Suomi.fi
kehittäjille

RESTful-arkkitehtuurityylin ja GraphQL:n mukaiset ohjelmointirajapinnat

Liite 3 - Ohjelmointirajapintojen suunnittelu julkisessa hallinnossa

DTY/Honkanen Mika (DVV)

18.2.2026

Dokumentinhallinta	
Omistaja	Mika Honkanen
Laatinut	Mika Honkanen
Tarkastanut	
Hyväksynyt	

Version hallinta		
versionro	mitä tehty	pvm/henkilö
1	Ensimmäinen versio	18.2.2026 / Mh

DTY/Honkanen Mika (DVV)

18.2.2026

Sisällysluettelo

DTY/Honkanen Mika (DVV)

18.2.2026

RESTful-arkkitehtuurityylin ja GraphQL:n mukaiset ohjelmointirajapinnat

1.1 Johdanto

Tämä dokumentti täydentää Suomi.fi kehittäjille -sivustolla julkaistua **Ohjelmointirajapintojen suunnittelu julkisessa hallinnossa** -opasta.

Dokumentti keskittyy EU:n European Interoperability Frameworkin (EIF) mukaisiin semanttisen ja teknisen yhteentoimivuuden näkökulmiin. Lisäksi kannattaa huomioida kaksi muuta EIF:n tasoa: lakisääteinen ja organisatorinen yhteentoimivuus.

Lakisääteinen yhteentoimivuus tarkoittaa, että tiedonvaihto ja yhteistyö perustuvat yhteensopiviin lakeihin ja sopimuksiin, kuten EU:n yleisen tietosuoja-asetuksen noudattamiseen. Organisatorinen yhteentoimivuus varmistaa, että eri organisaatioiden prosessit, roolit ja vastuut ovat sovitettu yhteen, jotta palvelut toimivat saumattomasti. Lähtökohtana on, että lakisääteisen ja organisatorisen tason asiat on jo sovittu, suunniteltu ja huomioitu.

Ohjelmointirajapinnat ovat keskeinen osa digitaalisten palveluiden rakentamista ja yhteentoimivuutta. Julkishallinnossa niiden merkitys korostuu erityisesti silloin, kun palveluita kehitetään useiden organisaatioiden ja tietojärjestelmien yhteistyönä. Oikein toteutettuna ne mahdollistavat:

- **Tietojen automaattisen siirron tietojärjestelmien välillä**
- **Tietojärjestelmien uudelleenkäytön**
- **Näiden kahden avulla tuottavuushyödyt ja kustannussäästöt**

Tämän liitteen tavoitteena on tukea julkishallinnon kehittäjiä suunnittelemaan ja toteuttamaan **laadukkaita, yhteentoimivia ja ylläpidettäviä ohjelmointirajapintoja**. Dokumentti tarjoaa käytännönläheisen katsauksen kahteen keskeiseen lähestymistapaan:

- **RESTful-arkkitehtuurityyliin** ja sen tukemiseen yhteisellä tyylioppaalla sekä OpenAPI-määrittelyllä
- **GraphQL-teknologiaan**, joka voi täydentää RESTiä monimutkaisissa tietokyselyissä

Lisäksi käsitellään **API First -mallia**, joka korostaa ohjelmointirajapintojen suunnittelua ennen varsinaista ohjelmistokehitystä.

DTY/Honkanen Mika (DVV)

18.2.2026

Liitteen tavoitteena on tukea ohjelmointirajapintojen suunnittelua ja toteutusta julkishallinnossa siten, että ohjelmointirajapinnat ovat yhteentoimivia, ylläpidettäviä ja skaalautuvia. Liite tarjoaa käytännön ohjeita REST-arkkitehtuurityylin ja GraphQL-teknologian hyödyntämiseen ohjelmointirajapintojen suunnittelussa ja toteutuksessa. Lisäksi tavoitteena on edistää semanttista ja teknistä yhteentoimivuutta EU:n European Interoperability Frameworkin (EIF) mukaisesti.

On tärkeää muistaa, että EIF on työkalu saavuttaa joku tärkeä ja selkeää arvoa luova tavoite. Työ kannattaakin aloittaa ymmärtämällä liiketalouden näkökulmasta asia, mitä ollaan tekemässä. Ohjelmointirajapintojen avulla käytetään jo olemassa olevia resursseja uudestaan.

1.2 Vaihe 1. Aloita tunnistamalla asiakkaat ja liiketoiminnallinen tarve

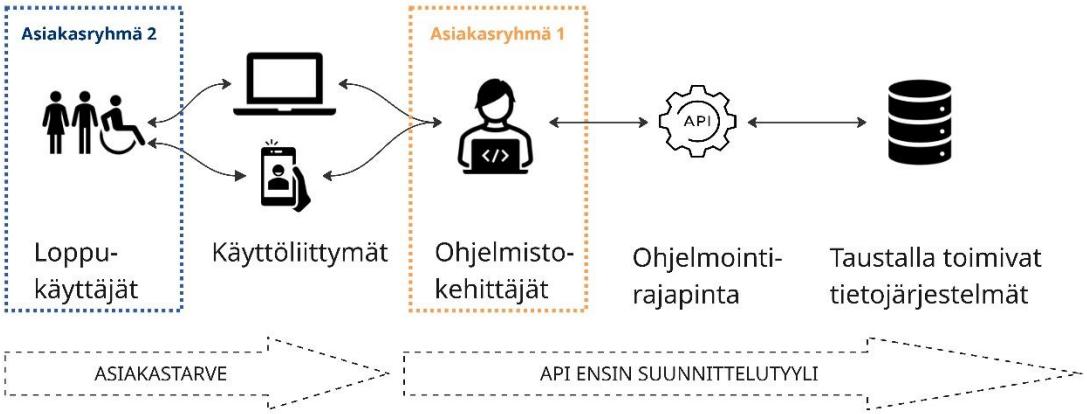
Ohjelmointirajapinnan suunnittelu kannattaa aloittaa taloudellisesta näkökulmasta: Mitä arvoa ohjelmointirajapinta luo, kenelle, miten ja miksi. Liiketaloudellisesta näkökulmasta ohjelmointirajapintojen voi ajatella olevan työvälineitä, joiden avulla käytetään uudestaan jo olemassa olevaa tietoa ja sen käsittelyyn hankittuja ohjelmistoja. Ohjelmointirajapintojen voi ajatella olevan ikkunoita tietoihin ja niitä käsittelevään tietojärjestelmään.

Ohjelmointirajapinnan suunnittelu alkaa sen tulevien hyödyntäjien tunnistamisesta ja arvon ymmärtämisestä. Ilman niitä suunnittelu perustuu satoihin arvaukseen siitä, miten ohjelmointirajapinnan pitäisi toimia. Asiakkuuksia voi tarkastella kahdella tasolla: suoria ohjelmointirajapinnan asiakkaita ovat ohjelmistokehittäjät (asiakasryhmä 1), ja heidän asiakkaitaan ovat usein virastojen, kuntien, hyvinvointialueiden sekä yrityksien käyttäjät (asiakasryhmä 2), joille ohjelmistokehittäjät luovat ratkaisuja. Työssä tulee huomioida molemmat asiakasryhmät. Liiketoiminnan näkökulmasta painottuu asiakasryhmä 2 (loppuasiakkaat / käyttäjät) ja teknologian näkökulmasta asiakasryhmä 1 (ohjelmistokehittäjät).

