Tampereen yliopisto

Onni Aho

VERKKO-OHJELMOINTIRAJAPINTOJEN VERTAILU

SOAP, REST ja GraphQL

Kandidaatintyö
Informaatioteknologian ja viestinnän tiedekunta
Tarkastaja: Joonas Multanen
Elokuu 2024

TIIVISTELMÄ

Onni Aho: Verkko-ohjelmointirajapintojen vertailu: SOAP, REST ja GraphQL
Kandidaatintyö
Tampereen yliopisto
Tietotekniikka
Elokuu 2024

Verkko-ohjelmointirajapinnat ovat keskeinen osa nykypäivän sovelluskehitystä, ja vaihtoehtoja niiden toteutukseen löytyy paljon. On tärkeää valita tiettyyn projektiin sopivin API, joka täyttää halutut vaatimukset, ja toteuttaa parhaiten halutut ominaisuudet. Tässä työssä vertaillaan kolmea suosituimpaa Web API:a, jotka ovat REST, SOAP ja GraphQL. Vertailun perusteella pyritään selvittämään kunkin rajapinnan vahvuudet ja heikkoudet. Vertailukohtina käytetään suorituskykyä, ominaisuuksia, käytettävyyttä ja turvallisuutta.

Vertailussa havaitaan, että GraphQL on useissa testeissä tehokkain vaihtoehto tarkasteltavista rajapinnoista, mutta suuria datamääriä haettaessa REST on tehokkaampi. GraphQL tarjoaa joustavuutta kyselyiden muodostamiseen, jonka avulla saadaan haettua tietoa tarkasti pienemmillä pakettikoolla. REST- ja SOAP-kyselyt suoritetaan HTTP:n peruskomennoilla, jonka takia tietoa haetaan todennäköisesti liikaa tai liian vähän.

Käytettävyydessä SOAP on todennäköisesti heikoin vaihtoehto, sillä sen kehittäminen on työlästä ja monimutkaista. Se on riippuvainen XML-formaatista, jonka takia se ei ole yhtä joustava kuin REST tai GraphQL. Käytettävyyttä vertailtaessa havaitaan kuitenkin, että GraphQL-kyselyiden toteuttaminen on nopeampaa ja helpompaa verrattuna REST:iin. Turvallisuuden ja yksityisyyden toteuttamisessa SOAP:ia pidetään kuitenkin yleisesti parhaimpana vaihtoehtona, sillä se tarjoaa ylimääräisiä salausmenetelmiä API-yhteyteen.

Avainsanat: API, SOAP, REST, GraphQL

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

API	Application Programming Interface
JSON	JavaScript Object Notation
XML	Extensible Markup Language
HTTP	Hypertext Transfer Protocol
SOAP	Simple Objects Access Protocol
REST	Representational State Transfer
GraphQL	Graphical Query Language
TCP	Transmission Control Protocol
IP	Internet Protocol
DNS	Domain Name Service

.

1. JOHDANTO

Ohjelmointirajapinnat eli API:t (Application Programming Interface) ovat ohjelmistojen kehityksessä käytettäviä rajapintoja, jotka mahdollistavat eri ohjelmistojen ja komponenttien välisen kanssakäymisen. Näihin rajapintoihin kuuluvat verkko-ohjelmointirajapinnat, jotka ovat internetin välityksellä keskenään kommunikoivia rajapintoja. Ne ovat suuressa osassa nykypäivän ohjelmistonkehitystä, sekä pienissä, että suuremmissa ohjelmistoissa. [1]

Oikean verkko-ohjelmointirajapinnan valinta on tärkeä osa ohjelmistoprojektien suunnittelua. Vaihtoehtoja on useita, joista jokaisella erilaiset ominaisuudet, vahvuudet ja heikkoudet. Eri rajapinnat tarjoavat parempia ominaisuuksia tiedon käsittelyyn ja kyselyiden suorittamiseen. On myös tärkeää, että rajapintojen välinen yhteys on turvallinen varsinkin arkaluontoista dataa käsitellessä. Rajapinnan nopeus vaikuttaa koko sovelluksen suorituskykyyn ja käytettävyyteen, varsinkin suuria tietomääriä käsitellessä.

Tämä työ keskittyy verkko-ohjelmointirajapintojen tutkimiseen ja vertailuun mahdollisimman käytännöllisestä näkökulmasta eri kokoisissa ohjelmistoprojekteissa. Vertailukohdat ovat valittu kehitettävien ohjelmistojen tärkeimpien osa-alueiden ja vaatimusten mukaisesti, jotta tuloksia voitaisiin soveltaa mahdollisimman hyvin käytännöllisiin tarkoituksiin. Tähän työhön on valittu REST (Representational State Transfer)-, SOAP (Simple Objects Access Protocol)-, ja GraphQL (Graphical Query Language)-rajapinnat, sillä ne ovat nykypäivän käytetyimmät ja tutkituimmat. Muitakin rajapintoja on olemassa, mutta niitä ei vertailla tässä työssä, sillä niiden käyttö on suhteellisen harvinaista. Vertailu keskittyy suorituskyvyn, ominaisuuksien ja käytettävyyden vertailuun rajapintojen välillä. Näiden vertailukohtien avulla pyritään saavuttamaan mahdollisimman laaja käytännönläheinen näkemys valittujen Web API-rajapintojen eduista ja heikkouksista erilaisiin ohjelmistoprojekteihin sovellettuna.

