HAMK
Hämeen ammatti-
korkeakoulu

JSON-skeemojen merkitys ohjelmistokehityksessä: Datan eheyden ja järjestelmien ylläpidettävyyden varmistaminen API-rajapinnoissa

Ammattikorkeakoulututkinnon opinnäytetyö

Tietojenkäsittelyn koulutusohjelma

Kevät 2025

Suvi Partanen

HAMK
Hämeen ammatti-
korkeakoulu

Tiivistelmä

Koulutus	Tietojenkäsittelyn koulutusohjelma	
Tekijä	Suvi Partanen	Vuosi 2025
Työn nimi	JSON-skeemojen merkitys ohjelmistokehityksessä: Datan eheyden ja järjestelmien ylläpidettävyyden varmistaminen API-rajapinnoissa	
Ohjaaja	Lasse Seppänen	

Opinnäytetyön tavoitteena oli tutkia JSON-skeemojen roolia ohjelmistokehityksessä, erityisesti niiden vaikutuksia datan eheyteen, API-kehitykseen ja monimutkaisten digitaalisten palveluiden ja järjestelmien hallintaan. JSON-skeema tarjoaa standardoidun tavan kuvata ja validoida JSON-dataa. Skeemat määrittelevät tarkkaan JSON-datan sisällön, rakenteen, tietotyypit ja rajoitteet. JSON-skeemojen käyttö vähentää olettamusten määrää, tehostaa kehitystyötä sekä varmistaa datan yhdenmukaisuuden ja eheyden API-rajapinnoissa.

Opinnäytetyön teoreettisessa osuudessa käsiteltiin JSON-skeeman määrittelyä, validoinnin mekanismeja sekä API-kehitystä. Opinnäytetyö oli luonteeltaan tutkimuksellinen. Koodiesimerkit havainnollistivat JSON-skeeman deklaratiivista rakennetta ja modulaarisuutta. Tapaustutkimusten avulla esitettiin JSON-skeemaa soveltavia ratkaisuja todellisissa ohjelmistokehitysprojekteissa. Kuuden organisaation tapaustutkimuksista koostettiin laadullinen sisällönanalyysi, jossa tunnistettiin JSON-skeeman soveltamisen keskeisiä teemoja.

Tutkimuksen tulokset osoittivat, että organisaatiot ovat merkittävästi parantaneet tiedon laatua ja eheyttä sekä ehkäisseet virheellisten tietojen päätymistä tuotantoon varhaisen skeemapohjaisen validoinnin avulla. JSON-skeeman yhdenmukainen validointi käyttöliittymä- ja palvelinpuolella sekä sen integroiminen automaatiotesteihin parantaa API-rajapintojen luotettavuutta. Ohjelmistojen vaatimusten kasvaessa ja monimutkaistuessa, modulaariset JSON-skeemat mahdollistavat järjestelmien joustavan hallinnan ja ylläpidon. JSON-skeemojen avulla voidaan rakentaa dynaamisia käyttäjäystävällisiä lomakkeita, varmistaa tiedonhallinta, pitää API-dokumentaatio ajan tasalla sekä parantaa tietoturvaa.

Työn tuloksena syntyi selkeä kuvaus siitä, miten JSON-skeemat voivat parantaa datan validointia, varmistaa tietojen eheyttä ja tukea ohjelmistojen ylläpitoa ja skaalautuvuutta. Tutkimuksen perusteella voidaan todeta, että JSON-skeemat tarjoavat merkittäviä etuja erityisesti API-kehityksessä, mikropalveluarkkitehtuureissa ja integraatioissa, joissa järjestelmät vaihtavat tietoa keskenään. Tulokset tarjoavat arvokasta tietoa ohjelmistokehittäjille, järjestelmäarkkitehdeille ja muille IT-alan ammattilaisille, jotka työskentelevät JSON-pohjaisten API-rajapintojen ja niiden kehityksen parissa.

Avainsanat	JSON-skeema, datan validointi, tiedon eheys, API-kehitys, modulaarisuus
Sivut	50 sivua ja liitteitä 3 sivua

HAMK
Hämeen ammatti-
korkeakoulu

Abstract

DP	Degree Programme in Information Technology	
Author	Suvi Partanen	Year 2025
Subject	Understanding JSON Schemas in Software Development: Ensuring Data Integrity and System Maintainability in APIs	
Supervisors	Lasse Seppänen	

The aim of the thesis was to examine the role of JSON Schema in software development, particularly its impact on data integrity, API development and the manageability of complex software systems and digital services. JSON Schema provides a standardized way to describe, validate, and document JSON data. It precisely defines the content, structure, data types and constraints of JSON data. Utilising JSON Schema reduces assumptions, enhances development efficiency, and ensures data consistency and integrity in API interfaces.

