Suomi.fi kehittäjille

Hyvät käytännöt ohjelmointiraja-pintojen suunnitteluun julkisessa hallinnossa

Liite 3

12.12.2024

DTY / Honkanen Mika (DVV)

Liite 3. RESTful-arkkitehtuurityylin ja GraphQL mukaiset ohjelmointirajapinnat

RESTful-arkkitehtuurityylin ohjelmointirajapinnat

RESTful-arkkitehtuurityyli on käytetyin tyyli luoda ohjelmointirajapintoja. Se koostuu kuudesta keskeisestä periaatteesta, jotka määrittelevät minimimäärän rajoitteita.

RESTful-arkkitehtuuri luo puitteet yhteentoimivuudelle, mutta todellisen yhteentoimivuuden saavuttaminen edellyttää sovittujen käytäntöjen, kuten tyylioppaan, johdonmukaista soveltamista kaikkialla.

REST (lyhenne sanoista Representational State Transfer) on ohjelmistoarkkitehtuurityyli hajautettujen hypermediajärjestelmien toteuttamiseen. Roy Fielding määritteli sen vuonna 2000 julkaistussa väitöskirjassaan *Architectural Styles and the Design of Network-based Software Architectures* luvussa viisi. Tyyli asettaa ohjelmointirajapinnoille viisi pakollista rajoitetta ja yhden vapaaehtoisen rajoitteen (taulukko).

Taulukko. 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 mitään tilatietoja, vaan jokainen kysely sisältää kaiken siihen liittyvän tiedon.
3.	Cache	Välimuistin hyödyntäminen	Kyselyihin merkitään, voiko ne tallentaa välimuistiin (asiakkaan tai palvelimen). Välimuistin hyödyntäminen parantaa suorituskykyä.
4.	Uniform interface	Yhdenmukainen ohjelmointirajapinta	4a. Jokaisella resurssilla on oma URL-osoite 4b. Resursseja käsitellään niiden ilmentymien kautta

DTY / Honkanen Mika (DVV)

			4c. Kyselyt sisältävät kaiken tarvittavan tiedon niiden käsittelemiseen. 4d. Vastaus sisältää hyperlinkkejä muihin resursseihin (HATEOAS: 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).

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 aina HTTP(S)-protokollaa. Roy Fielding, arkkitehtuurityylin kehittäjä, oli myös mukana kehittämässä HTTP-protokollaa 1990-luvulla. Tästä syystä RESTful-arkkitehtuurityylissä käytetään HTTP-protokollaa laajasti.

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.

Sovi yhteinen tyyliopas

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

1. **Johdonmukaisuus:** Tyyliopas varmistaa, että kaikki noudattavat yhtenäistä tapaa nimetä resursseja, käsitellä virheitä ja määrittää pyyntöjä ja vastauksia. Tämä johdonmukaisuus tekee API:sta helpommin ymmärrettävän ja käytettävän kehittäjille.

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

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

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

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

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

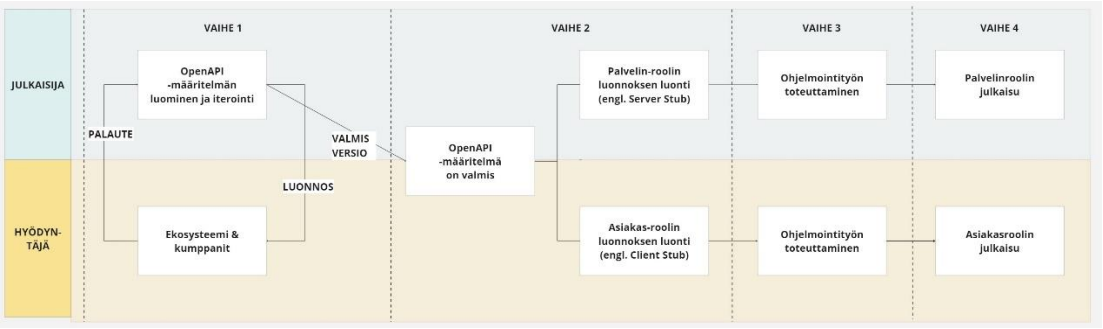
7. **Dokumentoinnin selkeys:** Tyyliopas auttaa myös dokumentoinnin yhdenmukaistamisessa. Yhtenäisesti kirjoitettu dokumentaatio auttaa kehittäjiä ymmärtämään ja hyödyntämään ohjelmointirajapintoja tehokkaammin.

Yhteenvetona tyyliopas on tärkeä työkalu, joka takaa laadun, käytettävyyden ja tehokkaan kehityksen niin rakentamisen kuin ylläpidonkin aikana. Suomessa julkisella hallinnolla ei ole omaa tyylioppaasta. Osa julkisen hallinnon organisaatioista hyödyntää esimerkiksi Googlen ja Zalandon avoimesti internetissä julkaisemia tyylioppaita.