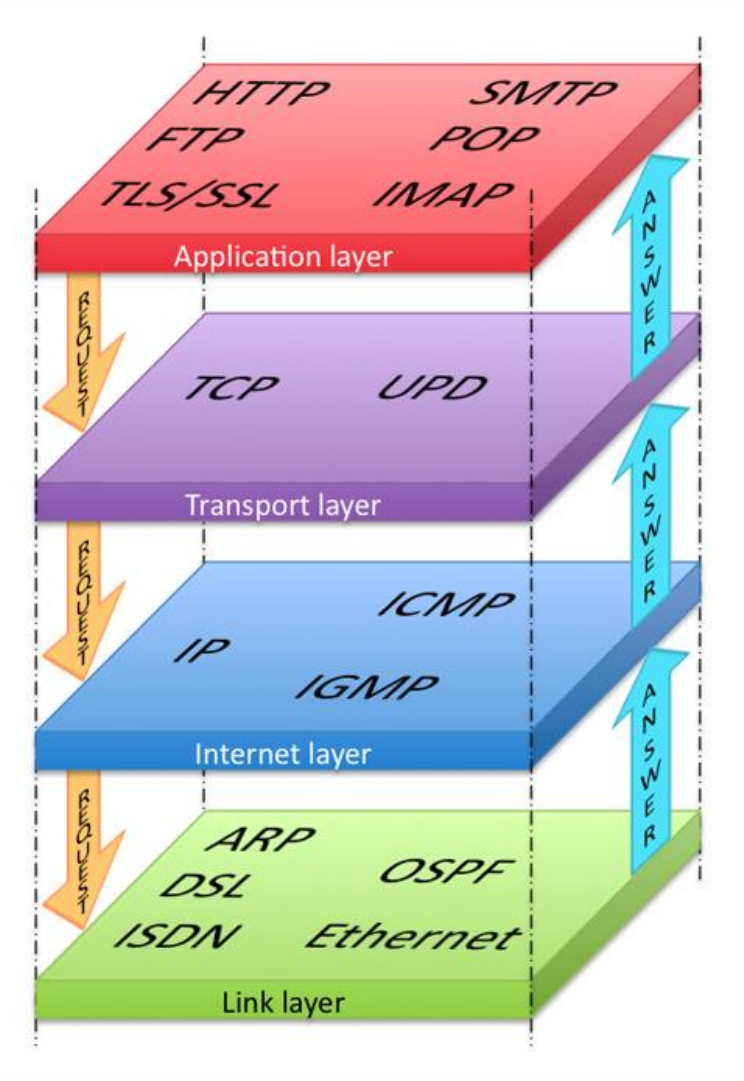
2. VERKKO-OHJELMOINTIRAJAPINTOJEN TOIMINTA

Internetin välityksellä toimivat ohjelmointirajapinnat, eli Web API:t, ovat kriittinen osa monia nykypäivän ohjelmistoprojekteja. Ne mahdollistavat eri ohjelmistojen ja tietokantojen välisen kanssakäymisen ja informaation vaihdon. Tämän kanssakäymisen avulla kehitettävät sovellukset kykenevät hyödyntämään valtavaa määrää informaatiota eri ohjelmistojen ja tietokantojen tarjoamien rajapintojen avulla.

Web API:t toimivat URL:ien (Unified Resource Locators) avulla. URL sisältää hyödynnettävän rajapinnan tietokannan verkko-osoitteen, sekä haettavien resurssien uniikin tunnisteen. Web API:n käyttö vaatii yleensä myös jonkinlaisen tunnistautumisen yhteyden turvallisuuden suojaamiseksi. Yleisimpiä tunnistautumistapoja ovat HTTP-tunnistautuminen, API avaimet ja OAuth. HTTP-tunnistautuminen turvautuu käyttäjäkohtaiseen käyttäjänimi- ja salasanapohjaiseen tunnistautumiseen, kun taas API avain on URL:ään sisällytetty satunnaisesti generoitu arvo, joka luodaan ensimmäisen rajapinnan kanssakäymisen yhteydessä. OAuth on monimutkaisempi versio API avaimesta, ja mahdollistaa yleisesti turvallisimman yhteyden rajapinnan kanssakäymiseen. [3]