The theoretical part of the thesis covered JSON Schema specification, validation mechanisms and API development. The study was research orientated. Code examples illustrate the declarative structure and modularity of the JSON Schema. Case studies presented solutions applying JSON Schema in real-world software development projects. A qualitative content analysis was conducted based on case studies from six organizations which identified key themes related to the application of JSON Schema.

The research findings showed that organizations have significantly improved data integrity and prevented erroneous data from entering production through early schema-based validation. Consistent validation on both the frontend and backend side, and JSON Schema integration into automated tests, enhanced the reliability of API interfaces. As software requirements grow and become more complex, modular JSON Schema enabled flexible management and maintenance of software systems. JSON Schemas can be used to generate dynamic user-friendly forms, ensure data governance, keep API documentation up to date, and improve information security.

The study resulted in a clear description of how JSON Schema can enhance data validation, ensure data integrity, and support software maintenance and scalability. Based on the research, it can be concluded that JSON Schema offers significant advantages, particularly in API development, microservices architectures, and integrations where systems exchange data. The findings provide valuable insights for software developers, system architects, and other IT professionals working with JSON-based APIs and large data sets.

Keywords	JSON Schema, data validation, data integrity, API development, modularity
Pages	50 pages and appendices 3 pages

Sisällys

Kuvat

Taulukot

Ohjelmakoodit

Liitteet

Sanasto

API	Ohjelmointirajapinta, jonka avulla käyttöliittymä ja palvelin sekä eri sovellukset, voivat kommunikoida keskenään.
API-endpoint	API-päätepiste, jonka kautta voi lähettää pyyntöjä ja saada vastauksia, kuten tietyn resurssin hakeminen tai päivittäminen.
API-gateway	API-yhdyskäytävä on välikerros, joka hallitsee ja ohjaa API-kutsuja eri taustajärjestelmiin, tarjoten esimerkiksi kuormituksen tasapainotusta, autentikointia ja välimuistia.
Datan eheys	Datan tarkkuuden ja johdonmukaisuuden säilyttämistä sekä varmistamista koko sen elinkaaren ajan.
Datan johdonmukaisuus	Varmistaa, että data noudattaa tiettyjä sääntöjä ja on muodollisesti oikeanlaista, esimerkiksi päivämääräkentässä on oikea formaatti.
Deklaratiivinen rakenne	Pyrkii kuvailemaan, mitä ohjelman tulee saavuttaa sen sijaan että kuvattaisiin yksittäisiä suoritettavia käskyjä.
JSON-skeema	JSON Schema -standardiin pohjautuva määrittelykieli, joka mahdollistaa datan validoinnin, dokumentoinnin ja yhteensopivuuden hallinnan ohjelmistojärjestelmissä.
Modulaarisuus	Mahdollistaa järjestelmän joustavan kehittämisen ja laajentamisen jakamalla ohjelman itsenäisiin, uudelleenkäytettäviin komponenteiksi.
REST API	(Representational State Transfer) on suosittu arkkitehtuurityyli, joka mahdollistaa verkkopalveluiden joustavan kommunikoinnin HTTP-protokollan avulla käyttäen resursseja, kuten URL-osoitteita ja HTTP-menetelmiä.
Skaalautuvuus	Järjestelmän tai prosessin kyky käsitellä kasvavaa työkuormaa.
Tiedonhallinta	(Data Governance) määrittelee prosessit ja kontrollit, joiden avulla datan laatu ja yhtenäisyys varmistetaan.

1 Johdanto

Digitaalisten palveluiden ja järjestelmien monimutkaistuessa ohjelmistokehitykselle asetettavat vaatimukset kasvavat. Tiedon eheyden ja järjestelmien ylläpidettävyyden varmistaminen ovat keskeisiä tekijöitä, jotka määrittävät onnistuneen API-kehityksen. JSON-skeema tarjoaa tehokkaan työkalun näiden haasteiden ratkaisemiseksi. JSON-skeema mahdollistaa standardoidun tavan määritellä, validoida ja dokumentoida JSON-dataa. JSON-skeemat eivät ainoastaan auta datan validoinnissa, vaan ne myös tukevat API-kehitystä ja ylläpitoa – hyvin määritellyt skeemat vähentävät yhteensopivuusongelmia ja helpottavat erilaisten järjestelmien ja palveluiden integrointia.

