyt termit ja lyhenteet

<b>CRUD</b>	Create, Read, Update ja Delete ovat neljä perustoimintoa tietokantojen manipulointiin.
<b>HATEOAS</b>	Hypermedia as the Engine of Application State on yksi REST-tyylin määrittelemistä rajoitteista.
<b>HTML</b>	Hypertext Markup Language on kuvauskieli, jota käytetään yleensä verkkosivujen luomiseen.
<b>HTTP</b>	Hypertext Transfer Protocol on tiedonsiirtoprotokolla, jota käytetään hypermediajärjestelmissä.
<b>Hypermedia</b>	Hypermedia tarkoittaa toisiinsa linkitettyä informaatiota, kuten tekstiä, kuvia tai videota.
<b>JSON</b>	JavaScript Object Notation on formaatti, jota käytetään tiedonsiirrossa ja tallennuksessa.
<b>JSON-LD</b>	JavaScript Object Notation for Linked Data on linkkejä tukeva formaatti, joka perustuu pääosin JSON-formaattiin.
<b>Representaatio</b>	Representaatio on informaatiota, joka kuvaa resurssin mennyttä, nykyistä tai haluttua tilaa.
<b>REST</b>	Representational State Transfer on arkkitehtuurityyli, jota käytetään hypermediajärjestelmissä.
<b>REST-rajapinta</b>	REST-rajapinta on web-rajapinta, joka noudattaa REST-tyylin määrittelemiä rajoitteita.
<b>Resurssi</b>	Resurssi voi olla melkein mikä tahansa asia, joka voidaan identifioida, kuten kuva tai dokumentti.
<b>RPC</b>	Remote Procedure Call on keino, jonka avulla asiakas voi käyttää palvelimen palveluita.



<b>Skripti</b>	Skripti on yleensä lyhyt tietokoneohjelma, joka on kirjoitettu asianmukaisella ohjelmointikielellä.
<b>SOAP</b>	Simple Object Access Protocol on viestiprotokolla, joka mahdollistaa RPC-kutsut.
<b>URI</b>	Uniform Resource Identifier on kompakti merkkijono, jota käytetään resurssien tunnistamiseen.
<b>Web tai WWW</b>	World Wide Web on hajautettu hypermediajärjestelmä ja yksi Internetin palvelumuodoista.
<b>Web-rajapinta</b>	Web-rajapinta on World Wide Web -pohjainen rajapinta, jonka avulla ohjelmistot voivat kommunikoida keskenään.
<b>XML</b>	Extensible Markup Language on formaatti, jota käytetään tiedonsiirrossa ja tallennuksessa.

# 1 JOHDANTO

REST-arkkitehtuurityyliin perustuvat web-rajapinnat ovat yleistyneet huomattavasti. Ne ovat käytännössä syrjäyttäneet vielä muutama vuosi sitten yleisesti käytetyt SOAP-pohjaiset web-rajapinnat. Nimestään huolimatta useimmat toteutukset eivät kuitenkaan ole varsinaisia REST-rajapintoja, koska ne eivät ole REST-tyylin mukaisia. REST on ollut eräänlainen muotisana, jonka avulla on markkinoitu web-rajapintoja, usein välittämättä siitä, kuinka tarkasti kyseistä arkkitehtuurityyliä on noudatettu. Ajan myötä tilanne on kuitenkin muuttunut ja REST-tyyliä on alettu noudattaa aiempaa tarkemmin.

## 1.1 Työn tavoite

Työn tavoitteena on antaa yleiskuva REST-arkkitehtuurityyliin perustuvista web-rajapinnoista. Keskeisimmät käsiteltävät asiat ovat niiden toiminta ja luokittelu. Luokittelun yhteydessä selvitetään lisäksi, millainen on tyypillinen REST-rajapinta ja mitkä ovat REST-arkkitehtuurityylin kiistellyn HATEOAS-rajoitteen vaikutukset. Luettuaan työn, lukijan tulisi ymmärtää, mikä REST-rajapinta on, miten ne toimivat ja kuinka toteutuksia voidaan luokitella Richardsonin kypsyyssmallia apuna käyttäen.

## 1.2 Työn rakenne

Työn toisessa luvussa käsitellään lyhyesti web-rajapintojen historiaa ja nykytilaa. Luvussa käydään läpi kaksi yleisimmin käytettyä web-rajapintojen toteutustapaa: REST ja SOAP.

Kolmannessa luvussa perehdytään REST-rajapintojen toimintaan. Sen yhteydessä käydään läpi seuraavat kolme asiaa: REST-arkkitehtuurityyli, HTTP-protokolla ja URI-tunnus.

Neljännessä luvussa esitellään Richardsonin kypsyyssmalli, jonka avulla voidaan luokitella REST-rajapintoja. Lisäksi selvitetään, millaisia nykyiset REST-rajapinnat ovat ja käydään läpi REST-arkkitehtuurityylin HATEOAS-rajoitteen hyödyt ja haitat.

Viidennessä luvussa luodaan esimerkkipohjainen sovellus käyttäen Flask-sovelluskehystä ja sen Flask-RESTful-lisäosaa. Luotava REST-rajapinta on Richardsonin kypsyyssmallin tason 2 mukainen.

Kuudes luku pitää sisällään työn tulokset ja yhteenvedon.

## 2 WEB-RAJAPINNAT

Web-rajapinnat ovat ehtineet olla yleisessä käytössä jo yli vuosikymmenen ajan. REST-rajapinnat eivät kuitenkaan ole olleet ainoa web-rajapintojen toteutustapa. Tässä luvussa käydään lyhyesti läpi web-rajapintojen historia ja niiden nykytilanne.

### 2.1 SOAP

Ennen kuin REST-rajapinnat olivat ehtineet yleistyä, tyypillinen web-rajapinta oli SOAP-pohjainen. SOAP on Microsoftin kehittämä standardisoitu viestiprotokolla. Sen yhteydessä voidaan käyttää ainoastaan XML-formaattiin perustuvia viestejä, jotka voivat olla tietyissä tapauksissa monimutkaisia ja suhteellisen hitaita käsitellä. SOAP on REST-tyylin tapaan täysin riippumaton käytetystä tiedonsiirtoprotokollasta. Se on suunniteltu viestiprotokollaksi, joka kattaa monta erilaista käyttötilannetta. SOAP-pohjaiset web-rajapinnat eivät kuitenkaan ole yhtä helposti lähestyttäviä kuin REST-rajapinnat. Tämä on tärkeää esimerkiksi avoimien rajapintojen yhteydessä. (ProgrammableWeb 2016.)

### 2.2 REST

REST on arkkitehtuurityyli, joka tulee sanoista Representational State Transfer. REST ei liity ainoastaan REST-rajapintoihin, vaan se on yleinen arkkitehtuurityyli, joka on alun perin suunniteltu käytettäväksi hajautetuissa hypermediajärjestelmissä. Paras esimerkki kyseiseen arkkitehtuurityyliin perustuvasta järjestelmästä on WWW (World Wide Web). REST esiteltiin Roy Fieldingin väitöskirjassa vuonna 2000. (Fielding 2000, 76–106.)

REST-rajapintojen ongelmana on ollut alusta asti kyseisen termin väärinkäyttö. Käytännössä mitä tahansa web-rajapintaa, joka ei ole SOAP-pohjainen on kutsuttu REST-rajapinnaksi, usein välittämättä siitä, kuinka tarkasti REST-tyyliä on noudatettu. Viime vuosina REST-tyyliä on alettu noudattaa tarkemmin, mutta vain harvoin täysin. (Richardson, Amundsen & Ruby 2013, XV–XVIII.)

## 2.3 Nykytila

Nykyään suurin osa uusista web-rajapinnoista perustuu REST-arkkitehtuurityyliin. Ne ovat syrjäyttäneet aiemmin yleisesti käytetyt SOAP-pohjaiset web-rajapinnat. REST-rajapinnat ovat myös lisänneet avoimien rajapintojen suosiota merkittävästi. Esimerkiksi ProgrammableWeb-verkkosivuston hakemistossa listattujen avoimien rajapintojen lukumäärä on kasvanut n. 2 000:sta n. 14 000:een vuosina 2010–2015. (ProgrammableWeb 2016.)