Ohjelmistokehittäjät (asiakasryhmä 1) käyttävät ohjelmointirajapintoja rakentaakseen sovelluksia, joita ihmiset (asiakasryhmä 2) hyödyntävät. Suunnittelu lähtee siis ihmisten tarpeista – taustalla toimivat tietojärjestelmät suunnitellaan sen perusteella (kuva 1).

DTY/Honkanen Mika (DVV)

18.2.2026



Kuva 1. Digitaalisen palvelun arvoketju.

Aiemmin palvelut on suunniteltu taustalla toimivista tietojärjestelmistä lähtien kohti loppukäyttäjiä, jolloin asiakkaiden tarpeet on välillä huomioitu heikosti. Ohjelmointirajapinnan suunnittelussa kannattaa ajatella koko digitaalista arvoketjua.

Kun suunnittelet ohjelmointirajapintaa, aloita hahmottamalla:

- **tiedon elinkaari**: missä vaiheissa tietoa syntyy, käsitellään ja hyödynnetään
- **toimijat**: ketkä tuottavat, käyttävät ja jakavat tietoa.

Ohjelmointirajapinnan suunnittelu kannattaa aloittaa hahmottamalla tiedon elinkaareen liittyvät toimijat. Ohjelmointirajapintaa kannattaa ajatella tuotteena, jolla on asiakkaat. Asiakkaita ovat ne toimijat, jotka hyödyntävät sitä. Ohjelmointirajapinnat suunnitellaan yleensä joko organisaation sisäisille tai ulkoisille asiakkaille. Usein ulkoisille asiakkaille suunniteltuja ohjelmointirajapintoja voivat hyödyntää myös sisäiset asiakkaat.

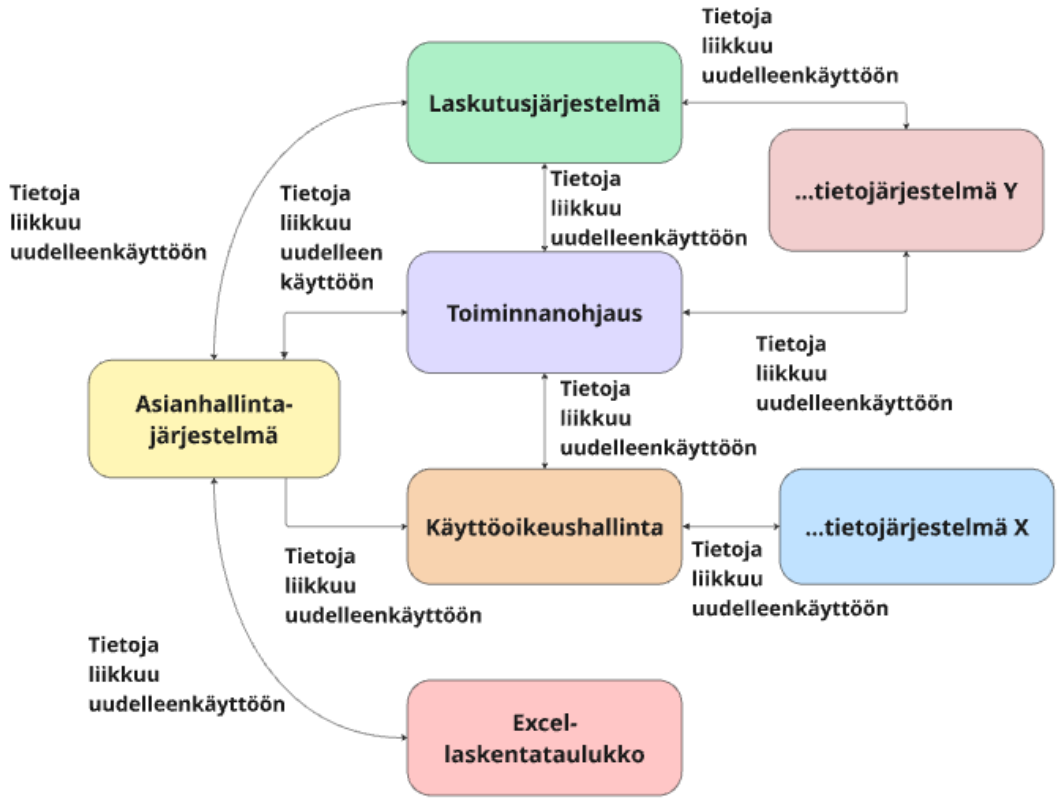
Sisäisille asiakkaille suunniteltu

Organisaation sisäisessä käytössä ohjelmointirajapintoja hyödyntävät saman organisaation muut sovellukset. Esimerkiksi asiakas- tai asianhallintajärjestelmän tietoja tarvitaan useissa tietojärjestelmissä (kuva 2). Ohjelmointirajapintojen avulla tiedot siirtyvät automaattisesti eri käyttötarkoituksiin hankittujen tietojärjestelmien välillä, jolloin työntekijöiden ei tarvitse syöttää niitä käsin eri tietojärjestelmiin – mikä on sekä kallista että altista virheille. Automaattinen tiedonsiirto parantaa tuottavuutta, vähentää päällekkäistä työtä ja inhimillisiä virheitä.

Ohjelmointirajapinnat kannattaa nähdä omiksi tuotteikseen, joilla on oma käyttäjäkuntansa. Julkisella sektorilla tietojärjestelmiä yhdistetään usein tietojärjestelmätasolla (kuva 2).

DTY/Honkanen Mika (DVV)

18.2.2026



Kuva 2. Organisaation sisällä ohjelmointirajapintojen avulla siirretään tietoa usean eri tietojärjestelmän välillä, jolloin työntekijöiden ei tarvitse syöttää eri käyttötarkoituksiin hankittuihin tietojärjestelmiin tietoja näppäimistöllä käsin uudestaan. Olennaista on tunnistaa tietojärjestelmät ennen ohjelmointirajapinnan suunnittelun aloittamista.

Suurissa organisaatioissa, kuten Helsingin kaupungilla, on käytössä satoja tietojärjestelmiä eri käyttötarkoituksiin (arviolta 700–900). Näissä sisäisten ohjelmointirajapintojen suunnittelu on keskeistä. Prosessi alkaa tunnistamalla sisäiset sidosryhmät – ne tietojärjestelmät ja niiden edustajat, jotka tarvitsevat ohjelmointirajapintaa – ja jatkaa suunnittelua tiiviissä yhteistyössä heidän kanssaan.