Tämä tutkimuspainotteinen opinnäytetyö tutkii JSON-skeeman roolia ohjelmistokehityksessä sekä sen vaikutuksia datan eheyden varmistamiseen ja järjestelmien skaalautuvuuteen. Työn tavoitteena on tuottaa kattava analyysi siitä, miten JSON-skeemoja voidaan hyödyntää JSON-pohjaisissa API-rajapinnoissa ja monimutkaisten palveluiden ja järjestelmien hallinnassa. Teoreettisen tarkastelun lisäksi työssä havainnollistetaan käytännön esimerkkien avulla, missä tilanteissa ja käyttötapauksissa JSON-skeemat ovat erityisen hyödyllisiä. Koodiesimerkit ja tapaustutkimukset tarjoavat näkökulman JSON-skeemojen soveltamiseen datan validoinnissa ja rakenteen hallinnassa, mikä auttaa ymmärtämään niiden merkitystä todellisissa ohjelmistokehitysprojekteissa.

Työn tuloksena syntyy selkeä kuvaus JSON-skeemojen validoinnin mekanismeista ja hyödyistä sekä analyysi niiden käytöstä ohjelmistokehityksessä, erityisesti API-rajapinnoissa. Tulokset tarjoavat arvokasta tietoa ohjelmistokehittäjille, järjestelmäarkkitehdeille ja muille IT-alan ammattilaisille, jotka työskentelevät JSON-pohjaisten API-rajapintojen ja niiden kehityksen parissa. Tuloksia voidaan hyödyntää ohjelmistokehityksessä, ylläpidossa ja datan laadunhallinnan tehostamisessa.

Tutkimuksessa vastataan seuraaviin kysymyksiin:

- Mitä JSON-skeemat ovat ja missä niitä kannattaa hyödyntää?
- Miten JSON-skeemojen validointi vaikuttaa datan eheydestä huolehtimiseen API-rajapinnoissa?
- Miten JSON-skeemat edistävät monimutkaisten ohjelmistojärjestelmien kehitystä ja ylläpidettävyyttä?

2 JSON-formaatti

Ennen JSON-skeemaan perehtymistä on hyvä esitellä JSON-formaatti, sillä skeemat toimivat JSON-datan rakenteiden määrittelyn ja validoinnin työkaluina. JSON on kevyt, helposti ymmärrettävä, tekstipohjainen tiedonvaihtoformaatti. Siitä on tullut alan standardi tiedon jakamiseen eri sovellusten ja järjestelmien välillä, erityisesti verkkosovelluksissa ja JavaScript-pohjaisissa ympäristöissä. (Gbadebo, 2023) JSON-pohjaisten REST API-rajapintojen räjähdysmäinen kasvu, JSON-formaatin yksinkertaiset tietorakenteet sekä JavaScriptin kasvava suosio ovat osaltaan vaikuttaneet JSON-formaatin suosioon. (Marrs, 2017, luku 1) Työskennellessä datan kanssa, JSON-formaatin ja sen ekosysteemin ymmärtäminen on arvokas taito (Itelman & Cruz Viotti, 2024, luku 2).

JSON-data esitetään avain-arvo-pareina, jonka perustietotyypit ovat objekti, taulukot, luku (kokonais- tai desimaaliluku), merkkijono, boolean-arvo (totuusarvo) sekä null (tyhjä arvo). JSON-syntaksi ei salli kommentointia. JSON tukee sisäkkäisiä rakenteita, jolloin objektit ja taulukot voivat sisältää muita objekteja ja taulukoita. (Marrs, 2017, luku 1) Ohjelmointikoodissa 1 kuvataan esimerkki yksinkertaisesta JSON-objektista.

Ohjelmointikoodi 1. Yksinkertainen JSON-objekti (mukaillen Gbadebo, 2023).

```
{
  "id": "123",
  "name": "Maija Meikäläinen",
  "address": {
    "streetAddress": "Mannerheiminkatu 10",
    "city": "Helsinki",
    "country": "FI"
  },
  "yearsOfExperience": 3,
  "languages": ["Finnish","English"],
  "createdAt": "2025-02-01T08:10:00Z"
}
```

Ghabedon (2023) mukaan JSON tekee datan tallentamisesta ja vaihtamisesta helppoa. Se ei kuitenkaan pysty välittämään lisätietoja, kuten datan rakennetta, puuttuvia kenttiä tai sitä, miltä datan pitäisi näyttää. Se ei myöskään tarjota ominaisuuksilleen lisäkontekstia, mikä jättää tilaa oletuksille ja voi vaikeuttaa JSON-datan käsittelyä.