2.1 Tietoliikenneprotokolla

Kehitettävän sovelluksen API kanssakäyminen perustuu pohjimmiltaan URL:ien lisäksi HTTP:n (Hypertext Transfer Protocol) peruskomentoihin, *GET, POST* ja *DELETE.* Nämä komennot toteuttavat toiminnallisuuden API:n resurssien käytössä. Ennen kyselyiden toteuttamista, sovelluksen ja tietokannan välille muodostetaan TCP (Transmission Control Protocol) yhteys, joka mahdollistaa virheettömän ja eheän kommunikaation. Sovellus hakee DNS (Domain Name Service) palvelimelta kutsuttavan tietokannan IP (Internet Protocol) osoitteen, jonka jälkeen TCP yhteys pystytään muodostamaan. Tämän jälkeen sovellus pystyy suorittamaan HTTP-kutsuja tietokannan palvelimille. Jos yhteydelle halutaan toteuttaa lisää turvallisuutta, voidaan käyttää HTTPS:ää, joka mahdollistaa HTTP-yhteyden päälle lisäturvallisuustasojen toteuttamista. [8] Kuvassa 1 on havainnollistettu TCP/IP-pinoa. Siinä näkyy pinon eri kerrokset, ja niiden välinen kommunikaatio molempiin suuntiin.



Kuva 1: TCP/IP-pino [12]

2.2 Kyselyiden tiedostomuodot

HTTP-kyselyillä haettava data voidaan palauttaa sovellukselle monissa eri muodoissa. Nykyään API:en kanssa käytetyimmät muodot ovat XML (Extensible Markup Language) ja JSON (JavaScript Object Notation) formaatit. Muita formaatteja kuten SGML (Standard Generalized Markup Language) ei enää nykyään käytetä, sillä ne ovat tyypillisesti monimutkaisempia ja vaikeampilukuisia kuin JSON ja XML. [2]

JSON on JavaScriptin pohjalta kehitetty formaatti, jota monet API-pohjaiset sovellukset hyödyntävät. Se on kevyt tiedostomuoto, joka on kehitetty ihmisille luettavan, sekä so-

velluksille mahdollisimman helposti parsittavaan ja kirjoitettavaan muotoon. Yksinkertainen JSON-tiedosto, jossa on kaksi objektia, ja molemmilla yksi attribuutti, kirjoitetaan muodossa

```
{
      ”objekti1”: {
            ”nimi”:”objekti1_nimi”
            ”objekti2”: {
                  ”nimi”:”objekti2_nimi”
            },
}.
```

JSON-notaatio perustuu avain/arvo-pareihin, joiden väliin tulee kaksoispiste. Parit erotetaan toisistaan pilkun avulla, ja objektit rajataan aaltosulkujen sisään. Suurin osa yleisimmistä ohjelmointikielistä sisältävät toiminnallisuuksia JSON-tiedostojen luomiselle ja käsittelylle.

XML on toinen yleisesti käytetty API-kyselyiden tiedostomuoto. Se on kehitetty SGML:n pohjalta yksinkertaistamaan ja helpottamaan SGML:n luettavuutta ja käytettävyyttä. XML on laajalti käytetty, sillä lähes kaikki ohjelmointikielet tukevat sitä, ja sen formaatti on hyvin selkeä. JSON esimerkkiä vastaava XML tiedosto olisi muotoa

```
<objekti1 nimi=”objekti1_nimi”>
      <objekti2>
            <nimi> objekti2_nimi </nimi>
      </objekti2>
</objekti1>.
```

XML-notaatio rakentuu elementeistä ja attribuuteista. Elementit jakavat tekstin sisällön osiin alku- ja loppumerkinnöillä, joiden sisälle voi tulla attribuutteja tai uusia elementtejä. Attribuutit ovat avain/arvo-pareja, jotka sisältävät tiedoston pääosaisen datan.

3. VERTAILTAVAT VERKKO-OHJELMOINTIRAJAPINNAT

Ohjelmointirajapintojen toiminta ja käyttötarkoitukset verkkopohjaisissa sovelluksissa on suhteellisen yksiselitteistä, mutta rajapintojen toteuttamiseen löytyy useita erilaisia ratkaisuja. Ohjelmointirajapinnan käyttö vaatii jonkinlaisen arkkitehtuurin taustalleen, jotta sitä voidaan hyödyntää halutulla tavalla kehitettävässä ohjelmistossa. Tunnetuimpia ohjelmointirajapintojen toteutuksia nykyään ovat SOAP, REST ja GraphQL. Näitä kolmea rajapintaa tarkastellaan tässä luvussa paremmin.

Muita verkko-ohjelmointirajapintoja ovat esimerkiksi JSON ja XML tiedostoihin sisäänrakennetut RPC (Remote Procedure Call) protokollat, jotka mahdollistavat hyvin yksinkertaisen rajapintojen välisen kommunikaation. [9] Falcor ja WebSocket ovat myös vaihtoehtoisia uudempia verkko-ohjelmointirajapintoja, jotka tarjoavat erilaisia ratkaisuja muiden rajapintojen heikkouksiin.