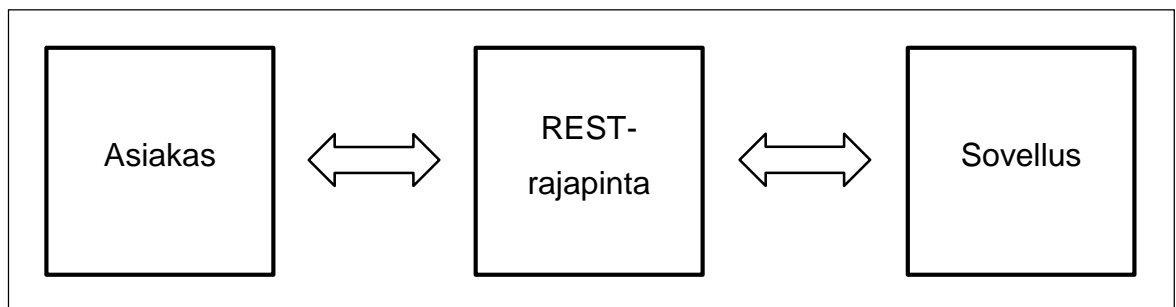
Aikaisemmin web-rajapinnat, kuten REST-rajapinnat kehitettiin usein jälkikäteen, vasta sen jälkeen kun kyseiseen rajapintaan liittyvä palvelu oli kehitetty valmiiksi. Viime vuosina on alettu käyttämään yhä enemmän niin kutsuttua API-first-mallia, jossa kehitetään ensin web-rajapinta ja vasta sen jälkeen kaikki muu tarpeellinen. API-first-mallissa keskitytään ensisijaisesti palvelun käsittelemään dataan ja sen hyödynnettäväksi saattamiseen. (ProgrammableWeb 2016.)

### 3 TOIMINTA

REST-rajapintojen toimintaan liittyy lukuisia tekniikoita, protokollia ja muita asioita. Keskeisimmät niistä ovat: REST-arkkitehtuurityyli, HTTP-protokolla ja URI-tunnus. Tässä luvussa käydään tarkemmin läpi kyseiset kolme keskeisintä asiaa ja niiden yhteistoiminta.

#### 3.1 Yleistä

REST-rajapinnan tehtävänä on saattaa sovelluksen toiminnallisuudet asiakkaiden käytettäväksi (kuvio 1). Sovellus ei ole välttämättä riippuvainen REST-rajapinnasta, koska sitä voidaan mahdollisesti käyttää myös jonkin muun käyttöliittymän avulla. Käytännössä yksi ohjelmistokokonaisuus käsittää usein sekä REST-rajapinnan että sovelluksen. (Jansen 2011.)



Kuvio 1. REST-rajapinnan sijainti.

Asiakas, kuten verkkoselain, kommunikoi REST-rajapinnan kanssa HTTP:n avulla. REST-rajapinnan ja sovelluksen välinen kommunikointi puolestaan riippuu täysin kyseisistä ohjelmistoista. Useimmiten sovellus tarvitsee toimiakseen tietokannan. Tämä tietokanta voi sijaita joko sovelluksen yhteydessä tai sovelluksesta erillään. (Jansen 2011.)

Sovelluksella on tietty tila, joka muuttuu suoritettavien operaatioiden seurauksena. REST-rajapinnan tarkoituksena on mallintaa kyseinen tila ja mahdolliset operaatiot asiakkaita varten. Käytännössä REST-rajapinta muuntaa sovelluksen datamallin mukaiset objektit eräänlaisiksi resursseiksi, jotka voidaan tunnistaa yksilöllisesti. (Jansen 2011.)

## 3.2 REST

REST (Representational State Transfer) on arkkitehtuurityyli, joka on suunniteltu käytettäväksi hajautetuissa hypermediajärjestelmissä. Se koostuu rajoitteista, jotka ohjaavat arkkitehtuuristen elementtien toimintaa. Kyseisiä rajoitteita on lainattu muista arkkitehtuurityyleistä, ja täydennetty yhdenmukaista rajapintaa käsittelevillä lisärajoitteilla. REST ei ota kantaa toteutuksen yksityiskohtiin, vaan keskittyy lähinnä perusasioihin, kuten komponenttien rooleihin ja niiden väliseen kommunikointiin. (Fielding 2000, 76–106.)

REST-arkkitehtuurityyli jakaa arkkitehtuuriset elementit kolmeen eri pääluokkaan:

- komponentit
- liittimet
- dataelementit (Fielding 2000, 86–97).

Komponenteilla tarkoitetaan ohjelmistoja, joilla on tarve kommunikoida muiden komponenttien kanssa. Liittimet mahdollistavat komponenttien kommunikoinnin. Dataelementit puolestaan ovat tietoa, jota komponentit siirtävät tai muokkaavat liittimien avulla. REST määrittelee neljä eri komponenttia, viisi liitintä ja kuusi erilaista dataelementtiä. (Fielding 2000, 86–97.)

REST-tyylin rajoitteilla pyritään parantamaan seuraavia arkkitehtuurisia ominaisuuksia:

- suorituskkyä
- skaalautuvuutta
- yksinkertaisuutta
- muunneltavuutta
- näkyvyyttä
- siirrettävyyttä
- luotettavuutta (Fielding 2000, 76–86).

Edellä mainittuja rajoitteita on yhteensä kuusi, joista yksi sisältää neljä lisärajoitetta. Jotta järjestelmää voitaisiin kutsua REST-arkkitehtuurityylin mukaiseksi, sen täytyy noudattaa kaikkia rajoitteita. (Fielding 2000, 76–86.) Kyseiset rajoitteet käydään läpi seuraavissa alaluvuissa.

### 3.2.1 Asiakas-palvelin

Ensimmäinen rajoite perustuu tietoverkoissa käytettyyn asiakas-palvelin-malliin. Rajoite edellyttää, että järjestelmä koostuu kahdesta eri komponentista: asiakkaasta ja palvelimesta. Molemmilla komponenteilla on järjestelmässä oma rajattu roolinsa. Yleensä tehtävät jaetaan siten, että asiakas huolehtii käyttöliittymästä ja palvelin huolehtii tiedon tallennuksesta. Kyseinen vastuiden jako parantaa järjestelmän skaalautuvuutta ja mahdollistaa komponenttien kehittämisen toisistaan riippumatta. (Fielding 2000, 45, 78.)

Asiakkaan ja palvelimen välinen vuorovaikutus tapahtuu tyypillisesti seuraavasti: Palvelin tarjoaa palveluita, odottaen pyyntöjä, jotka koskevat kyseisiä palveluita. Asiakas tarvitsee palvelimen tarjoamia palveluita, joten se lähettää sille pyynnön. Palvelin vastaanottaa pyynnön, ja tapauskohtaisesti joko hylkää tai suorittaa sen. Transaktion lopuksi palvelin lähettää vielä asianmukaisen vastauksen asiakkaalle. (Fielding 2000, 45.)

### 3.2.2 Tilattomuus

Seuraava rajoite täsmentää, miten asiakkaan ja palvelimen tulisi kommunikoida. Rajoite edellyttää, että kommunikointi tulee olla tilatonta. Jokaisen pyynnön täytyy tämän vuoksi sisältää kaikki informaatio, mikä on tarpeellista kyseisen pyynnön suorittamiseksi. Palvelimen tulee siis käsitellä pyynnot yksittäisinä kokonaisuuksina. Käytännössä sovelluksen tilatietojen hallinta tulee jättää yksinomaan asiakkaan vastuulle. (Fielding 2000, 78–79.)

Tilattomuus parantaa järjestelmän näkyvyyttä, koska jokaisen pyynnön todellinen tarkoitus on nähtävissä kustakin pyynnöstä. Se parantaa myös skaalautuvuutta, koska palvelimen ei tarvitse käyttää omia resursseja sovelluksen tilan tarkkailuun. Aiempaa helpompi palautuminen vikatilanteista puolestaan parantaa luotettavuutta. Tilattoman kommunikoinnin vuoksi pyyntöihin joudutaan kuitenkin sisällyttämään enemmän välttämätöntä dataa. (Fielding 2000, 79.)



### 3.2.3 Välimuisti

Edellisten rajoitteiden ongelmakohtia pyritään paikkaamaan välimuisti-rajoitteella. Rajoite edellyttää, että pyyntöjen vastauksista täytyy selvittää, voidaanko kyseinen vastaus tallentaa välimuistiin. Jos välimuistin käyttö sallitaan, asiakas voi tallentaa vastauksen ja voi täten mahdollisesti välttyä tekemästä samaa pyyntöä uudestaan. Välimuisti auttaa sellaisten pyyntöjen vähentämisessä, jotka kohdistuvat aiemmin pyydettyihin resursseihin, joiden vastaukset löytyvät yhä välimuistista, mutta eivät ole vielä vanhentuneita. (Fielding 2000, 48, 79.)

Välimuistin käyttäminen parantaa suorituskyvyn ja skaalautuvuuden lisäksi myös käyttäjäkokemusta. Se voi kuitenkin heikentää järjestelmän luotettavuutta, koska välimuistiin tallennettu vastaus saattaa toisinaan vanhentua odotettua aikaisemmin. Tällöin välimuistista haettu vastaus ei enää vastaa palvelimelta haettua vastausta. (Fielding 2000, 48, 80.)