Ohjelmointikoodissa 1 esitetty JSON-data esimerkki sisältää tietoja työnhakijasta, mutta se jättää pois yksityiskohtia, mikä voi hankaloittaa sen käsittelyä. Ensinnäkin JSON-rakenne

voi olla epäselvä, sillä esimerkissä ei määritellä, mitkä kentät ovat pakollisia tai valinnaisia tai millaisia tietotyyppejä niillä tulisi olla. Pelkän JSON-datan perusteella ei tiedetä, tuleeko ID-kentän olla aina merkkijono vai voiko se olla myös numero. Jos kenttä on merkkijono, jää epäselväksi, pitäisikö sen olla UUID vai jokin muu muoto. Epäselvyys jättää paljon tulkinnanvaraa. Tämän lisäksi JSON-data voi olla epätäydellistä. Työnhakijan osoitetiedot voisivat sisältää esimerkiksi postinumeron tai puhelinnumeron, mutta ilman erillistä dokumentaatiota ei voida tietää, mitkä kentät ovat tarkoituksella jätetty pois ja mitkä puuttuvat.

JSON ei tue kommentteja, mikä hankaloittaa kenttien merkityksen ymmärtämistä ilman ulkopuolista dokumentaatiota. Lisäksi JSON:ista puuttuu sisäänrakennettu mekanismi tietojen validointisääntöjen ja rajoitteiden määrittelyyn. Esimerkiksi JSON-datan sisällössä ei voida täsmentää, että työnhakijoilla tulisi olla tietty määrä vuosia työkokemusta tai että heidän kielivalintansa tulisi kuulua ennalta määriteltyyn listaan eikä satunnaisiin sanoihin. (Gbadebo, 2023)

3 JSON-skeema

Tässä luvussa perehdytään JSON-skeeman määrittelyyn, hallintaan ja etuihin. Aluksi tarkastellaan JSON-skeeman tunnisteet, tyyppikohtaiset avainsanat sekä ehtoperusteinen skeemavalidointi. Tämän jälkeen syvennytään modulaaristen skeemojen hallintaan ja uudelleenkäyttöön, mikä edistää ohjelmiston joustavuutta ja ylläpidettävyyttä. Lopuksi käsitellään validointityökalujen merkitys sekä JSON-skeemojen hyötyjä ja käyttökohteita ohjelmistokehityksessä.

Tässä opinnäytetyössä termillä ”JSON-skeemat” viitataan JSON Schema -standardin mukaisiin skeemoihin. JSON-skeema on deklaratiivinen kieli, joka tarjoaa standardoidun tavan kuvata ja validoida JSON-dataa. Se määrittelee JSON-dokumentin sisällön, rakenteen, tietotyypit ja odotetut rajoitteet. Tämä vähentää olettamusten määrää ja auttaa varmistamaan JSON-datan yhdenmukaisuuden ja eheyden eri sovelluksissa. Tämä on erityisen tärkeää tiedonsiirrossa eri järjestelmien välillä. JSON-skeema-määritys on kehittynyt ajan myötä ja sitä kehittää ja ylläpitää avoin yhteistyöhön perustuva JSON Schema -yhteisö, joka toimii json-schema.org -sivuston kautta. JSON-skeema määrityksen versio on tällä hetkellä Draft 2020–12, ja sitä tukevat monet ohjelmistokehykset ja työkalut. (JSON Schema, n.d.-a; McIntyre, 2014, s.17)

JSON-skeema käyttää erillistä JSON-dokumenttia JSON-datan suunnitelmana. Skeemat vaihtelevat eri käyttötarkoitusten mukaan ja usein on tarpeen käyttää useita skeemoja JSON-datan rakenteen ja validoinnin määrittelyyn. Skeemat itsessään ovat sekä koneen että ihmisen luettavissa. Ohjelmointikoodi 2 havainnollistaa JSON-skeemaa, joka perustuu aiemmin esiteltyyn JSON-dataan (Ohjelmointikoodi 1). Skeemaesimerkki tarjoaa paljon enemmän taustatietoa alkuperäisestä JSON-datasta. Se määrittelee selkeästi kunkin kentän tietotyypin, mahdollisen formaatin, rajoitukset ja kuvaukset. Esimerkiksi kokemusvuodet on määritelty tietotyypiksi number, sekä lisätty kuvaus ja rajoite, että vain yli 1 vuoden työkokemus hyväksytään. (Gbadebo, 2023)

Ohjelmointikoodi 2. Perustason JSON-skeema (mukaillen Gbadebo, 2023).