Hyödynnä OpenAPI-kuvausta suunnittelussa (aiempi nimi Swagger)

OpenAPI-kuvaus on standardoitu tapa määritellä ja dokumentoida RESTful-rajapintoja. Se helpottaa suunnitteluprosessia ja tekee siitä helpommin viestittävää, nopeuttaa kehitysprosessia ja tekee käytöstä sekä kehittämisestä nopeampaa. Hyvin laadittu OpenAPI-kuvaus nopeuttaa kehitystyötä, parantaa laatua ja käytettävyyttä.

RESTful-arkkitehtuurityylin mukaisessa ohjelmointirajapinnassa kannattaa hyödyntää OpenAPI-kuvausta. Sen avulla on nopeaa ja yksinkertaista suunnitella ohjelmointirajapinnan rakenne. 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 neljä on kuvattu prosessi.



Kuva. Uuden ohjelmointirajapinnan suunnitteluprosessi OpenAPI -kuvauksen avulla.

Suunnitteluprosessin vaiheessa yksi ohjelmointirajapinnan rakennetta voi suunnitella yhdessä sitä hyödyntävien organisaatioiden kanssa iteroiden. Kun rakenne on riittävän hyvä, vaiheessa kaksi OpenAPI-kuvauksessa luodaan automaattisesti asiakas- ja palvelinroolien ohjelmakoodien luonnokset (tyngät). Ne voi luoda automaattisesti kymmenillä ohjelmointikielillä. Vaiheessa kolme ne täydennetään eli ohjelmoidaan valmiiksi. Vaiheessa neljä molemmat roolit viedään tuotantoon.

OpenAPI-kuvauksen avulla voi automatisoida ja nopeuttaa ohjelmistokehitystyötä sekä palvelin- että asiakasroolissa. Ohjelmointirajapinnan julkaisijan on tärkeää julkaista se avoimesti hyödyntäjien saataville, koska se helpottaa heidän työtänsä (asiakasroolin toteuttaminen) merkittävästi.

Hyödynnä avointa formaattia

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 järjestelmien välillä että tiedostomuotona dokumenttien tallentamiseen. XML-kieli on rakenteellinen kuvauskieli, joka auttaa jäsentämään laajoja tietomassoja selkeämmin. XML:n kehittäjä 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.

Hyödynnä GraphQL harkitusti monimutkaisiin kyselyihin

GraphQL on Facebookin kehittämä avoimen lähdekoodin kysely- ja muokkauskieli sekä tietojen hakurajapinta, joka tarjoaa joustavan ja tehokkaan tavan hakea tietoja palvelimilta. Se on vaihtoehto perinteisille RESTful-ohjelmointirajapinnoille ja mahdollistaa tarkemman tiedonhaun sekä tehokkaamman tietojen siirron.

GraphQL antaa asiakkaille (client) mahdollisuuden määrittää tarkasti, mitä tietoja he haluavat palvelimelta (server). Toisin kuin REST-rajapinnoissa, joissa jokaisella resurssilla on oma ennalta määrätty URL-osoite, GraphQL-rajapinnassa on vain yksi URL-osoite, jonka kautta asiakas voi tehdä kyselyitä eri tietomalleihin. GraphQL:ää käytetään internetin yli HTTP(S)-protokollalla.

GraphQL kehitys alkoi, kun Facebookilla työskennellyt ohjelmistoinsinööri esitteli uuden lähestymistavan ohjelmointirajapintoihin vuonna 2011. Vuonna 2015 GraphQL-spesifikaatio julkaistiin avoimesti, ja sen esimerkkitoteutus JavaScript-kielellä julkaistiin samalla avoimena lähdekoodina.

DTY / Honkanen Mika (DVV)

GraphQL:n perusajatus on siirtää palvelimen lähettämän datan rakenteen ja sisällön määrittely palvelinroolilta asiakasroolille. Asiakas määrittelee kyselyssään haluamansa kentät ja saa vastauksessaan vain ne kentät, jotka on määritellyt.

GraphQL-ohjelmointirajapinta määritetään skeeman avulla. Skeema määrittelee resurssit, niiden tyypit ja resurssien väliset suhteet. Graafiteorian käsittein skeema voidaan ajatella graafina, jossa resurssit ovat solmuja ja niiden väliset suhteet ovat 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-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.

API edellä -toimintamallin hyödyntäminen GraphQL:ssä:

1. **Suunnittele skeema**. Skeema on GraphQL ydin. Se määrittelee kaikki käytettävissä olevat tiedot, operaatiot ja tiedon rakenteet.

2. **Tee kokeiluversio.** Luo mock-versio, jossa käytetään feikkidataa GraphQL-kyselyjen testaamiseen ilman todellista taustapalvelua.

3. **Toteuta**. Kirjoita tarvittava ohjelmointikoodi ja yhdistä GraphQL taustajärjestelmiin.

4. **Testaa**. Testaa järjestelmän toiminta eri työkaluilla.

5. **Jatkokehitä**. Jatkokehitä sovellusta saadun palautteen perusteella. Skeemaan voi lisätä uusia kenttiä ilman erillistä versiointia.