### 3.2.4 Yhdenmukainen rajapinta

REST määrittelee neljä rajoitetta, jotka erottavat sen muista arkkitehtuurityyleistä. Rajoitteet edellyttävät, että asiakkaan ja palvelimen välisen rajapinnan tulee toimia yhdenmukaisesti, riippumatta siitä, millaisia palveluita rajapinnan kautta välitetään. Kaksi ensimmäistä rajoitetta käsittelevät resurssien tunnistamista ja manipulointia. Kolmas rajoite edellyttää itseselitteisiä viestejä, ja neljäs puolestaan hypermedian käyttöä sovelluksen tilakoneena (luku 4.3). (Fielding 2000, 81–82.)

Koska rajapinnan toiminnollisuus on erotettu sen kautta välitettävistä palveluista, kumpaakin osa-aluetta voidaan kehittää itsenäisesti, toisistaan riippumatta. Tämä parantaa myös rajapinnan yleiskäyttöisyyttä sekä komponenttien vuorovaikutusten näkyvyyttä. Kyseinen erotus voi lisäksi yksinkertaistaa järjestelmäarkkitehtuuria. Yhdenmukainen rajapinta johtaa toisaalta huomompaan tiedonsiirron tehokkuuteen, koska tieto siirretään aina samassa ennalta määritellyssä muodossa, vaikka se ei olisikaan paras ratkaisu. (Fielding 2000, 81–82.)

### 3.2.5 Kerroksittainen järjestelmä

Seuraava rajoite käsittelee järjestelmän muodostamista hierarkkisista kerroksista. Rajoite edellyttää, että kerrosten tulee tarjota palveluita hierarkiassa yläpuolella oleville kerroksille ja hyödyntää hierarkiassa alapuolella olevien kerrosten palveluita. Komponentit voivat kommunikoida vain välittömästi ylä- tai alapuolella olevien kerrosten komponenttien kanssa, koska ne eivät ole tietoisia seuraavista kerroksista. (Fielding 2000, 46, 82.)

Kerroksittaisuus helpottaa järjestelmän monimutkaisuuden hallintaa ja edistää eri komponenttien itsenäisyyttä. Sen avulla on myös mahdollista kapseloida vanhoja järjestelmän osia sekä estää uusien palveluiden käytön vanhentuneilta asiakkailta. Kerroksittaisuus voi lisäksi parantaa järjestelmän skaalautuvuutta ja tietoturvaa. Sen haittapuolena on kuitenkin tiedon prosessoinnin lisääntymisestä aiheutuva viive. (Fielding 2000, 83.)

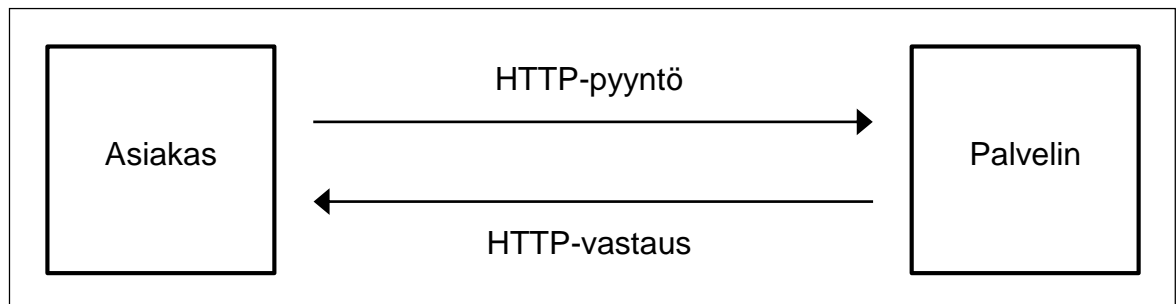
### 3.2.6 Ladattava koodi

REST-tyylin rajoitteista viimeinen, ladattava koodi (Code on Demand), on edellisistä poiketen valinnainen. Rajoite edellyttää, että asiakaskomponentin toiminnallisuutta voidaan tarvittaessa laajentaa lataamalla palvelimelta paikallisesti suoritettavaa koodia, kuten skriptejä. Rajoitteen valinnaisuus voi aluksi kuulosta ristiriitaiselta. Sen avulla voidaan kuitenkin suunnitella järjestelmiä, joissa rajoitteen vaikutukset ovat voimassa vain tietyissä osissa. (Fielding 2000, 84–85.)

Ladattavan koodin käyttäminen yksinkertaistaa asiakaskomponentteja, sillä harvoin käytettyjä toiminnallisuuksia voidaan ladata ja ottaa käyttöön vasta sitten, kun niitä tarvitaan. Palvelin puolestaan voi välttyä suorittamasta tehtäviä, jotka se voi antaa asiakkaiden vastuulle. Ladattavan koodin käyttäminen johtaa toisaalta järjestelmän näkyvyyden heikentymiseen, sillä koodia on vaikeampi tulkita kuin tavallista dataa. Käytännössä asiakkaat joutuvat olettamaan, että palvelimelta ladattu koodi on turvallista suorittaa. (Fielding 2000, 53, 84.)

### 3.3 HTTP

HTTP (Hypertext Transfer Protocol) on sovelluskerroksen tiedonsiirtoprotokolla, jota käytetään hypermediajärjestelmissä. Se on yksi webin keskeisimmistä protokollista. Sen tyypillisin käyttötapaus on HTML-sivujen välittäminen palvelimelta asiakkaalle. Nimestään huolimatta HTTP:n avulla voidaan välittää melkein mitä tahansa dataa, kuten tekstiä, videoita ja kuvia. (Richardson & Ruby 2007, 5–7.)



Kuvio 2. HTTP-pyyntö ja -vastaus.

HTTP on REST-arkkitehtuurityylin mukainen eli perustuu asiakas-palvelin-malliin. HTTP-transaktiot muodostuvat HTTP-pyyntöistä ja HTTP-vastauksista (kuvio 2). Niiden muoto on tarkasti määritelty, mutta varsinaiseen siirrettävään dataan HTTP ei ota juurikaan kantaa, koska protokolla on suunniteltu olemaan yleiskäyttöinen. (Richardson & Ruby 2007, 5–7.)

Vaikka HTTP itsessään on tilaton protokolla, se ei kuitenkaan voi estää palvelinta tallentamasta sovelluksen tilatietoja. HTTP:n käyttäminen ei näin ollen takaa, että järjestelmä on REST-arkkitehtuurityylin tilattomuus-rajoitteen (luku 3.2.2) mukainen. (Richardson & Ruby 2007, 86–89.)

Protokollasta on vuosikymmenten kuluessa julkaistu useita eri versioita, jotka ovat:

- HTTP/0.9 (v. 1991)
- HTTP/1.0 (v. 1996)
- HTTP/1.1 (v. 1997)
- HTTP/2 (v. 2015) (MDN 2016).

Tässä opinnäytetyössä käsitellään versiota HTTP/1.1. Kyseinen versio on nykyisin kaikista neljästä versiosta käytetyin.

### 3.3.1 Pyyntö

HTTP-transaktio asiakkaan ja palvelimen välillä tapahtuu tyypillisesti seuraavasti: Asiakas avaa yhteyden lähettämällä palvelimelle pyynnön. Käsiteltyään pyynnön palvelin lähettää asiakkaalle vastauksen. Kun vastaus on lopulta välitetty, palvelin sulkee yhteyden. (RFC 7230 2014, 7–8.)

Metodi	URI	HTTP:n versio
Otsikkotiedot		
Viestirunko		

Kuvio 3. HTTP-pyyntöjen rakenne (RFC 7230 2014, 19–34).

Kuviossa 3 on esitetty pyynnön rakenne, joka koostuu viidestä eri osasta, jotka ovat:

- metodi (GET, POST, PUT, DELETE jne.)
- URI (käytetään resurssin tunnistamiseen)
- HTTP:n versio (HTTP/1.0, HTTP/1.1, HTTP/2 jne.)
- otsikkotiedot (pakollisia ja valinnaisia avain-arvopareja)
- viestirunko (teksti- tai binäärimuotoinen) (RFC 7230 2014, 19–34).

```
GET /books/123 HTTP/1.1
Host: api.example.org
Accept: application/ld+json
```

Kuvio 4. Esimerkki HTTP-pyyntöstä.

Kuviossa 4 on esitetty esimerkki kuvitteellisesta pyynnöstä, jonka asiakas lähettää palvelimelle. Kyseisen palvelimen palveluvalikoimaan kuuluu REST-rajapinta, jonka avulla asiakas voi lisätä, poistaa, päivittää ja hakea tietoja kirjoista. Esimerkissä asiakas haluaa hakea yhden ennalta lisätyn kirjan tiedot. Asiakas tietää, että kirjan tunnus on "123", ja että se löytyy kokoelmasta "books". Näiden tietojen ja palvelimen verkkotunnuksen avulla voidaan muodostaa esimerkkipyynnön kaksi ensimmäistä riviä. Pyyntö kolmas rivi puolestaan ilmaisee, että asiakas haluaa vastaanottaa kirjan tiedot JSON-LD-formaatissa. Viestirunkoa ei kyseisessä esimerkissä tarvita.