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://example.com/document/candidate",
  "title": "Candidate data",
  "description": "This is an example candidate object illustrating JSON
Schema.",
  "type": "object",
  "properties": {
    "id": {
      "type": "string",
      "description": "Candidate ID",
      "example": "ABC12345",
      "pattern": "[A-Za-z0-9]{5,15}"
    },
    "name": {
      "type": "string",
      "description": "Candidate’s full name"
    },
    "address": {
      "type": "object",
      "properties": {
        "streetAddress": { "type": "string" },
        "postalCode": { "type": "string" },
        "city": { "type": "string" },
        "country": { "type": "string" }
      },
      "required": [
        "streetAddress", "city", "postalCode", "country"
      ]
    },
    "yearsOfExperience": {
      "type": "number",
      "description": "Work experience in years. Required min one year.
      "minimum": 1
    },

    "languages": {
      "type": "array",
      "items": {
        "type": "string",
        "enum": [ "Finnish", "Swedish", "English", "French", "German" ]
      }
    },
    "createdAt": {
      "type": "string",
      "format": "date-time"
    }
  },
  "required": [ "id", "name", "address" ]
}
```

3.1 JSON-skeeman määrittely

JSON-skeemassa käytettävät avainsanat määrittelevät, millaista dataa odotetaan ja miten sen oikeellisuus tarkistetaan. Avainsanat kertovat ohjelmille esimerkiksi, minkä tyyppisiä arvoja sallitaan ja mitkä ehdot datan on täytettävä. Avain-arvo-pareilla on mahdollista linkittää skeemoja toisiinsa, mikä tarjoaa joustavan ja skaalautuvan tavan yhdistää skeemarakenteita eri tilanteisiin. Yksinkertaisimmillaan yhdistäminen tapahtuu ehtoperusteisesti. (JSON Schema, n.d.-b)

3.1.1 Tunnisteet ja annotaatiot

Skeeman alkuun on hyvä kirjoittaa tunnisteet, kuten avainsanat $schema, millä ilmaistaan käytettävä versio sekä $id, millä yksilöidään skeema ja voidaan linkittää toisiin skeemoihin. JSON-skeema sisältää muutamia avainsanoja, jotka ovat tarkoitettu skeeman osien kuvaamiseen. Ne eivät ole pakollisia, mutta suositellaan hyvänä käytäntönä tukemaan dokumentointia. Title ja description ovat merkkijonoja, joista title on yleensä lyhyt otsikko ja description tarjoaa pidemmän selityksen skeemassa kuvatun tiedon tarkoituksesta (Ohjelmointikoodi 3). Default avainsanalla voidaan määritellä oletusarvo ja examples avainsanalla esimerkkejä datasta. (JSON Schema, n.d.-b)

Ohjelmointikoodi 3. JSON-skeeman tunnisteet (mukaillen JSON Schema, n.d.-b).

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://example.com/employee.schema.json",
  "title": "Employee data",
  "description": "Employee data, including personal and contact information.",
  "type": "object"
}
```

3.1.2 Perustietotyypit ja tyyppikohtaiset avainsanat

JSON-skeeman määrityksessä type-avainsana on keskeinen, sillä se määrittelee millaista tietoa skeeman mukainen JSON-rakenne voi sisältää. Taulukossa 1 esitetään JSON-skeeman tukemat perustietotyypit sekä tyyppikohtaiset avainsanat, jotka asettavat perustietotyypeille lisäehtoja (JSON Schema, n.d.-c).

Taulukko 1. Perustietotyypit ja tyyppikohtaiset avainsanat (mukaillen JSON Schema, n.d.-c).

Perustietotyypit	Tyyppikohtaiset avainsanat ja selite
object (objekti)	**properties**: Määrittää objektin ominaisuudet eli objektin sisältämät avaimet ja niiden arvojen tyypit. **additional properties**: Määrittää, sallitaanko objektissa muita kuin properties-kohdassa määriteltyjä avaimia. **required properties**: Luettelee objektin pakolliset kentät käyttämällä required-avainsanaa. **size** (minProperties, maxProperties): Rajoittaa objektin avain-arvo-parien määrän.
array (taulukko)	**items**: Määrittää taulukon alkioiden sallitut tyypit. **length** (minItems, maxItems): Rajoittaa taulukon pituuden.
string (merkkijono)	**length**: Määrittää merkkijonon sallitun pituuden (minLength, maxLength). **format**: Käytetään tiettyjen muotojen, kuten päivämäärän tai sähköpostiosoitteen, määrittelyyn. **pattern**: Mahdollistaa merkkijonon rakenteen tarkistamista säännöllisten lausekkeiden avulla (regular expressions).
number (liukuluku ja kokonaisluku), **integer** (kokonaisluku)	**range** (minimum, maximum): Määrittää pienimmän ja suurimman sallitun arvon käyttämällä minimum- ja maximum-avainsanoja.
boolean (totuusarvo)	Saa arvon true tai false eli tosi tai epätosi.
null (tyhjä arvo)	Edustaa tyhjää arvoa.