Sisäisiä ohjelmointirajapintoja hyödyntävät organisaation muut tietojärjestelmät tai saman tietojärjestelmän eri osat. Uudemmissa, pilvipohjaisissa tietojärjestelmissä ohjelmointirajapintoja on yleensä enemmän, ja niiden arkkitehtuuri muistuttaa mikropalvelulähtöistä lähestymismallia. Pilvipohjaiset tietojärjestelmät suunnitellaan yleensä ohjelmointirajapintalähtöisesti, mikä kasvattaa ohjelmointirajapintojen lukumäärää merkittävästi. Tämä arkkitehtuurimalli perustuu siihen, että tietojärjestelmät rakennetaan pienistä osista, jotka toimivat yhdessä ohjelmointirajapintojen kautta. Tällöin jo tehtyjä ohjelmiston osia on helpompi uudelleenkäyttää.

DTY/Honkanen Mika (DVV)

18.2.2026

Ulkoiset asiakkaat

Ulkoinen ohjelmointirajapinta suunnitellaan organisaation ulkoisille asiakkaille. Verohallinto on julkaissut noin 80 ohjelmointirajapintaa, joiden avulla se kerää veroehdotukseen sisältyviä tietoja automaattisesti. Ohjelmointirajapintojen ansiosta Verohallinto on siirtynyt 1980-luvulta lähtien paperilomakkeista automaattiseen veroilmoitukseen, mikä on merkittävästi parantanut tuottavuutta sekä asiakkaan että Verohallinnon näkökulmista. Asiakkaan ei enää tarvitse kerätä ja toimittaa paperisia kuitteja ja muita todistuksia Verohallintoon, vaan Verohallinto kerää nämä tiedot automaattisesti yrityksiltä ja pystyy tekemään yli 99,7 % yli 15 miljoonasta verotuspäätöksestä vuodessa täysin automaattisesti. Verohallinto lisännyt omaa tuottavuuttaan toimeenpanemalla tietoja kysytään vain kerran -käytännön tehokkaasti käytäntöön.

Tietoja kysytään vain kerran Once Only Principle (OOP):

EU on pyrkinyt nostamaan julkisen hallinnon työn tuottavuutta asettamalla tavoitteen *Once Only Principle (OOP)*: kansalaisilta ja yrityksiltä kysytään samat tiedot vain kerran. Suomessa tämä tarkoittaa, että viranomaiset hakevat jo kerätyt tiedot toisten viranomaisten rekistereistä, jos tietoluvan ehdot täyttyvät. Tiedon keräämistä useaan kertaan – esimerkiksi lomakkeilla – tulee välttää. Digitalisaation edelläkävijänä Verohallinto on huolellisesti toimeenpannut tämän periaatteen kaikissa prosesseissaan ja kerää automaattisen päätöksenteon tueksi laajasti ja automaattisesti tietoa myös yrityksiltä.

Once Only Principle (OOP) periaatteen toteuttaminen edellyttää:

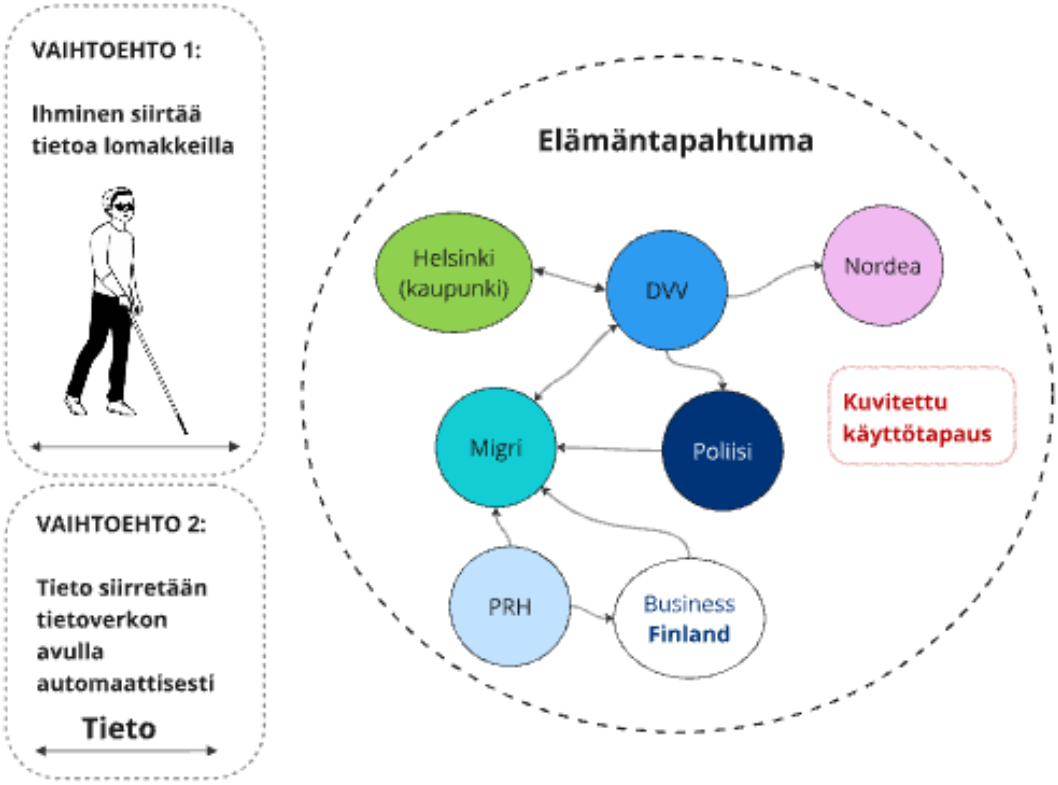
- tietojen tarkkaa ja yhteentoimivaa kuvausta
- tietoa siitä, mitä eri toimijat ovat jo keränneet
- omien tietotarpeiden ja prosessien ymmärtämistä
- luotettavaa ja turvallista tiedon jakamista.

Jos periaatetta tulkitsee laajimman tulkinnan mukaisesti, niin tämä tarkoittaisi, että tietoja kysytään vain kerran koko yhteiskunnassa – ja niitä jaettaisiin myös julkisen ja yksityisen sektorin välillä tietosuoja ja tietoturva huomioiden. Tietojen uudelleenkäytön lisäksi myös ohjelmistojen uudelleenkäyttö yleistyy, ja tulevaisuudessa velvoite voi laajentua myös ohjelmistoihin. Näiden avulla on mahdollista nostaa työn tuottavuutta ja tehokkuutta.

DTY/Honkanen Mika (DVV)

18.2.2026

Asiakkaan elämäntapahtuma voi edellyttää asiointia useissa organisaatioissa. Esimerkiksi ulkomaalaisen asiantuntijan muutto Suomeen voi vaatia yhteydenpitoa Poliisiin, Digi- ja väestötietovirastoon, pankkiin, Maahanmuuttovirastoon, Patentti- ja rekisterihallitukseen sekä Helsingin kaupunkiin (kuva 3). Nykyisin samaa tietoa siirretään usein paperilla tai fyysisesti jopa kuudessa eri organisaatiossa. Vielä paremmin automaattisen tiedonsiirron hyötyä sekä asiakkaalle että organisaatioille avaa käyttötapaus, jossa asiakas (vaihtoehto 1) on esimerkiksi sokea tai esimerkiksi muuten liikuntarajoitteinen.



Kuva 3. Palvelut, liike- ja elämäntapahtumat ylittävät usein organisaatioiden rajat, jolloin tuottavuutta voidaan lisätä tarkastelemalla kokonaisuutta. Kuvassa on kuvattu korkeakoulutetun erikoisosaajan Suomeen muuttoon liittyviä organisaatioita.