### 3.3.2 Vastaus

Palvelimelta asiakkaalle välitetyt vastaukset noudattavat pääosin samaa mallia kuin asiakkaalta palvelimelle välitetyt pyynnot. Vastauksissa metodi ja URI on korvattu tilakoodilla, ja otsikkotiedot ovat erilaisia. (RFC 7231 2014, 47, 64.)

HTTP:n versio	Tilakoodi
Otsikkotiedot	
Viestirunko	

Kuvio 5. HTTP-vastauksen rakenne  
(RFC 7230 2014, 19–34).

Kuviossa 5 on esitetty vastauksen rakenne, joka koostuu neljästä osasta, jotka ovat:

- HTTP:n versio (HTTP/1.0, HTTP/1.1, HTTP/2 jne.)
- tilakoodi (200, 301, 404, 500 jne.)
- otsikkotiedot (pakollisia ja valinnaisia avain-arvopareja)
- viestirunko (teksti- tai binäärimuotoinen) (RFC 7230 2014, 19–34).

```
HTTP/1.1 200 OK
Content-Type: application/ld+json

{
  "@context": "http://schema.org/",
  "@type": "Book",
  "name": "RESTful Web APIs",
  "isbn": "978-1-4493-5806-8"
}
```

Kuvio 6. Esimerkki HTTP-vastauksesta.

Kuviossa 6 on esitetty esimerkki kuvitteellisesta vastauksesta, jonka palvelin lähettää asiakkaalle. Kyseinen esimerkivastaus liittyy kuviossa 4 esitettyyn pyyntöön. Vastauksen ensimmäinen rivi ilmaisee käytetyn HTTP:n version ja tilakoodin, jonka mukaan pyynnön käsittelyssä ei ilmennyt ongelmia. Toinen rivi kertoo, että viestirunko sisältää JSON-LD-muotoista dataa. Kolmas rivi on tyhjä, koska otsikkotiedot täytyy erottaa viestirungosta tyhjällä rivillä. Vastauksen viimeiset rivit sisältävät itse viestirungon.

### 3.3.3 Metodit

Taulukossa 1 on esitetty REST-rajapintojen yhteydessä yleisimmin käytetyt metodit. Muita metodeja: HEAD, OPTIONS, TRACE ja CONNECT käytetään harvemmin. (Richardson, Amundsen & Ruby 2013, 33).

Taulukko 1. Yleisimmin käytetyt HTTP-metodit  
(Richardson, Amundsen & Ruby 2013, 33).

HTTP-metodi	Kuvaus
GET	Haetaan aikaisemmin luodun resurssin representaatio.
POST	Luodaan uusi resurssi.
PUT	Päivitetään resurssi.
DELETE	Poistetaan resurssi.

Taulukossa 1 mainitut HTTP-metodit vastaavat käytännössä CRUD-operaatioita. Resursseja voidaan päivittää myös RFC 5789:n määrittelemällä PATCH-metodilla. (Richardson, Amundsen & Ruby 2013, 34, 179.)

### 3.3.4 Tilakoodit

Taulukossa 2 on esitetty HTTP-protokollan määrittelemät tilakoodiryhmät. Kyseiset kolminumeroiset koodit kertovat asiakkaalle tietoja tehdyn pyynnön toteutumisesta. Tilakoodin ensimmäinen numero kertoo tilakoodiryhmän, joita on yhteensä viisi. Kaksi viimeistä numeroa erottelevat tilakoodin kyseisen tilakoodiryhmän sisällä. (RFC 7231 2014, 47–48.)

Taulukko 2. HTTP-protokollan tilakoodiryhmät  
(RFC 7231 2014, 47–48).

Tilakoodiryhmä	Kuvaus
1xx	Informatiiviset tilakoodit.
2xx	Onnistumista ilmaisevat tilakoodit.
3xx	Uudelleenohjaukseen liittyvät tilakoodit.
4xx	Asiakkaan tekemiä virheitä käsittelevät tilakoodit.
5xx	Palvelimen tekemiä virheitä käsittelevät tilakoodit.

### 3.3.5 Otsikkotiedot

HTTP-protokolla määrittelee suuren määrän otsikkotietoja, ja niitä käytetään sekä pyynnöissä että vastauksissa. Ne muodostuvat avain-arvopareista, ja voivat olla joko pakollisia tai valinnaisia. Otsikkotietoja voidaan tarvittaessa määritellä itse. (RFC 7231 2014, 33–73.) Taulukossa 3 on esitetty yleisesti käytettyjä otsikkotietoja.

Taulukko 3. Yleisesti käytettyjä otsikkotietoja  
(RFC 7231 2014, 33–73).

Otsikkotieto	Kuvaus
Host	Palvelimen osoite ja portti, johon asiakas ottaa yhteyden.
Date	Päivämäärä ja aika, jolloin pyyntö tai vastaus lähetettiin.
Accept	Lista vastauksen mediatyypeistä, jotka asiakas hyväksyy.
Accept-Encoding	Pakkausformaatit, jotka asiakas hyväksyy käytettäväksi.
Content-Type	Mediatyyppi, jota käytetään pyynnössä tai vastauksessa.
Content-Encoding	Pyynnössä tai vastauksessa käytetty pakkausformaatti.

Kirjautumista vaativien verkkopalveluiden yhteydessä käytetään usein evästeisiin liittyviä otsikkotietoja, kuten "Cookie" ja "Set-Cookie". REST-rajapinnoissa niitä ei tulisi kuitenkaan käyttää, koska ne ovat tilattomuus-rajoitteen (luku 3.2.2) vastaisia. (Richardson & Ruby 2007, 252–253.)

## 3.4 URI

URI (Uniform Resource Identifier) on kompakti merkkijono, jota käytetään resurssin tunnistamiseen. Sen avulla voidaan ilmaista resurssin sijainti, nimi tai molemmat. URI-tunnuksia käytetään tyypillisesti muun muassa verkkosivujen tunnistamiseen. (RFC 3986 2005, 1, 7.)

URI:n yhteydessä käytetään toisinaan kahta muuta termiä, jotka ovat URL ja URN. URL (Uniform Resource Locator) on URI, joka ilmaisee resurssin sijainnin ja kertoo miten kyseisen resurssin representaation voi noutaa. Myös URN (Uniform Resource Name) on URI. Se ilmaisee resurssin pysyvän, sijainnista riippumattoman nimen. (RFC 3986 2005, 7.)

### 3.4.1 Syntaksi

URI:n syntaksi koostuu hierarkkista osista, joiden pakollisuus on tapauskohtaista. Kyseiset osat erotetaan toisistaan varatuilla erikoismerkeillä, kuten kaksoispisteellä, vinoviivalla tai ristikkomerkillä. (RFC 3986 2005, 11–25.)

```
[skeema:][//omistaja][polku][?kysely][#tarkenne]
```

Kuvio 7. URI-tunnuksen syntaksi  
(RFC 3986 2005, 16–25).

Kuviossa 7 on esitetty URI-tunnuksen syntaksi, joka koostuu seuraavista eri osista:

- skeema (http, ftp, mailto, data, file jne.)
- omistaja (verkkotunnus, IP-osoite jne.)
- polku (resurssin tunniste)
- kysely (avain-arvopareja)
- tarkenne (käytetään asiakaskomponenteissa) (RFC 3986 2005, 16–25).

Skeema ja polku ovat URI:n ainoat pakolliset osat, tosin polku saa olla myös tyhjä. Kaikkien osien täytyy joka tapauksessa olla kuviossa 7 esitettyssä järjestyksessä. (RFC 3986 2005, 16.)

### 3.4.2 Resurssi

URI:n avulla tunnistettava resurssi voi olla joko fyysinen objekti tai abstrakti käsite. Käytännössä resurssi voi olla melkein mikä tahansa asia, joka voidaan identifioida, kuten elektroninen dokumentti, kuva, säätiedotus tai muiden resurssien kokoelma. Resurssin ei tarvitse olla saatavissa internetin kautta, joten se voi olla edellisten lisäksi esimerkiksi ihminen tai yritys. (RFC 3986 2005, 1, 5.)

Jokaisella resurssilla on vähintään yksi URI, jolla kyseinen resurssi tunnistetaan. Asiakaskomponenttien näkökulmasta ei ole merkitystä mikä resurssi on, koska ne eivät koskaan käsittele itse resursseja, vaan kyseisten resurssien representaatioita. (Richardson, Amundsen & Ruby 2013, 30.)