Nämä tyypit vastaavat useimmissa ohjelmointikielissä käytettäviä tietotyyppejä, vaikka niiden nimitykset voivat vaihdella eri järjestelmissä. Enum- ja const-avainsanat tarjoavat lisärajoitteita tietotyyppien määrittelyyn: enum määrittelee joukon ennalta määriteltyjä arvoja, joista tietyn ominaisuuden arvon tulee valita, kun taas const asettaa ominaisuudelle arvoksi ainoastaan yhden kiinteän arvo. Ohjelmointikoodi 2 perustason JSON-skeemasta havainnollistaa perustietotyyppien, tyyppikohtaisten avainsanojen ja enum-avainsanan käyttöä. Tämä konkretisoi, kuinka JSON-skeeman määrittelyssä käytetään type-avainsanaa ja siihen liittyviä lisämääritteitä tietotyypeille. (JSON Schema, n.d.-c)

3.1.3 Skeemojen yhdistäminen

JSON-skeema tarjoaa avainsanoja, joiden avulla useita skeemoja voidaan yhdistää. Yksinkertaisimmillaan yhdistäminen perustuu ehtoihin, joiden avulla voidaan varmistaa, että JSON-rakenne täyttää useita määriteltyjä ehtoja samanaikaisesti. Boolean-logiikkaan perustuva skeemojen yhdistäminen perustuu tunnetuille Boolen algebran periaatteille, kuten AND, OR, XOR ja NOT. Niiden avulla voidaan usein määritellä monimutkaisempia ehtoja, joita ei ole mahdollista esittää JSON-skeemassa pelkästään tavanomaisia avainsanoja käyttämällä. Boolean-logiikan mukaisten ehtojen yhdistämisessä käytettävät avainsanat ovat esimerkiksi **allOf**, joka tarkoittaa, että skeeman täytyy olla validi kaikkien aliskeemojen suhteen, **anyOf**, joka edellyttää validiutta vähintään yhdessä aliskeemassa, ja **oneOf**, jossa skeeman tulee olla validi tarkalleen yhdessä aliskeemassa. Lisäksi käytetään **not**-avainsanaa, joka määrittelee, että skeeman ei tule olla validi annetun skeeman suhteen. Nämä avainsanat tulee määritellä taulukoksi, jossa jokainen alkio on skeema. Boolean-logiikan käyttö mahdollistaa tarkemman kontrollin skeemojen validoinnissa. Rekursiivisia skeemoja käyttäessä on oltava varovainen, sillä ne voivat merkittävästi kasvattaa käsittelyaikaa. (JSON Schema, n.d.-d)

Ehtoperusteisessa yhdistämisessä skeeman rakenne määräytyy dynaamisesti JSON-datan sisällön perusteella. Tämä vastaa perinteisissä ohjelmointikielissä käytettyjä **if/then/else**-rakenteita ja mahdollistaa erilaisten aliskeemojen soveltamisen eri tilanteissa. Ehtoperusteiseen yhdistämiseen käytetään avainsanoja, kuten if, joka määrittelee ehdon, jonka perusteella valitaan sovellettava aliskeema. Mikäli if-ehto on validi, sovelletaan then-aliskeemaa, ja jos if-ehto ei ole validi, sovelletaan else-aliskeemaa. Näin JSON-datan validointi voidaan ohjata tilanteen mukaan, jolloin skeeman valinta perustuu datan sisältöön. Ehtoperusteinen yhdistäminen mahdollistaa skeeman mukautumisen eri tietosisältöihin, mikä parantaa joustavuutta JSON-datan validoinnissa (JSON Schema, n.d.-e).

Ohjelmointikoodissa 4 määritellään kolmen maan osoitetiedot. allOf-avainsanaa käytetään yhdistämään ehtoja, jotka mukauttavat validointisääntöjä valitun maan perusteella. if/then-rakenteella määritellään postinumeron muoto kussakin maassa: Suomessa se on viisinumeroinen, Ruotsissa kolmen numeron ja kahden numeron yhdistelmä, ja Alankomaissa neljän numeron ja kahden kirjaimen yhdistelmä. Postinumerot validoidaan säännöllisiä lausekkeita. (JSON Schema, n.d.-e).