Tehokkuutta ja sujuvuutta saataisiin:

- jos tieto liikkuisi toimijoiden välillä digitaalisesti
- jos päällekkäisiä työvaiheita voitaisiin vähentää.

Tämä voi edellyttää:

- uudenlaista sääntelyä, erityisesti erityissäätelyn päivittämistä
- yhdessä suunniteltuja prosesseja
- avoimia tietomalleja

- ohjelmointirajapintoja, joiden avulla tieto ja tiedonkäsittely ovat yhteiskäyttöisiä.

Vaikka yksittäisen organisaation toiminta olisi optimoitu tuottavuuden näkökulmasta, ei se usein tarkoita, että elämä- tai liiketapahtuma kokonaisuudessaan on optimoitu. Kuvassa 3 esitetty esimerkki osoittaa, että eri viranomaiset voivat toimia tehokkaasti omassa organisaatiossaan, mutta asiakkaan näkökulmasta useita organisaatioita sisältävä elämäntapahtuma voi silti olla raskas ja sisältää paljon päällekkäistä työtä.

Julkinen hallinto on perinteisesti lähestynyt tietotekniikkaa organisaatiokohtaisesti:

- Kukin toimija on rakentanut kokonaisarkkitehtuurin omasta näkökulmastaan.
- Investoinnit on tehty pääsääntöisesti organisaation sisäisiin tarpeisiin.

Kokonaisuuteen saadaan paljon lisää työn tuottavuutta ja tehokkuutta, jos asioita tarkastellaan elämän- tai liiketapahtumien avulla, jotka ylittävät organisaatiorajat. Niissä tiedot liikuttavat sujuvasti eri toimijoiden välillä – vähintään julkisen hallinnon sisällä (valtio, hyvinvointialueet, kunnat) ja usein myös yksityisen sektorin kanssa.

Elämäntapahtuma tai liiketoimintatapahtuma ajattelu auttaa ohjelmointirajapintojen suunnittelussa. Kun olet hahmotellut joko organisaation sisäisesti tai laajemmin, ketkä kaikki toimijat liittyvät uuden ohjelmointirajapinnan suunnitteluun (ohjelmointirajapinnan asiakkaat), voit siirtyä suunnittelussa seuraavaan vaiheeseen.

1.3 Vaihe 2. Jäsennä kokonaisuus

Onnistunut ohjelmointirajapinnan suunnittelu vaatii kokonaisuuden hallintaa. Tätä ei kannata ulkoistaa, vaan se on organisaation ydintehtäviä. Mitä paremmin kokonaisuus ymmärretään ja hallitaan, sitä paremmin suunnittelu onnistuu.

Kokonaisuuden hallintaan on kehitetty vuosien saatossa useita erilaisia menetelmiä. Esimerkiksi kokonaisarkkitehtuuri on tapa kuvata tietotekniikan muodostamaa kokonaisuutta.

Kun asiakkaat tai heidän edustajansa on tunnistettu, tietomalli suunnitellaan kahdesta näkökulmasta: asiakkaiden käyttötapaukset ja organisaation tarjoamat resurssit (esim. tietovarannot ja ohjelmistot) sovitetaan yhteen. Näin varmistetaan, että ohjelmointirajapinta tukee todellisia tarpeita ja hyödyntää olemassa olevia tietoja ja palveluja tehokkaasti.

1.4 Vaihe 3. Suunnittele ja valitse semanttinen yhteentoimivuus

Suunnittelun toisessa vaiheessa laaditaan asiakkaiden kanssa yhteinen tietomalli, joka sisältää tarvittavat sanastot ja koodistot. Henkilötietojen käsittelyssä keskeinen

vaatimus on tietojen minimointi – mukaan otetaan vain välttämättömät tiedot. Ohjelmointirajapintaan voidaan lisäksi määritellä eri toimijoille erilaisia käyttöoikeuksia.

Semanttinen yhteentoimivuus on tärkeää, koska eri tietojärjestelmät voivat tulkita samaa tietoa eri tavoin. Jokaisella rajapinnalla on oma tietomallinsa, johon sisältyvät esimerkiksi sanastot ja erilaiset koodistot, kuten virheilmoitukset. Tietomalleja ei yleensä kannata rakentaa alusta asti, sillä moniin käyttötarkoituksiin on jo olemassa valmiita ja toimivia kansainvälisiä tietomalleja, joita voidaan tarvittaessa täydentää. Ajan ja päivämäärän esittämiseen käytetään esimerkiksi ISO 8601 -standardia, ja schema.org tarjoaa laajasti tunnettuja ja valmiiksi määriteltyjä tietomalleja.

Kun tietoja siirretään maiden välillä, esimerkiksi Pohjoismaissa, on järkevää käyttää englanninkielisiä tietomalleja, koodistoja ja sanastoja. Niiden avulla voidaan määritellä termien merkitykset täsmällisesti, jotta tieto tulkitaan samalla tavalla eri maissa. Vaikka Pohjoismaiden perusrekisterit ovat toiminnaltaan melko samanlaisia, ne eivät ole täysin yhteneviä. Siksi Suomen rajojen ulkopuolelle siirrettävien tietomallien tulee olla englanninkielisiä. Lisäksi tietojen siirron odotetaan lisääntyvän EU-maiden välillä, mikä korostaa englanninkielisten tietomallien merkitystä entisestään.

Selkeä tietomallien, koodistojen ja sanastojen suunnittelu säästää merkittävästi resursseja myöhemmissä vaiheissa.

1.5 Vaihe 4. Valitse toteutusteknologia

Kun tietomalli tai tarvittaessa useita tietomalleja on laadittu, siirrytään kolmanteen vaiheeseen: toteutusteknologian valintaan. Tässä vaiheessa on hyödyllistä ymmärtää eri rajapintateknologioiden perusteet, myös teknisen henkilön näkökulmasta.

Nykyisin arviolta 80–90 % rajapinnoista toteutetaan REST-arkkitehtuurityylillä. REST ei ole standardi, vaan joukko periaatteita, joita sovelletaan vaihtelevasti – siksi tarvitaan yhteisiä käytäntöjä, jotka määritellään tyylioppaassa.

GraphQL on uudempi vaihtoehto RESTille. Se ei ole standardi samalla tavalla kuin HTTP, mutta se on tarkasti määritelty ja ratkaisee monia RESTin käytännön haasteita, kuten tietojen yli- ja alikyselyn (overfetching ja underfetching).

1.5.1 RESTful-arkkitehtuurityylin ohjelmointirajapinnat

RESTful-arkkitehtuurityyli on käytetyin tyyli luoda ohjelmointirajapintoja. REST (lyhenne sanoista Representational State Transfer) on ohjelmistoarkkitehtuurityyli hajautettujen hypermediajärjestelmien toteuttamiseen. Roy Fielding määritteli sen vuonna 2000 julkaistussa *Architectural Styles and the Design of Network-based Software Architectures* -väitöskirjansa luvussa viisi. Tyyli asettaa

ohjelmointirajapinnoille viisi pakollista rajoitetta ja yhden vapaaehtoisen rajoitteen (taulukko 1).