### 3.4.3 Representaatio

Representaatio on informaatiota, joka kuvaa tietyn resurssin mennyttä, nykyistä tai haluttua tilaa. Se koostuu datasta, ja kyseiseen dataan liittyvästä metatiedosta. Representaatioita voidaan välittää sekä palvelimelta asiakkaalle että päinvastoin. Yhdellä resurssilla voi olla monta representaatiota, ja ne voivat muuttua erilaisiksi vaikka itse resurssi pysyy samana. (Richardson, Amundsen & Ruby 2013, 29–33.)

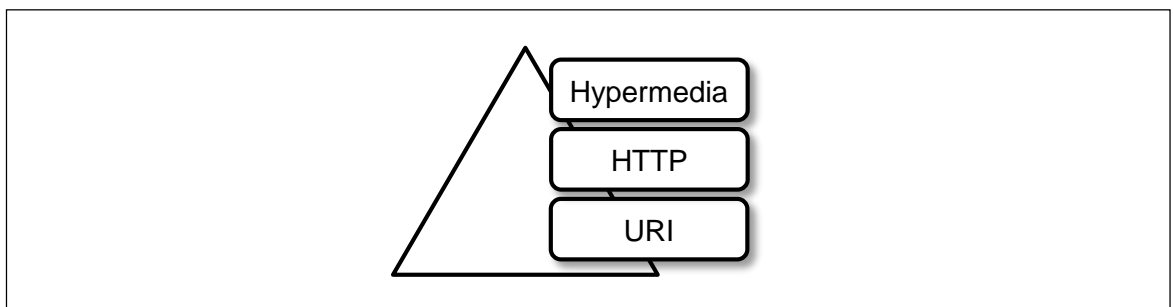
REST-rajapintojen yhteydessä resurssi on usein tietoa, joka sijaitsee tietokannassa. Kyseinen resurssi voi olla esimerkiksi säätiedotus. Kun asiakaskomponentti tekee pyynnön, REST-rajapinta palauttaa representaation, joka kuvaa edellä mainitun resurssin tilaa tietyssä ajanhetkenä. REST-rajapinnan palauttama representaatio voi olla esimerkiksi JSON-LD-muotoista dataa, joka sisältää kyseisen säätiedotuksen. (Webber, Parastatidis & Robinson 2010, 7–11.)

## 4 LUOKITTELU

REST-rajapintojen luokittelu on tärkeää, koska useimmat nykyisistä toteutuksista eivät noudata REST-tyylin kaikkia rajoitteita, eivätkä siten ole oikeita REST-rajapintoja. Tyypillisesti luokitteluun käytetään Leonard Richardsonin kehittämää kypsyyssmallia, joka käydään seuraavaksi tarkemmin läpi.

### 4.1 Richardsonin kypsyyssmalli

Leonard Richardsonin kehittämä kypsyyssmalli (kuvio 8) koostuu kolmesta tasosta. Kyseiset kolme tasoa käsittelevät URI-tunnusten, HTTP:n ja hypermedian käyttöä. Kypsyyssmalli määrittelee myös neljännen tason (taso 0). Sen mukaiset palvelut eivät hyödynnä yhtäkään edellä mainituista kolmesta tekniikasta vaaditulla tavalla. (Webber, Parastatidis & Robinson 2010, 18–20.)



Kuvio 8. Richardsonin kypsyyssmalli.

Richardsonin kypsyyssmallin korkein taso (taso 3) ei takaa, että REST-rajapinta on REST-arkkitehtuurityylin mukainen, koska kypsyyssmalli ei ota kaikkea huomioon. Tasoa 3 tulisi ajatella REST-tyylin edellytyksenä, eikä takeena sen mukaisuudesta. (Fowler 2010.)

Kypsyyssmallin neljä tasoa käydään läpi seuraavissa alaluvuissa esimerkkien avulla. Kyseisessä esimerkkitalanteessa haetaan luettelo kirjoista, jotka on julkaistu vuosien 2005 ja 2015 välillä, ja joiden kirjoittamiseen Leonard Richardson on osallistunut. Esimerkit koostuvat HTTP-pyyntöistä ja -vastauksista, joita välitetään kuvitteellisen REST-rajapinnan ja asiakaskomponentin välillä. HTTP-pyyntöistä ja vastauksista on jätetty pois ylimääräiset otsikkotiedot, jotka ovat tässä esimerkissä tarpeettomia.

#### 4.1.1 Taso 0

Tason 0 mukaisten palveluiden yhteydessä käytetään yhtä URI-tunnusta ja yhtä HTTP-metodia (tyypillisesti POST). Tämän tason palvelut käyttävät HTTP:tä lähinnä yksinkertaisena tunnelointimekanismina, jonka avulla voidaan tehdä RPC-kutsuja. (Richardson 2008.)

```
POST /bookService HTTP/1.1

{
  "bookSearchRequest": {
    "author": "leonard-richardson",
    "published": "2005-2015"
  }
}
```

Kuvio 9. Tason 0 mukainen pyyntö.

Kuviossa 9 on esitetty esimerkki kypsyyssmallin tason 0 mukaisesta pyynnöstä. Pyyntöns ensimmäinen rivi ilmaisee pyyntöns yhteydessä käytetyn HTTP-metodin (POST) ja URI:n (/bookService). Seuraavat rivit sisältävät varsinaisen viestirungon. Kyseinen viestirunko sisältää suoritettavan operaation nimen (bookSearchRequest) ja hakuehdot (author ja published).

```
HTTP/1.1 200 OK

{
  "bookList": [
    {
      "title": "RESTful Web APIs",
      "isbn": "978-1-4493-5806-8"
    },
    {
      "title": "RESTful Web Services",
      "isbn": "978-0-596-52926-0"
    }
  ]
}
```

Kuvio 10. Tason 0 mukainen vastaus.

Kuviossa 10 on esitetty esimerkki kypsyyssmallin tason 0 mukaisesta vastauksesta. Vastauksen viestirunko sisältää pyyntöns (kuvio 9) tapaan JSON-muotoista dataa. Tason 0 yhteydessä viestirunkojen sisällöllä ei ole vielä merkitystä, koska kyseinen taso ei ota niihin kantaa.

### 4.1.2 Taso 1

Tason 1 mukaisten palveluiden yhteydessä käytetään useita URI-tunnuksia, mutta vain yhtä HTTP-metodia (tyypillisesti POST). Edellisellä tasolla (taso 0) URI:n tarkoituksena oli tunnistaa palvelu. Tällä tasolla niiden avulla tunnistetaan yksittäisiä resursseja. (Richardson 2008.)

```
POST /authors/leonard-richardson HTTP/1.1

{
  "bookSearchRequest": {
    "published": "2005-2015"
  }
}
```

Kuvio 11. Tason 1 mukainen pyyntö.

Kuviossa 11 on esitetty esimerkki kypsyysmallin tason 1 mukaisesta pyynnöstä. Toisin kuin tasolla 0, kirjailijoita käsitellään nyt resursseina. Tämä muutos tarkoittaa käytännössä sitä, että kirjailijan yksilöllinen tunnus (leonard-richardson) on siirretty osaksi URI-tunnusta.

```
HTTP/1.1 200 OK

{
  "bookList": [
    {
      "id": "1234",
      "title": "RESTful Web APIs",
      "isbn": "978-1-4493-5806-8"
    },
    {
      "id": "5678",
      "title": "RESTful Web Services",
      "isbn": "978-0-596-52926-0"
    }
  ]
}
```

Kuvio 12. Tason 1 mukainen vastaus.

Kuviossa 12 on esitetty esimerkki kypsyysmallin tason 1 mukaisesta vastauksesta. Kyseinen esimerkki sisältää samat perustiedot kuin tason 0 esimerkki (kuvio 10). Ainoa ero on se, että kirjoja käsitellään tällä tasolla resursseina, joten niillä on oma tunnuksensa (id).

### 4.1.3 Taso 2

Tason 2 mukaisten palveluiden yhteydessä käytetään monen URI:n lisäksi useita HTTP-metodeja. Toisin kuin tasojen 0 tai 1 mukaiset palvelut, tämän tason palvelut käyttävät tyypillisesti neljää HTTP-metodia, jotka ovat GET, POST, PUT ja DELETE. Jotta palvelu olisi tason 2 mukainen, sen täytyy lisäksi palauttaa asianmukaisia tilakoodeja. (Fowler 2010.)

```
GET /authors/leonard-richardson/books?published=2005-2015 HTTP/1.1
```

Kuvio 13. Tason 2 mukainen pyyntö.