3.1 SOAP

Ensimmäinen laajalti käytetty API-arkkitehtuuri on SOAP. Se on protokolla, jonka kehittivät Mentor, User Land Company, sekä Microsoft vuonna 1998, ja julkaisivat vuonna 1999 [5]. SOAP hyödyntää HTTP-pohjaista tiedonsiirtoa XML-tiedostojen avulla. Sitä pidetään yleisesti nykyään hieman vanhanaikaisena teknologiana verrattuna muihin käytetyimpiin API-ratkaisuihin. [2]

Vaikka SOAP ei ole nykyään enää käytetyin teknologia API:den toteutukseen sovelluksissa, on sillä myös hyviä puolia. Se on ohjelmistoille kevytkäyttöinen teknologia, ja sen toteutus on alustariippumaton. SOAP:ia voidaan siis hyödyntää eri ohjelmointikielillä, ja useiden eri alustojen avulla kehitettyjen sovellusten välillä. Sen lisäksi SOAP:in tiukemman protokollamäärittelyn ja suojausten takia, se on lähtökohtaisesti turvallisempi verrattuna muihin API-teknologioihin. [5]

SOAP:issa on kuitenkin monia yleisesti todettuja heikkouksia. Vaikka sen toteutus on kieli- ja alustariippumaton, SOAP-rajapinnan kommunikaatio pystyy hyödyntämään vain XML-tiedostoja. SOAP-rajapinnan toteuttaminen on yleisesti myös monimutkainen ja vaikea prosessi ohjelmistoja kehittäessä. [2]

3.2 REST

REST-arkkitehtuuri kehitettiin ratkaisemaan monia ongelmia, joita ohjelmistonkehittäjillä oli SOAP:in kanssa. Sen kehittämisen aloitti Roy Fielding vuonna 2000. REST on joukko ohjelmistoarkkitehtuuriperiaatteita, jotka mahdollistavat tiedonsiirron eri ohjelmistojen välillä internetin välityksellä. Se ei siis ole standardi pohjainen protokolla kuten SOAP. Tämä mahdollistaa yleisesti joustavamman API-toteutuksen. [2]

REST on nykyään suosituin API-teknologia [1], ja sillä onkin monia hyviä puolia verrattuna muihin toteutustapoihin. REST tarjoaa joustavammat mahdollisuudet API:n toteuttamiseen, eikä se ole rajoitettu mihinkään vastausmuotoihin. Yleisesti REST API:a käytetään kuitenkin HTTP:n avulla XML tai JSON muotoja hyödyntäen. REST-rajapinnan toteuttaminen on myös yksinkertaisempaa ja helpompaa kuin SOAP-rajapinnan toteutus. [2]

3.3 GraphQL

GraphQL on Facebookin kehittämä API-teknologia, joka julkaistiin avoimeen käyttöön vuonna 2015. Se on siis huomattavasti uudempi kuin SOAP ja REST. GraphQL:n suurin ero muihin rajapintoihin verrattuna on se, että se on kyselypohjainen ohjelmointikieli. Sen lähestymistapa ohjelmistojen väliseen kanssakäymiseen on myös erilainen, ja se tarjoaakin monia ainutlaatuisia mahdollisuuksia API:en toteuttamiseen. [2]

SOAP- ja REST-pohjaisten rajapintojen yhtenä ongelmana on tiedon suodattaminen. Kyselyitä toteuttava ohjelma ei pysty tarkkaan määrittämään, mitä dataa rajapinnan avulla pyritään hakemaan. Tämän seurauksena suuri määrä tietoa saatetaan siirtää turhaan. GraphQL tarjoaa mahdollisuuden tiedonhaun rajaamiseen kyselyitä suorittaessa.

4. VERTAILU

Kuten edellisessä luvussa lyhyesti kerrottiin, on Web API:en välillä useita eroavaisuuksia. Toiset teknologiat tarjoavat hyviä ratkaisua tiettyihin sovellutuksiin ja ympäristöihin, mutta saattavat olla heikompia toisilla osa-alueilla. Ohjelmistojen kehityksessä onkin tärkeää valita rajapinta, joka soveltuu parhaiten kehitettävän ohjelmiston, ja sen kehittäjien tarpeisiin.

Tätä tutkielmaa varten vertailukohdiksi on valikoitunut suorituskyky, ominaisuudet, turvallisuus ja käytettävyys. Nämä vertailukohdat ovat projektiluontoisessa ohjelmistonkehityksessä tärkeimmät yleisen tason kriteerit, joilla rajapintoja voidaan vertailla. Tämän vertailun avulla pyritään selvittämään millaisiin ohjelmistoprojekteihin mikäkin rajapinta soveltuisi parhaiten, sillä eri kriteerien painoarvo on hyvin tapauskohtaista.

4.1 Suorituskyky

Tämän tutkielman rajapintojen välinen suorituskyvyn vertailu perustuu Oggierin [4], sekä Pontus ja Remeksen [2] toteuttamiin ohjelmointirajapintoja vertaileviin tehokkuustesteihin. Oggierin työssä vertailtiin REST- ja GraphQL-rajapintojen suorituskykyjä standardoitujen testien avulla. Työtä varten kehitettiin testiympäristö, jonka avulla kyettiin suorittamaan täsmälleen sama API-kysely molemmilla rajapinnoilla. Testien mittareina käytettiin suoritusaikaa, sekä haettujen datapakettien kokoja.