Ohjelmointikoodi 4. Ehtoperusteinen skeemojen yhdistäminen (mukaillen JSON Schema, n.d.-e).

```
{
  "type": "object",
  "properties": {
    "streetAddress": { "examples": "Mannerheimintie 10", "type": "string" },
    "country": {
      "default": "Finland",
      "enum": [ "Finland", "Sweden", " Netherlands" ],
    }
  },
  "allOf": [
    {
      "if": {
        "properties": {
          "country": { "const": "Finland" }
        }
      },
      "then": {
        "properties": {
          "postalCode": {
            "examples": "00100",
            "pattern": "[0-9]{5}"
          }
        }
      }
    },
    {
      "if": {
        "properties": {
          "country": { "const": "Sweden" }
        },
        "required": ["country"]
      },
      "then": {
        "properties": {
          "postalCode": {
            "examples": "111 52",
            "pattern": "[0-9]{3} [0-9]{2}"
          }
        }
      }
    },
    {
      "if": {
        "properties": {
          "country": { "const": " Netherlands" }
        },
        "required": ["country"]
      },
      "then": {
        "properties": {
          "postalCode": {
            "examples": " 1015 AB",
            "pattern": "[0-9]{4} [A-Z]{2}"
          }
        }
      }
    }
  ]
}
```

3.2 Modulaaristen JSON-skeemojen hallinta ja uudelleenkäyttö

Koodin uudelleenkäytettävyys on keskeinen käsite ohjelmistokehityksessä. On yleensä parempi rakentaa ohjelma uudelleenkäytettäviksi komponenteiksi kuin kopioida ja liittää koodia joka paikkaan, jossa sitä tarvitaan. Sama periaate pätee myös JSON-skeemojen osalta. Tässä luvussa tarkastellaan JSON-skeemojen jäsentämistä, tunnistamista ja viittaustapoja. Näiden rakenteiden avulla skeemat voivat olla joustavampia, uudelleenkäytettäviä ja helpommin ylläpidettäviä. (JSON Schema, n.d.-f)

3.2.1 Skeeman tunnistaminen ja viittaaminen

JSON-skeemojen hallintaa helpottaa niiden selkeä jäsentely ja viittausten käyttö. Skeemadokumentit tunnistetaan yksilöllisesti ei-relatiivisilla URI-osoitteilla, mutta niihin voidaan viitata myös suhteellisilla viitteillä. Skeematunnisteet voidaan määritellä seuraavasti (JSON Schema, n.d.-f):

- URI tai ei-relatiivinen URI (non-relative URI): Täydellinen URI, joka sisältää skeeman (https) ja mahdollisesti fragmentin (#foo). Esimerkki https://example.com/document#section.
- Suhteellinen viite: Osittainen URI ilman skeemaa (https), mutta voi sisältää fragmentin (#foo). Esimerkki /document#section.
- URI-viite: Tämä on yleinen termi, joka kattaa sekä suhteelliset viitteet että ei-relatiiviset URI:t. Se voi siis olla joko osittainen (relatiivinen) tai täydellinen URI (ei-relatiivinen), joka sisältää fragmentin (#foo). Esimerkki: Sekä https://example.com/document#section (ei-relatiivinen) että /document#section (relatiivinen) ovat molemmat URI-viitteitä.
- Absoluuttinen URI: Täydellinen URI, joka sisältää skeeman (https) ilman fragmenttia (#foo). Esimerkki: https://example.com/document.

Skeematunnisteet eivät välttämättä ole todellisia verkko-osoitteita. Toteutukset eivät tee HTTP-pyyntöjä (https://) tai lue tiedostoja (file://) hakeakseen skeemoja. Sen sijaan ne tarjoavat tavan ladata skeemat sisäiseen skeemarekisteriin, josta ne haetaan tarvittaessa URI-tunnisteen perusteella. (JSON Schema, n.d.-f)

Perus-URI (base URI) helpottaa skeemojen hallintaa. Perus-URI:n ja URI-viitteen yhdistämällä saadaan muodostettua täydellinen URI (JSON Schema, n.d.-f) Esimerkiksi jos perus-URI on https://example.com/document ja URI-viite on /contact-info.schema.json,

saadaan muodostettua täydellinen URI: https://example.com/document/contact-info.schema.json.

Haku-URI (retrieval URI) on URI, jota käytetään skeeman hakemiseen (JSON Schema, n.d.-f). Esimerkkinä URI https://example.com/document/contact-info hakee Ohjelmointikoodissa 5 kuvatun skeeman.