Kuviossa 13 on esitetty esimerkki kypsyysmallin tason 2 mukaisesta pyynnöstä. Koska kaikki HTTP-metodit ja URI:n ominaisuudet ovat nyt vapaasti käytettävissä, esimerkkipyyntöissä ei enää tarvita varsinaista viestirunkoa, kuten tasoissa 0 ja 1. Tällä tasolla suoritettavan operaation nimeä ei siis enää mainita missään, vaan se on korvattu kyseisen esimerkkipyyntöön tapaukseen soveltuvalla GET-metodilla. Edellisessä tasossa (taso 1) yhä käytetty hakuehto (published) on nyt siirretty URI:n yhteyteen.

```
HTTP/1.1 200 OK

{
  "bookList": [
    {
      "id": "1234",
      "title": "RESTful Web APIs",
      "isbn": "978-1-4493-5806-8"
    },
    {
      "id": "5678",
      "title": "RESTful Web Services",
      "isbn": "978-0-596-52926-0"
    }
  ]
}
```

Kuvio 14. Tason 2 mukainen vastaus.

Kuviossa 14 on esitetty esimerkki kypsyysmallin tason 2 mukaisesta vastauksesta. Vastaus on tässä tapauksessa täysin samanlainen kuin tason 1 vastaus (kuvio 12), koska taso 2 ei tuo muutoksia kyseiseen esimerkkiin.

#### 4.1.4 Taso 3

Tason 3 mukaisten palveluiden yhteydessä käytetään useita URI-tunnuksia ja HTTP-metodeja. Ainoa ero edellisen tason (taso 2) palveluihin verrattuna on se, että tämän tason palvelut noudattavat REST-arkkitehtuurityylin HATEOAS-rajoitetta. (Fowler 2010.)

```
GET /authors/leonard-richardson/books?published=2005-2015 HTTP/1.1
```

Kuvio 15. Tason 3 mukainen pyyntö.

Kuviossa 15 on esitetty esimerkki kypsyysmallin tason 3 mukaisesta pyynnöstä. Pyyntö on tässä tapauksessa täysin samanlainen kuin tason 2 pyyntö (kuvio 13), koska taso 3 ei tuo muutoksia kyseiseen esimerkkiin.

```
HTTP/1.1 200 OK

{
  "bookList": [
    {
      "id": "1234",
      "title": "RESTful Web APIs",
      "isbn": "978-1-4493-5806-8",
      "links": [
        { "rel": "order", "uri": "/books/1234" }
      ]
    },
    {
      "id": "5678",
      "title": "RESTful Web Services",
      "isbn": "978-0-596-52926-0",
      "links": [
        { "rel": "order", "uri": "/books/5678" }
      ]
    }
  ]
}
```

Kuvio 16. Tason 3 mukainen vastaus.

Kuviossa 16 on esitetty esimerkki kypsyysmallin tason 3 mukaisesta vastauksesta. HATEOAS-rajoitteen noudattaminen näkyy käytännössä siten, että vastaukseen on lisätty linkkejä. Kyseiset linkit ilmaisevat, että molemmat kirjat ovat tilattavissa (rel). Ne kertovat myös, että tilausprosessi tulee suorittaa manipuloimalla URI-tunnuksen osoittamaa resurssia.

## 4.2 Tyypillinen REST-rajapinta

REST-rajapinnat ovat tyypillisesti Richardsonin kypsyyssmallin tason 2 mukaisia. (Heritage 2015; ProgrammableWeb 2016). Niiden yhteydessä käytetään useita URI-tunnuksia ja HTTP-metodeja. URI-tunnusten tehtävänä on tunnistaa resurssit. HTTP-metodeja puolestaan käytetään edellä mainittujen resurssien manipulointiin. Tason 2 mukaisten REST-rajapintojen yhteydessä käytetään myös asianmukaisia tilakoodeja, jotka kertovat tietoja tehdystä pyynnöstä. (Webber, Parastatidis & Robinson 2010, 18–20.)

Tyypilliset REST-rajapinnat eivät sisällytä linkkejä vastauksiin, joten ne eivät ole HATEOAS-rajoitteen mukaisia. Tämä tarkoittaa käytännössä sitä, että tyypilliset REST-rajapinnat eivät ole REST-rajapintoja sanan varsinaisessa merkityksessä. (Fielding 2008.) Ilman HATEOAS-rajoitteen mukaisia linkkejä asiakaskomponentti ei voi tietää, mitä se voi tehdä seuraavaksi, ellei kyseistä tietoa ole annettu jollakin muulla tavalla. Yleensä tämä tieto on kovakoodattu osaksi asiakaskomponenttia. (Webber, Parastatidis & Robinson 2010, 93–96.)

## 4.3 HATEOAS

HATEOAS (Hypermedia as the Engine of Application State) on kaikista keskeisin REST-arkkitehtuurityylin rajoite. Rajoite edellyttää, että asiakkaalle välitettyjen vastauksien tulee sisältää linkkejä, jotka kertovat kuinka palvelua voidaan käyttää. Nämä linkit yhdistävät palvelun resurssit toisiinsa ja kuvaavat niiden ominaisuudet koneluettavalla tavalla. (Webber, Parastatidis & Robinson 2010, 93–96.)

HATEOAS-rajoitteen mukainen REST-rajapinta toimii periaatteessa samoin kuin tyypillinen verkkosivusto. Molemmissa tapauksissa palvelun käyttäminen perustuu vastauksiin sisällytettyihin linkkeihin, jotka yhdistävät resurssit/verkkosivut toisiinsa. Jotta linkkien merkitys olisi asiakkaille selvä, tarvitaan formaatti, joka tukee niitä. Tällaisia ovat esimerkiksi JSON-LD, HAL ja Siren. Useimpien REST-rajapintojen yhteydessä käytetään kuitenkin JSON- tai XML-formaattia, jotka eivät tue linkkejä. Näitä formaatteja käytettäessä asiakaskomponentit täytyy ohjelmoida siten, että ne osaavat tunnistaa kyseiset linkit. (Richardson, Amundsen & Ruby 2013, 109–132.)

### 4.3.1 Hyödyt

HATEOAS-rajoitteen noudattaminen helpottaa REST-rajapintojen kehittämistä, koska resurssien yksilöllisiä tunnuksia (URI) voidaan muuttaa aiempaa vapaammin. Jos rajoitetta ei noudatettaisi, kaikki asiakaskomponentit täytyisi päivittää tunnuksiin tehtyjen muutosten jälkeen. Käytännössä vanhat asiakaskomponentit jättävät uusia resursseja koskevat linkit huomioimatta, koska ne eivät tiedä niiden merkitystä. REST-rajapintoihin voidaan siis lisätä myöhemmin uusia lisäominaisuuksia ilman että niistä aiheutuu haittaa vanhoille päivittämättömille asiakaskomponenteille. (Stowe 2014.)

Koska vastaukset sisältävät linkkejä, asiakaskomponenttien kehittäjät voivat selata REST-rajapinnan tarjoamia palveluita käyttämällä tähän soveltuvia ohjelmistoja. Tämä voi auttaa kehittäjiä ymmärtämään miten kyseinen REST-rajapinta toimii. Vastauksiin on myös mahdollista lisätä linkkejä, jotka ohjaavat kehittäjät resurssia käsittelevään dokumenttiin. Kyseinen dokumentti voi olla esimerkiksi HTML-sivu. (Stowe 2014.)

### 4.3.2 Haitat

HATEOAS-rajoitteen noudattamisella on hyvien puolien lisäksi huonoja puolia. Koska vastauksiin joudutaan lisäämään linkkejä, siirrettävän datan määrä kasvaa. Joissain tapauksissa ero voi olla hyvin suuri verrattuna edeltävään tilanteeseen, jossa linkkejä ei käytetty. Useimpien REST-rajapintojen kohdalla siirrettävän datan määrä on kuitenkin melko pieni, käytettiin vastauksien yhteydessä linkkejä tai ei. (Stowe 2014.)

Toinen keskeinen haitta liittyy REST-rajapintojen suunnitteluun ja toteutukseen. HATEOAS-rajoitteen noudattaminen lisää työn määrää etenkin lyhyellä aikavälillä. Vaikka REST-rajapinta noudattaisi kyseistä rajoitetta, se ei kuitenkaan takaa sitä, että asiakaskomponenttien kehittäjät hyödyntäisivät ominaisuuksia oikealla tavalla. Tämä voi johtaa esimerkiksi tilanteeseen, jossa REST-rajapintaan ei voida tehdä muutoksia, koska kyseiset muutokset aiheuttaisivat haittaa asiakaskomponenteille. (Stowe 2014.)



## 5 ESIMERKKISOVELLUS

REST-rajapintojen luomiseen on olemassa paljon erilaisia sovelluskehyskiä. Tässä luvussa tutustutaan ensin Flask-sovelluskehukseen ja sen Flask-RESTful-lisäosaan. Sen jälkeen kyseisten ohjelmistojen avulla luodaan yksinkertainen REST-rajapinta.