Erlandssonin ja Remeksen toteuttama tutkimus vertaili REST-, SOAP- ja GraphQL-rajapintojen suorituskykyjä mahdollisimman neutraalissa testiympäristössä. Kuten Oggierin työssä, Remeksen ja Erlandssonin testimittareina käytettiin API-kyselyiden suoritusaikaa, sekä pakettikokoa. Testit koostuivat erilaisista kyselyistä, joista kaikki suoritettiin eri rajapintojen avulla mahdollisimman neutraalissa paikallisverkon testiympäristössä.

Molemmissa töissä mitattiin samoja arvoja, sekä testiympäristöt olivat hyvin samankaltaiset. Kyselyiden toteutukset, ja saadut tulokset eroavat tutkimusten välillä kuitenkin huomattavasti. Molemmista tutkimuksista tärkeimmät tulokset on kerätty tiivistettynä taulukoihin 1 ja 2.

Taulukko 1. *Oggierin tutkimustulosten keskiarvot* [4]

Testitoteutus	GraphQL nopeus (ms)	REST nopeus (ms)	GraphQL koko (KB)	REST koko (KB)
Testi A ka.	187	346	4,91	6,18
Testi B ka.	165	253	1,37	4,74

Taulukon 1 perusteella on havaittavissa, että Oggierin testeissä GraphQL voittaa REST:in sekä nopeudessa, että pakettien koossa. Ensimmäisessä testissä molemmilla API:lla toteutettiin sarja yksinkertaisia kyselyitä, jossa haettiin tietokannasta useita objekteja ja attribuutteja. Tulosten perusteella GraphQL suoriutti testin 46 % nopeammin ja 21 % pienemmillä pakettiko'oilla. Toisessa testissä REST-kysely suoritettiin rinnakkaisilla kyselyillä, kun taas GraphQL tarvitsi tiedon hakemiseen vain yksittäisen kyselyn. Tästä testistä GraphQL suoriutui 35 % nopeammin, ja 71 % pienemmillä pakettiko'oilla. [4]

Oggierin tutkimus demostroi hyvin GraphQL:n vahvuudet REST:iin verrattuna. Testit ovat kuitenkin rajallisesti toteutettu eivätkä välttämättä vastaa kaikkien sovellusten toteuttamien API-kutsujen tarpeita. Toteutettu testiympäristö suosii vahvemmin GraphQL-kyselyiden toteutusperiaatteita, mutta erilaisessa sovellusympäristössä tutkimuksen tulokset eivät välttämättä päde. Lisäksi suhteellisen pienet pakettikoot suosivat GrapQL:ää.

Taulukko 2. *Erlandssonin ja Remeksen tutkimustulosten keskiarvot* [2]

Testito-teutus	GraphQL nopeus (ms)	REST nopeus (ms)	SOAP nopeus (ms)	GraphQL koko (KB)	REST koko (KB)	SOAP koko (KB)
testien 1–6 ka.	58	33	35	24,46	88,87	121,71
testien 7–12 ka.	63	33	35	67,60	103,33	141,60
testien 13–18 ka.	71	33	34	103,33	103,33	141,60

Erlandssonin ja Remeksen tutkimuksessa [2] toteutettiin 18 eri testiä, joiden tulokset on kerätty tiivistetyssä muodossa taulukkoon 2. Testien edetessä kyselyissä kasvatettiin haettavien sarakkeiden ja rivien määrää. Tuloksista havaitaan, että sarakkeiden määrän kasvaessa, GraphQL hidastuu, kun REST:in ja SOAP:in suoritusnopeudet pysyvät suhteellisen samoina. Pienemmissä kyselyissä GraphQL on huomattavasti kevyempi, mutta haettavien sarakkeiden määrän lisääntyessä GraphQL on keskimäärin yhtä kevyt kuin REST. SOAP on kaikissa testeissä raskain vaihtoehto, mutta on keskimäärin nopeudeltaan vain hieman REST:iä hitaampi.

REST:in nopeus Erlandssonin ja Remeksen tutkimuksessa selittyy todennäköisesti sen arkkitehtuurin rakenteellisilla eduilla. Kaikki sen mekanismit on rakennettu mahdollisimman yksinkertaisesti hyödyntäen CRUD-operaatioita (Create, Read, Update, Delete), jotka kääntyvät suoraan HTTP-peruskomennoiksi. Lisäksi REST ei tarvitse palvelinpuolen istuntojen ylläpitoa, joka vähentää kuormitusta palvelimella.

Taulukoita ja tutkimusten tuloksia vertaillessa huomataan, että tulokset eroavat toisistaan suuresti. Oggierin testeissä GraphQL on selkeästi parempi vaihtoehto, mutta Erlandssonin ja Remeksen tuloksien perusteella REST on pääasiallisesti tehokkain. Oggierin testeissä on käytetty paljon pienempiä datamääriä ja yksinkertaisempia kyselyitä, jotka ovat todennäköisesti GraphQL:n eduksi. Erlandssonin ja Remeksen tutkimuksessakin huomataan, että GraphQL:n suorituskyky heikkenee kyselyiden suurentuessa. Voidaan siis päätellä, että GraphQL on tehokkain vaihtoehto, kun halutaan suorittaa kevyempiä ja spesifimpiä kyselyitä. Kun dataa haetaan kyselyillä suurempia määriä, on REST todennäköisesti tehokkain valinta.