Taulukko 1. RESTful-arkkitehtuurityyli rakentuu viidestä periaatteesta. Tavoitteena on antaa ohjelmistokehittäjälle mahdollisimman paljon vapautta.

	Rajoite englannin kielellä	Rajoite suomen kielellä	Kuvaus rajoitteesta
Pakolliset rajoitteet			
1.	Client-server	Asiakas-palvelin -malli	Arkkitehtuurissa on kaksi selkeää roolia: asiakas ja palvelin. Näiden tehtävät on eroteltu selkeästi.
2.	Stateless	Tilattomuus	Palvelin ei tallenna tilatietoja, vaan jokainen kysely sisältää kaiken siihen liittyvän tiedon.
3.	Cache	Välimuistin hyödyntäminen	Kyselyihin merkitään, voiko ne tallentaa välimuistiin (asiakkaan tai palvelimen). Välimuistin käyttäminen parantaa suorituskykyä.
4.	Uniform interface	Yhdenmukainen ohjelmointi-rajapinta	4a. Jokaisella resurssilla on yksilöllinen URL-osoite. 4b. Resursseja käsitellään niiden ilmentymien kautta. 4c. Kyselyt sisältävät kaiken tarvittavan tiedon niiden käsittelemiseen. 4d. Vastaus sisältää hyperlinkkejä muihin resursseihin (lyhenne HATEOAS, joka tarkoittaa Hypermedia as the Engine of Application State).
5.	Layered system	Kerrostettu järjestelmä	Järjestelmä voi rakentua kerroksittain, ja välityspalvelimia voidaan käyttää eri kerroksissa.
Vapaaehtoinen rajoite			
1.	Code on demand	Ohjelma tarpeeseen	Asiakas voi ladata palvelimelta suoritettavaa ohjelmakoodia (esim. JavaScriptiä). Käytetään hyvin harvoin.

RESTful-arkkitehtuurityyli on avoin ja teknologianeutraali. Sen asiakas- ja palvelinroolit voidaan toteuttaa useilla eri ohjelmointikielillä erilaisissa toimintaympäristöissä. REST

on myös teknologiariippumaton asiakas- ja palvelinroolien välisessä tiedonsiirrossa, sillä se ei rajoitu tiettyyn tietoliikenneprotokollaan. Käytännössä REST-arkkitehtuuria toteutettaessa hyödynnetään kuitenkin lähes aina HTTP(S)-protokollaa. Roy Fielding, arkkitehtuurityylin kehittäjä, oli myös mukana kehittämässä HTTP-protokollaa 1990-luvulla. Tästä syystä RESTful-arkkitehtuurityylissä käytetään HTTP-protokollaa laajasti.

RESTful-arkkitehtuuri luo lähtökohtia teknisen tason yhteentoimivuudelle, mutta teknisen yhteentoimivuuden saavuttaminen edellyttää esimerkiksi tyylioppaan johdonmukaista soveltamista kaikissa ohjelmointirajapinnoissa.

Kuvassa 5 on esitetty esimerkki RESTful-arkkitehtuurityyliä seuraavan ohjelmointitajapinnan toiminnasta. Siinä lähetetään kysely user123 -nimisen käyttäjän tiedoista. Vastauksena saadaan kysytyt tiedot.

Esimerkin lähdekoodi	**Esimerkin rivikohtaiset selitykset**

GET /users/123
 	
Accept: application/json
 	
Vastaus
 	
{
"id": 123,
"name": "Anna Virtanen",
"email": "anna.virtanen@example.com",
"address": {
"street": "Esimerkkikatu 1",
"city": "Helsinki"
}
}
	Ohjelmointirajapinnan kutsu (123-käyttäjän tiedot)
	Hyväksytään vastaus JSON-muodossa.
	
	Vastaus
	
	{
	id on 123
	nimi on ”Anna Virtanen”
	Sähköpostiosoite ”anna.virtanen…”
	Osoite
	Katuosoite Esimerkkikatu 1
	Kaupunki Helsinki
	}
	}

Kuva 5. Käytännön esimerkki RESTful-arkkitehtuurin ohjelmointirajapinnan kutsusta ja vastauksesta, jossa tieto on JSON-muodossa.

Ohjelmointirajapintoja, jotka eivät täytä kaikkia viittä pakollista rajoitusta, kutsutaan RESTin kaltaisiksi ohjelmointirajapinnoiksi. Usein esimerkiksi vaatimusta 4d, joka edellyttää hyperlinkkejä muihin resurssiin liittyviin tietoihin ja palveluihin, ei noudateta.

DTY/Honkanen Mika (DVV)

18.2.2026

1.5.2 Käytä yhteistä tyyliopista RESTille

REST ei ole tarkka standardi, vaan joukko yleisiä arkkitehtuuriperiaatteita, jotka ohjaavat ohjelmointirajapintojen suunnittelua. Tämä joustavuus on RESTin vahvuus, mutta samalla se aiheuttaa haasteita: ilman yhteisiä käytäntöjä ohjelmointirajapintojen toteutukset voivat poiketa toisistaan merkittävästi. Jos jokainen ohjelmistokehittäjä suunnittelee ja toteuttaa ohjelmointirajapinnat omien tulkintojensa mukaan, syntyy suuri määrä erilaisia ratkaisuja. Tämä johtaa siihen, että tietojärjestelmien välinen tekninen yhteentoimivuus heikkenee, virheiden mahdollisuus kasvaa, integraatiot vaikeutuvat ja palveluiden toimivuus kärsii.

Yhteentoimivuus on kriittistä erityisesti silloin, kun ohjelmointirajapintoja käytetään laajasti eri tietojärjestelmien ja organisaatioiden välillä. Ilman yhtenäisiä periaatteita jokainen integraatio vaatii ylimääräistä työtä, mikä lisää kustannuksia, hidastaa kehitystä ja kasvattaa virheiden riskiä. Tämän vuoksi tarvitaan selkeät ohjeet ja suositukset, jotka tukevat yhdenmukaista suunnittelua ja toteutusta.

Tyyliopas on keskeinen väline tämän tavoitteen saavuttamisessa. Se määrittelee yhteiset käytännöt ja parhaat toimintatavat, joiden avulla varmistetaan, että ohjelmointirajapinnat ovat yhtenäisiä, ennakoitavia ja helposti integroitavia. Näin voidaan edistää teknistä yhteentoimivuutta, parantaa palveluiden laatua ja vähentää kehitystyön kustannuksia.

Käytä tyyliopasta määrittämään, miten resurssit nimetään ja dokumentoidaan RESTful-arkkitehtuurissa, jotta kaikki palvelut seuraavat samoja periaatteita. Näin ohjelmointirajapinnat ovat enemmän yhteensopivia. Tässä keskeisiä syitä tyylioppaan merkitykseen:

1. **Johdonmukaisuus:** Tyyliopas varmistaa, että kaikki ohjelmointirajapinnat noudattavat samoja nimeämiskäytäntöjä, virheilmoitusrakenteita, URI-malleja ja vastausrakenteita. Tämä johdonmukaisuus tekee niistä helpommin ymmärrettäviä ja käytettäviä ohjelmistokehittäjille.