### 5.1 Flask

Flask on Python-pohjainen mikrosovelluskehys, jonka avulla voidaan luoda erilaisia verkkosovelluksia ja -palveluita. Sitä on kehitetty aktiivisesti vuodesta 2010 lähtien. Flask on monien muiden avoimen lähdekoodin ohjelmistojen tavoin BSD-lisensioitu. Se soveltuu sekä pienten että isojen järjestelmien toteuttamiseen. Sen yhteydessä käytetään yleensä erilaisia lisäosia, joiden avulla puuttuvia toimintoja voidaan lisätä. (Ronacher 2016.)

Flask on muihin vastaaviin sovelluskehyskiin verrattuna suhteellisen yksinkertainen. Sen yhteydessä ei tarvita yhtä paljon boilerplate-koodia kuin esimerkiksi Djangossa, joka on samankaltainen, mutta ominaisuuksiltaan selvästi laajempi sovelluskehys. Flask ei aseta rajoituksia esimerkiksi sen yhteydessä käytettävällä tietokantatyypille. Yksinkertaisen hello world -verkkosovelluksen luominen vaatii noin kymmenen riviä koodia. Muita Flaskin kaltaisia sovelluskehyskiä ovat esimerkiksi Bottle ja Falcon. (Ronacher 2016.)

### 5.2 Flask-RESTful

Flask-RESTful on Flaskin lisäosa, jota käyttämällä voidaan luoda REST-rajapintoja. Sitä on kehitetty vuodesta 2012 lähtien, ja se on Flaskin tapaan BSD-lisensioitu. Flask-RESTful tekee REST-rajapintojen luomisesta helpompaa ja nopeampaa, mutta kyseisiä rajapintoja voidaan haluttaessa luoda myös pelkän Flaskin avulla. Se on Flaskin tapaan ominaisuuksiltaan kevyt, mutta toisaalta hyvin mukautuva. Flask-RESTful on konfiguroitu käyttämään oletuksena JSON-muotoisia viestejä, mutta muitakin formaatteja voidaan käyttää. (Burke ym. 2015.)

### 5.3 Toteutus

Seuraavaksi käydään läpi, miten REST-rajapinta voidaan luoda käyttäen apuna Flask-sovelluskehystä ja sen Flask-RESTful-lisäosaa. Luotava REST-rajapinta on Richardsonin kypsyyssmallin tason 2 mukainen, eli sen yhteydessä käytetään useita URI-tunnuksia ja HTTP-metodeja. Tämänkaltaiset REST-rajapinnat ovat nykyään hyvin yleisiä.

REST-rajapinnan avulla voidaan lisätä, poistaa, päivittää ja hakea tietoja kirjoista. Jokaisella kirjalla on oma tunnuksensa ja kaksi kenttää, jotka ovat title ja isbn. Kyseisen REST-rajapinnan toimintaperiaate on esitetty tarkemmin taulukossa 4.

Taulukko 4. REST-rajapinnan toimintaperiaate.

HTTP-metodi	URI-tunnus	Kuvaus
GET	/books	Haetaan kaikkien kirjojen tiedot.
GET	/books/[id]	Haetaan yhden kirjan tiedot.
POST	/books	Lisätään uusi kirja.
PUT	/books/[id]	Päivitetään kirja.
DELETE	/books/[id]	Poistetaan kirja.

REST-rajapintojen toimintaan liittyy tyypillisesti lukuisia erilaisia ohjelmistoja. Tämän rajapinnan yhteydessä käytetyt keskeisimmät ohjelmistot ja niiden versiot ovat:

- Debian 8.4 (kerneli 3.16)
- Python 2.7.9
- Flask 0.10.1
- Flask-RESTful 0.3.5.

Edellä mainittujen eri ohjelmistojen asentamista ei käydä tässä tapauksessa läpi. Oletuksena on, että kaikki edellä mainitut ohjelmistot on asennettu oletusasetuksin.

Tämän REST-rajapinnan yhteydessä tarvitsee luoda vain yhden Python-tiedoston, jolle annetaan nimi api.py. Kyseinen tiedosto tulee pitämään sisällään myöhemmin esiteltävät lähdekoodit. Se voidaan tallentaa esimerkiksi käyttäjän kotikansioon.

REST-rajapinnan ohjelmointi aloitetaan importoimalla sen yhteydessä tarvittavat luokat (kuvio 17). Seuraavaksi Flask- ja Api-luokista luodaan ilmentymät (instanssit). Näiden pakollisten osioiden jälkeen voidaan keskittyä varsinaiseen sovellukseen.

```
from flask import Flask
from flask_restful import abort, Api, reqparse, Resource

app = Flask(__name__)
api = Api(app)

books = {
    1: {'title': 'RESTful Web APIs', 'isbn': '978-1-4493-5806-8'},
    2: {'title': 'RESTful Web Services', 'isbn': '978-0-596-52926-0'}
}

def check_if_exists(id):
    if id not in books:
        abort(404, message="Book {} does not exist".format(id))

parser = reqparse.RequestParser()
parser.add_argument('title')
parser.add_argument('isbn')
```

Kuvio 17. REST-rajapinnan lähdekoodi, osa 1/3.

Tämän REST-rajapinnan yhteydessä kirjojen tiedot tallennetaan books-muuttujaan, joka alustettu kahden eri kirjan tiedoilla. Useimmiten tietovarastona toimii tietokanta, joka voi olla joko perinteinen relaatiotietokanta tai NoSQL-tyyppinen tietokanta.

Seuraavaksi määritellään funktio `check_if_exists`, jolla voidaan tarkistaa löytyykö books-muuttujasta kirjaa, jolla on sama tunnus kuin annetulla parametrillä `id`. Jos haettua kirjaa ei löydy, kyseinen funktio palauttaa virheviestin ja asianmukaisen tilakoodin 404 (Not Found).

Jotta HTTP-pyyntöjen sisältämät tiedot (`title` ja `isbn`) voitaisiin validoida, määritellään jäsennin (`parser`). Samalla voitaisiin haluttaessa asettaa tietoihin liittyviä lisäehtoja. Lisäehtojen avulla voidaan esimerkiksi määritellä ovatko tiedot pakollisia vai eivät.

Seuraavaksi määritellään Book-luokka (kuvio 18), joka sisältää kolme metodia: `get`, `delete` ja `put`. Kyseiset kolme metodia vastaavat samannimisiä HTTP-metodeja. Tämä tarkoittaa sitä, että HTTP-pyyntö (`/books/[id]`), jonka yhteydessä käytetään esim. HTTP-metodia DELETE, johtaa Book-luokan delete-metodin suorittamiseen.

```

class Book(Resource):
    def get(self, id):
        check_if_exists(id)
        return books[id]

    def delete(self, id):
        check_if_exists(id)
        del books[id]
        return '', 204

    def put(self, id):
        args = parser.parse_args()
        book = {'title': args['title'], 'isbn': args['isbn']}
        books[id] = book
        return book, 201

class BookList(Resource):
    def get(self):
        return books

    def post(self):
        args = parser.parse_args()
        id = max(books.keys()) + 1
        books[id] = {'title': args['title'], 'isbn': args['isbn']}
        return books[id], 201

```

Kuvio 18. REST-rajapinnan lähdekoodi, osa 2/3.

Book-luokan get- ja delete-metodien yhteydessä käytetään aiemmin mainittua funktiota `check_if_exist`. Get-metodi palauttaa yhden kirjan tiedot ja tilakoodin 200. Delete-metodin yhteydessä poistetaan yksi kirja `books`-muuttujasta ja palautetaan oletuksena käytettävän tilakoodin 200 (OK) sijasta tilakoodi 204 (No Content). Put-metodi puolestaan päivittää yhden kirjan tiedot ja palauttaa lopuksi nämä tiedot. Kyseisen metodin yhteydessä hyödynnetään aiemmin mainittua parser-jäsennintä. Put-metodi palauttaa tilakoodin 201 (Created).

Seuraava luokka, `BookList`, sisältää kaksi metodia, jotka ovat `get` ja `post`. Nämä kaksi metodia vastaavat edellisen Book-luokan tavoin samannimisiä HTTP-metodeja. `BookList`-luokan `get`-metodi palauttaa kaikkien kirjojen tiedot toisin kuin `Book`-luokan samanniminen metodi, joka palautti yksittäisen kirjan tiedot. Molemmat `get`-metodit palauttavat tilakoodin 200 (OK). `Post`-metodia käytetään uuden kirjan lisäämiseen. Sen yhteydessä käytetään aiemmin mainitun `put`-metodin tavoin parser-jäsennintä. Lisättävälle kirjalle määritellään tunnus (`id`), jolla kyseinen kirja voidaan myöhemmin tunnistaa. `Post`-metodi palauttaa lisätyn kirjan tiedot ja tilakoodin 201 (Created).