4.2 Ominaisuudet

SOAP, REST, ja GraphQL ovat arkkitehtuuriltaan hyvin erilaisia, ja ne on kehitetty keskenään erilaisten käyttötarkoitusten helpottamiseksi. Tämän vuoksi niiden ominaisuuksissa ja käyttöedellytyksissä on monia eroavaisuuksia. Näiden ominaisuuksien ja käyttöedellytyksien kriittisyys vaihtelee kuitenkin projektikohtaisesti, eikä niiden painoarvo välttämättä ole joka tilanteessa sama.

API-kyselyt hakevat tietokannasta dataa tietyillä formaateilla, joista yleisimmät ovat JSON ja XML. SOAP on kuitenkin rajoitettu käyttämään pelkästään XML formaattia HTTP:n avulla. REST- ja GraphQL-rajapinnat ovat tältä osin monipuolisempia, sillä ne eivät rajoitu mihinkään kyselyformaattiin. Ne eivät myöskään ole rajoitettu pelkkään HTTP-pohjaiseen kommunikaatioon, vaikka se onkin Web-ohjelmistoissa yleisimmin käytetty tekniikka. [2]

Vaikka REST on kyselyiden tekemiseen SOAP:ia joustavampi, tarjoaa GraphQL vielä lisätasoja kyselyiden joustavuuteen. REST- ja SOAP-kyselyt perustuvat ennalta määrättyihin parametreihin ja *endpointteihin*, jotka määrittävät mitä dataa rajapinnan avulla palautetaan. Tämä johtaa siihen, että kyselystä saadaan paljon dataa, jolle ei ole ohjelmassa välttämättä käyttöä. GraphQL-kyselyt eivät ole riippuvaisia näistä *endpointeista*, jonka ansiosta haettavan datan määrä ja formaatti voidaan määritellä tarkemmin. [6]

Esimerkkikysely, jonka avulla voitaisiin hakea säädataa Tampereelta olisi SOAP:in avulla muotoa

```
<soapenv:Envelope   xmlns:soapenv=http://schemas.xmlsoap.org/soap/envelope/
xmlns:wea="http://www.example.com/weather">
   <soapenv:Header/>
   <soapenv:Body>
      <wea:GetWeather>
         <wea:City>Tampere</wea:City>
      </wea:GetWeather>
   </soapenv:Body>
</soapenv:Envelope>.
```

SOAP kyselyt perustuvat XML formaattiin, ja ne sisältävät usein paljon tekstiä verrattuna esimerkiksi vastaavaan REST kyselyyn

```
GET /weather?city=Tampere HTTP/1.1
Host: api.example.com.
```

REST kyselyssä on kaikki sama toiminnallisuus, kuin SOAP kyselyssä, mutta se perustuu HTTP:n peruskomentoihin, jonka avulla kyselyt ovat tiiviimpiä ja helpommin luettavissa. Vastaava kysely suoritettuna GraphQL:n avulla olisi muotoa

```
query {
  weather(city: "Tampere") {
    temperature
    description
    humidity
    windSpeed
  }
}.
```

Kyselyssä itsessään määritellään haettava säädata, eikä tiedon karsiminen jää muun ohjelman vastuulle. Kysely on myös helposti luettavassa muodossa, eikä sen parametreja ole vaikea muokata.

API:t saattavat tarvita ohjelmistonkehityksessä joitakin ulkoisia kirjastoja ja työkaluja toimiakseen valitussa ympäristössä. Kaikki käytetyimmät ohjelmointikielet kuten Java, JavaScript, Python ja PHP sisältävät SOAP-, REST- ja GraphQL-apukirjastoja, jotka mahdollistavat niiden käytön. Sillä REST on tällä hetkellä laajimmin käytetty API-teknologia,

löytyy siihen myös eniten kirjastoja ja työkaluja, jotka laajentavat sen käytön mahdollisuuksia ja käytettävyyttä.

4.3 Käytettävyys

Ohjelmointirajapintojen valitsemisessa voi olla monia preferenssejä ja vanhoja tottumuksia, joiden perusteella mikä tahansa API voi olla yhdelle kehittäjälle käytettävämpi kuin toiselle. Yleisemmällä tasolla SOAP:ia pidetään kuitenkin vaikeampikäyttöisenä REST- ja GraphQL-rajapintoihin verrattuna [2]. Sen XML-pohjainen formaatti ja jäykät raamit voivat vaikeuttaa kehittämisprosessia [10]. REST:in ja GraphQL:än joustavuus ja laajempi HTTP tuki lähtökohtaisesti helpottavat API:n kehittämisprosessia.