2. **Yhteentoimivuus:** Kun useita kehitystiimejä tai IT-toimittajia osallistuu kehittämiseen, auttaa tyyliopas pitämään kaikki samassa linjassa. Se estää erilaisten toteutustapojen ja standardien käytön, mikä voisi johtaa yhteentoimivuusongelmiin.

3. **Helppokäyttöisyys:** Tyyliopas edistää käyttökokemusta ohjelmistokehittäjien näkökulmasta. Kun rakenne, palautteet ja kutsujen logiikka ovat johdonmukaisia, ohjelmistokehittäjät voivat oppia yhden perusteella käyttämään muita helpommin.

4. **Virheiden vähentäminen:** Selkeästi määritellyt käytännöt ja standardit vähentävät väärinkäsityksiä ja virheitä suunnittelussa ja toteutuksessa. Ohjelmistokehittäjät voivat luottaa siihen, että tietyt käytännöt ja toimintatavat pätevät kaikkialla.

5. **Skaalautuvuus:** Kun organisaatio kasvaa ja kokonaisuudesta tulee monimutkaisempi, tyyliopas auttaa ylläpitämään hallittavuutta ja kehityksen sujuvuutta. Uudet kehitystiimit voivat liittyä mukaan helposti ja noudattaa määriteltyjä standardeja ilman, että heidän täytyy tulkita yksittäisten ohjelmistokehittäjien tekemiä valintoja.

6. **Ylläpidettävyys:** Tyyliopas tekee ylläpidosta helpompaa pitkällä aikavälillä. Koska kaikki noudattavat samoja toimintatapoja, on helpompaa tehdä päivityksiä ja korjauksia, sillä samanlaiset ongelmat voidaan ratkaista yhtenäisesti.

7. **Dokumentoinnin selkeys:** Tyyliopas auttaa myös dokumentoinnin yhdenmukaistamisessa. Yhtenäisesti kirjoitettu dokumentaatio auttaa ohjelmistokehittäjiä ymmärtämään ja hyödyntämään ohjelmointirajapintoja tehokkaammin.

Yhteenvetona tyyliopas on tärkeä työkalu, joka takaa laadun, käytettävyyden ja tehokkaan kehityksen niin rakentamisen kuin ylläpidonkin aikana.

Suomessa julkiselle hallinnolle ei ole määritelty omaa tyylioppaasta. Osa julkisen hallinnon organisaatioista hyödyntää esimerkiksi Googlen ja Zalandon avoimesti internetissä julkaisemia tyylioppaita, jotka löytyvät lähdeluettelosta. Jonkun yleisesti tunnetun tyylioppaan käyttö on paljon parempi ratkaisu kuin se, ettei tyyliopasta käytetä lainkaan. Usein tyylioppaissa on mietitty ja ratkaistu paljon suunnitteluun liittyviä asioita hyvin ja siksi niitä kannattaa kopioida. Tyylioppaan ja ohjelmointirajapinnan väliseen automaattiseen vertailuun ja tarkistamiseen kannattaa hyödyntää esimerkiksi avoimen lähdekoodin Spectral-ohjelmaa, johon linkin löydät lähdeluettelosta.

1.5.3 Hyödynnä GraphQL harkitusti monimutkaisiin kyselyihin

GraphQL alettiin kehittää Facebookilla vuonna 2011, kun Facebook oli kasvanut jo satoihin miljooniin käyttäjiin ja mobiilisovellusten datantarve alkoi kuormittaa hitaita ja epävakaita mobiiliverkkoja. REST-arkkitehtuurityylin ohjelmointirajapinnat hakivat liikaa dataa, vaativat useita pyyntöjä ja hidastivat kehitystä tilanteessa, jossa liiketoiminta kasvoi nopeasti ja käyttäjäkokemus oli kriittinen kilpailutekijä. GraphQL ratkaisi ongelman tarjoamalla tavan hakea juuri tarvittava data yhdellä pyynnöllä, mikä paransi mobiilisuorituskykyä, vähensi

kustannuksia ja nopeutti tuotekehitystä – kaikki suoraan kasvua ja skaalautuvuutta tukevia tekijöitä.

GraphQL on Facebookin kehittämä avoimen lähdekoodin kysely- ja muokkauskieli sekä tietojen hakurajapinta, joka tarjoaa joustavan ja tehokkaan tavan hakea dataa palvelimilta. Se toimii vaihtoehtona perinteisille REST-rajapinnoille ja mahdollistaa tarkemman tiedonhaun sekä tehokkaamman tiedonsiirron. GraphQL:ssa asiakas määrittää itse, mitä tietoja palvelimelta tarvitaan, ja kaikki kyselyt tehdään yhden URL-osoitteen kautta HTTP(S)-protokollaa käyttäen.

GraphQL:ssa asiakas määrittelee tarkasti, mitä tietoja haluaa — ei enempää eikä vähempää (kuva 6). Se vähentää sekä tietoliikenteen määrää että helpottaa tiedon käsittelyä.

Esimerkin lähdekoodi	**Rivikohtaiset selitykset**

query {
  user(id: 123) {
  name
  email
  }
}
 	
Vastaus
 	
{
  "data": {
  "user": {
  "name": "Anna Virtanen",
  "email": "anna.virtanen@example.com"
  }
  }
}
	Aloitetaan GraphQL-kysely
	id on 123
	kysytään nimeä
	ja sähköpostiosoitetta
	  }
	}
	
	Vastaus
	
	{
	Käyttäjä
	Nimi Anna Virtanen
	Sähköpostiosoite
	  }
	  }
	}

Kuva 6. GraphQL:ssa asiakas määrittelee tarkasti, mitä tietoja haluaa — ei enempää eikä vähempää.

GraphQL kehitys alkoi, kun Facebookilla työskennellyt ohjelmistoinsinööri esitteli uuden lähestymistavan ohjelmointirajapintoihin vuonna 2011. Vuonna 2015 GraphQL-spesifikaatio julkaistiin avoimesti, ja sen mukana julkaistiin myös JavaScript-kielellä toteutettu esimerkkikirjasto avoimen lähdekoodin projektina.

GraphQL:n perusidea on antaa asiakkaalle (esim. selain tai mobiilisovellus) mahdollisuus määritellä tarkasti, mitä tietoa se haluaa palvelimelta. Perinteisessä REST-rajapinnassa palvelin päättää, mitä kenttiä ja rakenteita palautetaan, mikä voi johtaa tilanteeseen, jossa asiakas saa paljon ylimääräistä dataa tai joutuu tekemään useita pyyntöjä. GraphQL ratkaisee tämän ongelman siirtämällä kontrollin asiakkaalle.

GraphQL-kyselyssä asiakas määrittelee haluamansa kentät ja rakenteen. Palvelin palauttaa vastauksessa vain ne kentät, jotka asiakas on pyytänyt – ei enempää eikä vähempää. Tämä vähentää datan siirtomäärää ja parantaa suorituskykyä erityisesti mobiili- ja rajallisen tiedonsiirron sovelluksissa.