Kun molemmat luokat on määritelty, seuraava tehtävä on yhdistää ne tiettyihin URI-tunnuksiin (kuvio 19). BookList-luokka yhdistetään URI-tunnukseen /books. Book-luokka puolestaan yhdistetään URI-tunnukseen /books/[id], jossa id tarkoittaa kirjan yksilöllistä tunnusta. Tunnus muutetaan samassa yhteydessä kokonaisluvuksi, jotta tätä muutosta ei tarvitse enää tehdä Book- ja BookList-luokkien yhteydessä.

```
api.add_resource(BookList, '/books')
api.add_resource(Book, '/books/<int:id>')

if __name__ == '__main__':
    app.run(debug=True)
```

Kuvio 19. REST-rajapinnan lähdekoodi, osa 3/3.

Lopuksi luodaan ehto, joka käynnistää Flaskin oman palvelimen jos kyseinen sovellus ajetaan suoraan komentorivin kautta. Samalla asetetaan debug-tila päälle, joka on hyödyllinen testaamisen yhteydessä muun muassa parempien virheviestien vuoksi.

Kun sovellus on saatu valmiiksi ja tallennettu aiemmin mainittuun api.py-tiedostoon, se voidaan käynnistää suoraan komentoriviltä komennon "python api.py" avulla. REST-rajapinta on oletuksena ei-julkinen, joten sitä voidaan käyttää vain samalta koneelta.

REST-rajapinnan toiminta voidaan testata esimerkiksi cURL-sovelluksen avulla. Liitteessä 1 on esitetty viisi cURL-komentoa, joiden avulla kyseisen REST-rajapinnan toiminnot voidaan testata.

## 6 TULOKSET JA YHTEENVETO

Opinnäytetyön tavoitteena oli antaa yleiskuva REST-arkkitehtuuriin perustuvista web-rajapinnoista. Tarkoituksena oli keskittyä etenkin niiden toimintaan ja luokitteluun.

Työn alussa käsiteltiin lyhyesti web-rajapintojen historiaa ja nykytilaa. Seuraavaksi perehdyttiin REST-rajapintoihin tarkemmin käymällä läpi niiden toimintaan liittyvät keskeiset asiat, jotka olivat REST-arkkitehtuuri, HTTP-protokolla ja URI-tunnus. REST-rajapintojen toimintaan perehtymisen jälkeen siirryttiin käsittelemään niiden luokittelua. Aluksi esiteltiin Richardsonin kypsyyssmalli, joka koostuu neljästä tasosta. Tämän jälkeen todettiin, että tyypillinen REST-rajapinta sijoittuu mallin tasolle 2. Lisäksi selvitettiin, mitkä ovat REST-tyylin kiistellyn HATEOAS-rajoitteen vaikutukset. Työn lopuksi luotiin esimerkkisovellus käyttäen apuna Flask-sovelluskehystä ja sen Flask-RESTful-lisäosaa.

REST-rajapinnat ovat kehittyneet parempaan suuntaan, mutta edelleen vain harva toteutus on nimensä mukainen. Kehittäjät ymmärtävät useimmissa tapauksissa URI-tunnusten ja HTTP-metodien oikeaoppisen käytön, mutta eivät hypermedian. Hypermedian eli linkitetyn informaation hyödyt ymmärretään, kun kyseessä on verkkosivusto, mutta syystä tai toisesta tätä ominaisuutta ei kuitenkaan ole otettu vielä laajasti käyttöön REST-rajapinnoissa. Tämä on ongelma, koska hypermedia mahdollistaisi pitkäikäisten ja helpommin muokattavien järjestelmien toteuttamisen. Joissain tapauksissa sen käytöstä voi kuitenkin olla enemmän haittaa kuin hyötyä. Nähtäväksi jää, tuleeko hypermedian käytöstä koskaan yleistä REST-rajapintojen yhteydessä.

## LÄHTEET

- Burke, K., Conroy, K., Horn, R., Stratton, F. & Binet, G. 2015. Flask-RESTful. [Verkkosivusto]. Twilio, Inc. [Viitattu 27.5.2016]. Saatavana: <http://flask-restful.readthedocs.io/>
- Fielding, R. 2000. Architectural Styles and the Design of Network-based Software Architectures. [Verkkójulkaisu]. Irvine: University of California. [Viitattu 29.3.2016]. Saatavana: [https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding\\_dissertation.pdf](https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf)
- Fielding, R. 2008. REST APIs must be hypertext-driven. [Verkkosivu]. [Viitattu 9.5.2016]. Saatavana: <http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven>
- Fowler, M. 2010. Richardson Maturity Model. [Verkkosivu]. [Viitattu 10.5.2016]. Saatavana: <http://martinfowler.com/articles/richardsonMaturityModel.html>
- Heritage, L. 2015. The Movement Towards Hypermedia APIs – Is it Happening?. [Verkkosivu]. Akana, Inc. [Viitattu 9.5.2016]. Saatavana: <https://blog.akana.com/hypermedia-apis/>
- Jansen, G. 2011. The Job of the API Designer. [Verkkosivu]. GitHub, Inc. [Viitattu 3.5.2016]. Saatavana: <https://github.com/geertj/restful-api-design/blob/master/scope.rst>
- MDN. 2016. HTTP. [Verkkosivu]. Mozilla Developer Network. [Viitattu 2.5.2016]. Saatavana: <https://developer.mozilla.org/en-US/docs/Web/HTTP>
- ProgrammableWeb. 2016. APIs, Mashups and the Web as Platform. [Verkkosivusto]. ProgrammableWeb. [Viitattu 22.5.2016]. Saatavana: <http://www.programmableweb.com/>
- RFC 3986. 2005. Uniform Resource Identifier (URI): Generic Syntax. [Verkkosivu]. Network Working Group. [Viitattu 25.4.2016]. Saatavana: <http://www.ietf.org/rfc/rfc3986.txt>
- RFC 7230. 2014. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. [Verkkosivu]. Internet Engineering Task Force. [Viitattu 13.4.2016]. Saatavana: <http://www.ietf.org/rfc/rfc7230.txt>
- RFC 7231. 2014. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. [Verkkosivu]. Internet Engineering Task Force. [Viitattu 13.4.2016]. Saatavana: <http://www.ietf.org/rfc/rfc7231.txt>

- Richardson, L. 2008. The Maturity Heuristic. [Verkkosivu]. [Viitattu 4.5.2016]. Saatavana: <https://www.crummy.com/writing/speaking/2008-QCon/act3.html>
- Richardson, L., Amundsen, M. & Ruby, S. 2013. RESTful Web APIs. Sebastopol, CA: O'Reilly Media, Inc.
- Richardson, L. & Ruby, S. 2007. RESTful Web Services. Sebastopol, CA: O'Reilly Media, Inc.
- Ronacher, A. 2016. Flask. [Verkkosivusto]. Poccoo Team. [Viitattu 25.5.2016]. Saatavana: <http://flask.pocoo.org/>
- Stowe, M. 2014. API Best Practices: Hypermedia (Part 1). [Verkkosivu]. MuleSoft, Inc. [Viitattu 11.5.2016]. Saatavana: <http://blogs.mulesoft.com/dev/api-dev/api-best-practices-hypermedia-part-1/>
- Webber, J., Parastatidis, S. & Robinson, I. 2010. REST in Practice. Sebastopol, CA: O'Reilly Media, Inc.



## **LIITTEET**

Liite 1. cURL-komennot REST-rajapinnan testaamiseen

**LIITE 1. cURL-komennot REST-rajapinnan testaamiseen**

```
# Haetaan kaikkien kirjojen tiedot.
curl http://localhost:5000/books

{
  "1": {
    "isbn": "978-1-4493-5806-8",
    "title": "RESTful Web APIs"
  },
  "2": {
    "isbn": "978-0-596-52926-0",
    "title": "RESTful Web Services"
  }
}

# Haetaan yhden kirjan tiedot.
curl http://localhost:5000/books/1

{
  "isbn": "978-1-4493-5806-8",
  "title": "RESTful Web APIs"
}

# Lisätään uusi kirja.
curl http://localhost:5000/books -X POST -H "Content-Type: application/json" -d '{"title":"a","isbn":"1"}'

{
  "isbn": "1",
  "title": "a"
}

# Päivitetään kirja.
curl http://localhost:5000/books/2 -X PUT -H "Content-Type: application/json" -d '{"title":"b","isbn":"2"}'

{
  "isbn": "2",
  "title": "b"
}

# Poistetaan kirja.
curl http://localhost:5000/books/1 -X DELETE
```