Tutkimuksessa [6] Gleison Brito ja Marco Tulio Valente vertailivat suoraan REST- ja GraphQL-rajapintojen kehittämisen prosesseja. Tutkimus toteutettiin 22 opiskelijan ryhmälle, jotka suorittivat useita kyselyitä sekä REST:in, että GraphQL:n avulla. Siinä havaittiin, että kyselyiden toteuttaminen GraphQL:n avulla oli kaikissa tapauksissa huomattavasti nopeampaa, ja monimutkaisten kyselyiden kanssa erot olivat vielä suuremmat. Keskimäärin kyselyiden toteuttaminen kesti REST:in avulla 9 minuuttia, ja 6 minuuttia GraphQL:n avulla. Tutkimusjoukosta kenelläkään ei ollut aikaisempaa käyttökokemusta GraphQL:stä, mutta osalla oli kokemusta REST:in kanssa työskentelystä. Vaikka tutkimuksen testit ovat rajallisesti toteutettu, antaa se silti vahvan viitteen siihen suuntaan, että GraphQL on helpompikäyttöinen teknologia varsinkin kokemattomammille kehittäjille. [6]

4.4 Turvallisuus

Web API:t hyödyntävät usein ulkoisia tietokantoja, jotka jakavat dataa ulkoiseen käyttöön jonkinlaisen rajapinnan avulla. Tämän takia rajapintoja hyödyntäviä sovelluksia kehittäessä, on turvallisuus ja yksityisyys tärkeä ottaa huomioon. Se ei kuitenkaan ole pelkästään sovellusta kehittävien tahojen vastuulla, vaan myös API-tietokantoja ylläpitävien tahojen vastuulla. Turvallisen API-yhteyden tärkeimpiä tekijöitä ovat yhteyden salaus, eheys, tarkkailu ja autentikointi [11].

Kaikki tässä työssä tarkasteltavat API-teknologiat tukevat suosituimpia autentikointimenetelmiä kuten API avain, OAuth ja OpenID [7]. Tunnistautumismenetelmän valitseminen on tärkeää, sillä sen avulla pystytään rajoittamaan ja tarkkailemaan yhteyttä hyödyntäviä tahoja.

Yhteyden salaamisen ja eheyden varmistaminen on tärkeä osa API-yhteyden luomista, varsinkin arkaluontoisia ja yksityisiä asioita käsitellessä. REST ja GraphQL tukevat standardin HTTP:n lisäksi turvallisempaa HTTPS protokollaa, joka tarjoaa lisätasoja yhteyden salaukseen. Vaikka SOAP ei varsinaisesti tue HTTPS protokollaa, se sisältää sisäänrakennettuja turvallisuus teknologioita kuten WS-Security protokollan. Tämän takia SOAP:ia pidetään turvallisimpana API vaihtoehtona, jonka takia sitä käytetään monien rahoitus- ja kommunikaatioyritysten ohjelmistoissa. [5]

4.5 Vertailu kokonaisuutena

Edellisien lukujen vertailun perusteella saadaan muodostettua kokonaiskuva eri verkko-ohjelmointirajapintojen vahvuuksista ja heikkouksista. Vertailukohtien tärkeimmät kohdat on kerätty tiivistettynä taulukkoon 3.

Taulukko 3. *vertailun kokonaiskuva*

vertailukohde	REST	SOAP	GraphQL
suorituskyky	Tehokkain erityisesti suuria datamääriä käsitellessä.	Lähes kaikissa tilanteissa heikoin tehokkuus.	Tehokkain pienien datamäärien ja tarkkojen kyselyiden suorittamisessa.
ominaisuudet	Tukee monia eri teknologioita kyselyiden suorittamiseen ja tarjoaa joustavuutta rajapinnan toteutuksessa esimerkiksi laajojen ulkoisten kirjastojen avulla.	Jäykkä toteutus, joka on riippuvainen XML- ja HTTP-teknologioista.	Tukee monia eri teknologioita, ja tarjoaa joustavan datan hakemisen kyselyiden suorittamisessa. Ei kuitenkaan tarjoa yhtä paljon ulkoisten kirjastojen tukia kuin REST.
käytettävyys	Kaikista käytetyin verkko-ohjelmointirajapinta, jonka käytettävyyttä pidetään yleisesti suoraviivaisena ja joustavana.	XML-pohjainen formaatti ja vanhanaikainen toteutus on yleisesti vaikeakäyttöinen ja jäykkä.	Tutkimuksen [6] mukaan GraphQL on helpommin opeteltavissa varsinkin uusille kehittäjille kuin REST.
turvallisuus	Tukee HTTPS-protokollaa ja mahdollistaa monien ulkoisten turvallisuusteknologioiden käytön.	Tukee WS-Security protokollaa, jonka takia sitä pidetään yleisesti turvallisimpana vaihtoehtona	Tukee HTTPS-protokollaa ja mahdollistaa monien ulkoisten turvallisuusteknologioiden käytön.

5. YHTEENVETO

Web API pohjaisia ohjelmistoja kehittäessä on tärkeää, että valitaan käyttötarkoitukseen parhaiten sopiva API teknologia. Tässä työssä vertailtiin keskenään REST, SOAP ja GraphQL rajapintoja, ja niiden vahvuuksia ja heikkouksia. Keskeisinä vertailukohtina toimivat rajapintojen suorituskyky, ominaisuudet, käytettävyys ja turvallisuus.