GraphQL-rajapinta määritetään skeemalla, joka kuvaa resurssit, niiden tyypit ja suhteet. Graafiteorian näkökulmasta skeema on graafi, jossa resurssit ovat solmuja ja suhteet kaaria.

GraphQL-skeemaa voidaan kysyä ohjelmointirajapinnasta. Skeemassa määritellään kaikki ohjelmointirajapinnan tarjoamat operaatiot ja kenttien tyypit, mikä tuottaa automaattisesti ajan tasalla olevan dokumentaation ja toimii mallina kyselyjen validointiin. GraphQL Inspector on työkalu, joka auttaa hallitsemaan ja tarkistamaan GraphQL-rajapinnan muutoksia. GraphQL on tapa hakea dataa palvelimelta joustavasti – asiakas voi itse päättää, mitä kenttiä tarvitsee. Inspector varmistaa, että kun skeemaa (eli rajapinnan rakennetta) muutetaan, mikään vanha kysely ei mene rikki.

GraphQL-ohjelmointirajapinnassa on kaksi tärkeää tietoturvaan liittyvää seikkaa, joista on hyvä olla tietoinen. Ensinnäkin se ei tue piilotettuja kenttiä, joten kaikki ohjelmointirajapinnan kentät voidaan selvittää kysymällä. Toiseksi kyselyiden monimutkaisuutta ei ole rajoitettu, mikä tarkoittaa, että monimutkaisia kyselyitä voidaan käyttää hyväksi kuormittamalla GraphQL-ohjelmointirajapintaa.

GraphQL versiointitarve on pienempi kuin vastaavilla RESTful-arkkitehtuuria noudattavilla ohjelmointirajapinnoilla. Uusien kenttien lisääminen voi rikkoa RESTful-ohjelmointirajapinnan toimintalogiikkaa, koska se muuttaa kyselyihin saatavia vastauksia.

GraphQL edut verrattuna RESTful-arkkitehtuuriin korostuvat, kun kyselyt ovat monimutkaisia ja URL-osoitteet ovat pitkiä sekä sisältävät useita parametrejä. Tällöin GraphQL tarjoama joustavuus ja tehokkuus tulevat erityisesti esiin.

1.6 Vaihe 5. Sovella API ensin mallia valitun teknologian kanssa

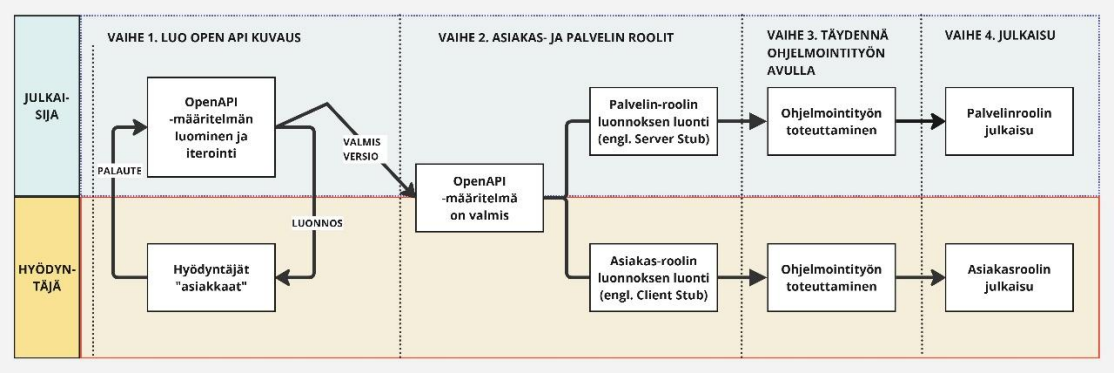
API ensin -mallin hyödyntäminen REST-ohjelmointirajapinnoissa

OpenAPI on standardoitu menetelmä RESTful-rajapintojen määrittelyyn ja dokumentointiin. Sen hyödyntäminen tuo merkittäviä hyötyjä ohjelmointirajapinnan suunnitteluun ja kehittämiseen:

- **Selkeyttää suunnittelua**: Rajapinnan rakenne on helppo hahmottaa ja viestiä eri sidosryhmille.
- **Nopeuttaa kehitystä**: OpenAPI-kuvauksesta voidaan automaattisesti tuottaa osia lähdekoodista, palvelimen rajapintamäärittelyjä ja dokumentaatiota.

- **Parantaa laatua ja käytettävyyttä**: Hyvin laadittu kuvaus tukee yhteentoimivuutta ja vähentää virheitä.
- **Tukee työkaluja**: OpenAPI toimii monien kehitystyökalujen perustana, mikä tehostaa koko kehitysprosessia.

RESTful-arkkitehtuurityylin mukaisessa ohjelmointirajapinnassa kannattaa hyödyntää OpenAPI-kuvausta. Sen avulla voi kuvata ohjelmointirajapinnan rakenteen ja toiminnan. OpenAPI on suoraviivainen tapa määritellä, mitä ohjelmointirajapinnan kautta voi tehdä ja miten se tehdään. OpenAPI-kuvauksesta voidaan helposti tuottaa ohjelmiston lähdekoodin osia, määritellä ohjelmointirajapinta sitä tarjoavaan palvelimeen ja tuottaa dokumentaation runko. Lisäksi kuvaus toimii mitä moninaisimpien aputyökalujen toiminnan avainosana. Kuvassa 4 on kuvattu prosessi.



Kuva 4. Uuden ohjelmointirajapinnan suunnitteluprosessi OpenAPI -kuvauksen avulla. Malli sopii sekä sisäisten että ulkoisten ohjelmointirajapintojen suunnitteluun.

Kuvassa neljä esitetty suunnitteluprosessi etenee neljässä vaiheessa:

1. **Rakenne**: Ohjelmointirajapinnan rakenne suunnitellaan yhdessä sitä hyödyntävien asiakkaiden kanssa iteratiivisesti kokeillen.

2. **Luonnokset**: Kun rakenne riittävän hyvä ja yhdessä hyväksytty, OpenAPI-kuvauksen avulla luodaan automaattisesti asiakas- ja palvelinroolien ohjelmakoodin luonnokset (tyngät). Näitä voidaan tuottaa useilla eri ohjelmointikielillä.

3. **Toteutus**: Luonnokset täydennetään ohjelmointityön avulla valmiiksi. Aluksi voidaan luoda testausympäristö ja sen jälkeen tuotantoympäristö.

4. **Tuotanto**: Molemmat roolit viedään tuotantoon.

OpenAPI-kuvaus nopeuttaa ja automatisoi kehitystyötä sekä palvelin- että asiakaspuolella. On tärkeää, että ohjelmointirajapinnan julkaisija julkaisee kuvauksen avoimesti, sillä se helpottaa merkittävästi ohjelmointirajapinnan hyödyntäjien työtä.