Suorituskykyä vertaillessa havaittiin, että GraphQL oli pienikokoisia, ja spesifejä kyselyitä suorittaessa tehokkain vaihtoehto. Sen avulla pystytään rajaamaan haetun datan määrä, jonka takia se oli lähes kaikissa tilanteissa kevyin vaihtoehto. Suurempia datamääriä haettaessa REST oli kuitenkin nopein suorittamaan kyselyitä. SOAP on riippuvainen XML-formaatista, jonka seurauksena sen avulla suoritettujen kyselyiden suuruus ja nopeus hävisi REST:ille.

SOAP on ominaisuuksien ja käytettävyyden kannalta todennäköisesti heikoin vaihtoehto. GraphQL ja REST tarjoavat enemmän joustavuutta, ja tukevat monia uudempia teknologioita ja formaatteja kuten JSON ja HTTPS. REST on rajapinnoista laajimmin käytetty, ja sille on saatavilla monia apukirjastoja ja työkaluja, joita muille API:lle ei ole kehitetty. GraphQL tarjoaa kuitenkin joustavuutta ja helppoutta kyselyiden luomiseen, sillä haettavan datan määrä voidaan karsia tarkasti jo kyselyä suorittaessa. Tutkimuksen [6] avulla havaittiinkin, että kokemattomille kehittäjille GraphQL kyselyiden luominen oli huomattavasti tehokkaampaa, kuin REST kyselyiden luominen.

Ainoa vertailukohta, jossa SOAP:in havaittiin olevan REST:iä ja GraphQL:ää edistyksellisempi, on turvallisuus. Kaikki API:t tukevat monia tunnistautumiskeinoja, ja tarjoavat useita keinoja yhteyden salauksen ja eheyden varmistamiseen. SOAP:iin sisäänrakennettu WS-Security protokolla tarjoaa kuitenkin vahvempia turvallisuus- ja salausmenetelmiä verrattuna GraphQL ja REST rajapintoihin.

Jos turvallisuus on selkeästi tärkein osa kehitettävää ohjelmistoa, eikä helppokäyttöisyydellä tai ominaisuuksilla ole niin suurta painoarvoa, saattaa SOAP olla projektiin paras valinta. GraphQL tarjoaa joustavuutta, joka helpottaa sen käytettävyyttä ja tehokkuutta varsinkin tarkkoja ja monimutkaisia kyselyitä suorittaessa. Se on todennäköisesti hyvä valinta moniin projekteihin ja ympäristöihin. REST on kuitenkin vuosien saatossa levinnyt laajaan käyttöön, sillä se on tehokas ja tukee suurta määrää ominaisuuksia ja eri alustoja.

LÄHTEET

[1] Cooksey, Brian. "An introduction to APIs." Zapier (2014). (viitattu 5.8.2024) https://cdn.zapier.com/storage/learn_ebooks/e06a35cfcf092ec6dd22670383d9fd12.pdf

[2] Erlandsson, Pontus, and Joakim Remes. "Performance comparison: Between graphql, rest & soap." Bachelor's Degree Project. University of Skövde. Digitala Vetenskapliga Arkivet. (2020).

[3] Sandoval, K. (2018). 3 Common Methods of API Authentication Explained (viitattu 12.4.2024): https://nordicapis.com/3-common-methods-api-authentication-explained/

[4] Oggier, Camille. "How fast GraphQL is compared to REST APIs." Bachelor's Thesis. Haaga-Helia University of Applied Sciences. Theseus. (2020).

[5] Soni, Anshu, and Virender Ranga. "API features individualizing of web services: REST and SOAP." International Journal of Innovative Technology and Exploring Engineering 8.9 (2019): 664-671.

[6] Brito, Gleison, and Marco Tulio Valente. "REST vs GraphQL: A controlled experiment." 2020 IEEE international conference on software architecture (ICSA). IEEE, 2020.

[7] Kornienko, D. V., et al. "Principles of securing RESTful API web services developed with python frameworks." Journal of Physics: Conference Series. Vol. 2094. No. 3. IOP Publishing, 2021.

[8] Bermbach, D., & Wittern, E. (2016). Benchmarking web api quality. In Web Engineering: 16th International Conference, ICWE 2016, Lugano, Switzerland, June 6-9, 2016. Proceedings 16 (pp. 188-206). Springer International Publishing.

[9] Kiraly, S., & Szekely, S. (2018). "Analysing RPC and testing the performance of solutions." Informatica, 42(4).

[10] Wodehouse, C. "SOAP vs. REST: A look at two different api styles." (2018). (viitattu 4.5.2024): https://www.upwork.com/hiring/development/soap-vs-rest-comparing-twoapis/

[11] Siriwardena, Prabath. "Advanced API Security: OAuth and Beyond." Springer. (2020).

[12] Bughunter. "A graphic representation of the Internet Protocol Stack according to RFC 1122, with some of the important protocols visible". (2009). (viitattu 5.8.2024): https://commons.wikimedia.org/wiki/File:InternetProtocolStack.png