API ensin -mallin hyödyntäminen GraphQL:ssä

GraphQL-prosessissa noudatetaan samankaltaista periaatetta kuin OpenAPI-mallissa, mutta OpenAPI-määritelmän sijasta luodaan skeema, joka rakennetaan yhteistyössä asiakkaiden kanssa. Prosessi etenee seuraavasti:

1. **Suunnittele skeema**. Skeema on GraphQL ydin. Se määrittelee kaikki käytettävissä olevat tiedot, operaatiot ja tiedon rakenteet.
2. **Tee kokeiluversio.** Luo testiversio, jossa käytetään testidataa GraphQL-kyselyjen testaamiseen ilman todellista taustalla toimivaa tietojärjestelmää tai sen sisältämää oikeaa tuotantodataa. Tästä toteutuksesta voi luoda testiympäristön.
3. **Hae asiakaspalautetta ja muokkaa skeemaa ja kokeiluversiota.** Kun se muokkauttanut asiakaspalautteen perusteella riittävät hyväksi, siirry eteenpäin.
4. **Toteuta**. Kirjoita tarvittava ohjelmointikoodi ja yhdistä GraphQL taustalla toimiviin tietojärjestelmiin.
5. **Testaa**. Testaa tietojärjestelmän toiminta eri työkaluilla.
6. **Jatkokehitä**. Jatkokehitä sovellusta saadun palautteen perusteella. Skeemaan voi lisätä uusia kenttiä ilman erillistä versiointia.

1.7 Vaihe 6. Suosi avoimia dataformaatteja

Teknologianeuraalisuuden tavoite ja tärkeää huomioida ohjelmointirajapinnan dataformaatissa. Ohjelmointirajapinnan avulla siirtyvä tieto tallennetaan rakenteellisesti dataformaatin sisään. Käytetyimpiä ovat JavaScript Object Notation (JSON) ja Extensible Markup Language (XML).

JSON on yksinkertainen ja kevyt avoimen standardin dataformaatti tiedonvälitykseen ja tallennukseen. Nimestään huolimatta JSON on täysin ohjelmointikielestä riippumaton ja sitä voi pitää teknologianeutraalina.

XML on merkintäkielien standardi, joka määrittää tietojen merkintämuodon loogisella rakenteella. XML-kieliä käytetään sekä formaattina tiedonvälitykseen tietojärjestelmien välillä että tiedostomuotona dokumenttien tallentamiseen. XML-kieli on rakenteellinen kuvauskieli, joka auttaa jäsentämään laajoja tietomassoja selkeämmin. XML:n kehittäjäorganisaatio on World Wide Web Consortium.

Tärkeintä on valita avoin dataformaatti, jota voi käyttää ilman teollisuus- tai tekijänoikeuksia tai erillisiä maksuja ohjelmistotoimittajalle. Tästä perusperiaatteesta ei kannata poiketa kuin erittäin tarkasti harkittujen syiden perusteella. Suljetun tiedostomuodon käyttö voi luoda toimittajalukon.

1.8 Vaihe 7. Huolehdi tietosuojasta ja tietoturvasta

Ohjelmointirajapintojen tietoturva on keskeinen osa turvallista ja luotettavaa tietojärjestelmäarkkitehtuuria. Sekä REST- että GraphQL-rajapinnoilla on omat erityispiirteensä, jotka vaikuttavat siihen, miten tietoturva tulisi toteuttaa.

REST-rajapintojen tietoturva

REST-arkkitehtuurityyli ei itsessään määrittele tietoturvamekanismeja, mutta sen toteutuksissa on vakiintuneita käytäntöjä, joita tulisi noudattaa:

- **Autentikointi ja autorisointi:** Yleisimpiä menetelmiä ovat OAuth 2.0, API-avaimet ja JWT (JSON Web Token). On tärkeää erottaa käyttäjän tunnistaminen (autentikointi) ja käyttöoikeuksien hallinta (autorisointi).
- **Rate limiting:** Ohjelmointirajapintojen kuormituksen hallitsemiseksi voidaan rajoittaa pyyntöjen määrää aikayksikköä kohden. Tämä suojaa palvelua väärinkäytöksiltä ja palvelunestohyökkäyksiltä.
- **HTTPS:** Kaikki tietoliikenne tulisi salata TLS:n avulla. Salaamattomia HTTP-yhteyksiä ei tule sallia.
- **CORS (Cross-Origin Resource Sharing):** Määrittelee, mistä verkkotunnuksista ohjelmointirajapintaa saa kutsua. Tämä suojaa selaimessa toimivia sovelluksia väärinkäytöksiltä.
- **Virheiden hallinta:** Virheilmoituksissa ei tule paljastaa liikaa tietoa tietojärjestelmän sisäisestä toiminnasta.

GraphQL-rajapintojen tietoturva

GraphQL tarjoaa joustavuutta, mutta se tuo mukanaan myös erityisiä tietoturvahaasteita:

- **Ylikysely ja resurssien kuormitus:** Koska asiakas voi määritellä tarkasti, mitä tietoja haluaa, on mahdollista rakentaa erittäin raskaita kyselyitä. Tämän vuoksi on tärkeää käyttää kyselyiden monimutkaisuuden rajoittamista (esim. depth limiting, query cost analysis).

- **Kenttien näkyvyys:** GraphQL ei tue piilotettuja kenttiä skeemassa. GraphQL:llä on toiminnallisuus nimeltä introspektio, jonka avulla voi kysyä palvelimelta sen oman skeeman rakenteen (tyypit, kentät, kyselyt, mutaatiot jne.). Oletuksena introspektio näyttää kaikki kentät ja tyypit, jotka on määritelty skeemassa. Eli jos introspektio on sallittu, käyttäjät voi tehdä kyselyn, joka paljastaa koko API:n rakenteen.
- **Autentikointi ja autorisointi:** Vaikka GraphQL käyttää usein samoja mekanismeja kuin REST (esim. JWT), autorisointi on toteutettava kenttä- ja resolver-tasolla, koska kyselyt voivat kohdistua useisiin resursseihin samanaikaisesti.
- **Rate limiting ja throttling:** Vaikka GraphQL käyttää yhtä päätepistettä, on tärkeää valvoa ja rajoittaa pyyntöjen määrää ja kompleksisuutta.

Lähteet

Fielding, R. T. 2000. Architectural Styles and the Design of Network-based Software Architectures (Väitöskirja). University of California, Irvine. PDF-tiedostomuoto. Saatavissa: https://roy.gbiv.com/pubs/dissertation/fielding_dissertation.pdf. Viitattu 18.2.2024.

GraphQL. 2024. GraphQL. WWW-dokumentti. Saatavissa: https://spec.graphql.org/draft/. Viitattu 10.1.2024.

Google Cloud. 2025. Google Cloud API Design Guide. WWW-dokumentti. Saatavissa: https://cloud.google.com/apis/design/. Viitattu 5.11.2025.

Zalando. 2024. Zalando RESTful API and Event Guidelines. WWW-dokumentti. Saatavissa: https://opensource.zalando.com/restful-api-guidelines/. Viitattu 5.11.2025.

Verohallinto.2025. Vero API. WWW-dokumentti. Saatavissa: https://www.vero.fi/tietoa-verohallinnosta/kehittaja/vero-api/. Viitattu 5.11.2025.

Stoplight. 2024. Spectral: Open Source API Description Linter. WWW-dokumentti. Saatavissa: https://stoplight.io/open-source/spectral. Viitattu 5.11.2025.