Ohjelmointikoodi 5. Haku-URI (mukaillen JSON Schema, n.d.-f).

```
{
  "type": "object",
  "properties": {
    "phoneNumber": {
      "type": "string"
    },
    "email": {
      "type": "string",
      "format": "email"
    }
  },
  "required": ["phoneNumber", "email"]
}
```

Perus-URI voidaan määritellä $id-avainsanan avulla ja asettaa se skeeman alkuun. $id-avainsanan arvo on URI-viite ilman fragmenttia, joka yhdistetään haku-URI:in, mikä muodostaa näin skeeman perus-URI:n. (JSON Schema, n.d.-f)

3.2.2 Aliskeemojen viittaaminen ja uudelleenkäyttö

JSON-skeemoissa voidaan viitata aliskeemoihin useilla eri tavoilla, mikä mahdollistaa skeemojen joustavan uudelleenkäytön ja selkeän rakenteen. Näitä keinoja ovat JSON Pointer, $anchor, $ref ja $defs. Jokaisella menetelmällä on omat käyttötarkoituksensa ja etunsa. (JSON Schema, n.d.-f)

JSON Pointer on yleisin tapa viitata aliskeemaan. Se määrittelee polun dokumentin sisällä käyttämällä kauttaviivalla (/) eroteltuja avaimia. Esimerkiksi polku /properties/email viittaa properties-avaimeen ja sen sisällä email-avaimeen. (JSON Schema, n.d.-f) Esimerkiksi URI https://example.com/document/contact-info#/properties/email viittaa contact-info-skeeman sisällä olevaan email-aliskeemaan, joka on korostettuna Ohjelmointikoodissa 6.

Ohjelmointikoodi 6. JSON Pointer -viittaus aliskeemaan (mukaillen JSON Schema, n.d.-f).

```
{
  "type": "object",
  "properties": {
    "phoneNumber": {
      "type": "string"
    },
    "email": {
      "type": "string",
      "format": "email"
    }
  },
  "required": ["phoneNumber", "email"]
}
```

$anchor tarjoaa vaihtoehtoisen tavan viitata aliskeemaan. Sen avulla voidaan luoda nimetty ankkuri skeemaan käyttämällä $anchor-avainsanaa ja käyttää tätä nimeä URI-fragmentissa. (JSON Schema, n.d.-f) Esimerkiksi URI https://example.com/document/contact-info#email viittaa contact-info-skeeman sisällä olevaan email-aliskeemaan, joka on korostettuna Ohjelmointikoodissa 7.

Ohjelmointikoodi 7. $anchor-viittaus aliskeemaan (mukaillen JSON Schema, n.d.-f).

```
{
  "type": "object",
  "properties": {
    "phoneNumber": {
      "type": "string"
    },
    "email": {
      "$anchor": "email",
      "type": "string",
      "format": "email"
    }
  },
  "required": ["phoneNumber", "email"]
}
```

$ref-avainsanan avulla voidaan viitata ulkoisiin tai sisäisiin skeemoihin, mikä mahdollistaa skeemojen uudelleenkäytön. Esimerkiksi työntekijätietue voi viitata samaan yhteystietoskeemaan useissa käyttötarkoituksissa (Ohjelmointikoodi 8). Tämä yhteystietoskeema voi sijaita erillisessä JSON-skeemassa ja sisältää esimerkiksi työntekijän puhelinnumeron ja sähköpostiosoitteen. Jos myöhemmin halutaan lisätä uusia kenttiä, kuten osoitetiedot, tai päivittää nykyisiä kenttiä yhteystietoskeemassa, on

huomattavasti helpompaa tehdä tämä vain yhteen paikkaan. Tämä parantaa skeemojen ylläpidettävyyttä ja vähentää virheiden riskiä. (JSON Schema, n.d.-f)

Ohjelmointikoodi 8. $ref-viittaus toiseen skeemaan (mukaillen JSON Schema, n.d.-f).

```
{
  "$id": "https://example.com/document/employee",
  "type": "object",
  "properties": {
    "employeeId": {
      "type": "string",
      "pattern": "[A-Za-z0-9]{5,15}"
    },
    "firstName": {
      "type": "string"
    },
    "lastName": {
      "type": "string"
    },
    "contactInfo": {
      "$ref": "/document/contact-info"
    },
    "emergencyContact": {
      "$ref": "/document/contact-info"
    }
  },
  "required": [
    "employeeId",
    "firstName",
    "lastName",
    "contactInfo"
  ]
}
```

$defs-avainsana on hyödyllinen pienille aliskeemoille, jotka on tarkoitettu vain kyseisessä skeemassa käytettäviksi, jolloin erillisen skeeman luominen ei ole perusteltua. $defs-avainsana tarjoaa vakiintuneen tavan säilyttää nykyisessä skeemadokumentissa uudelleenkäytettävät aliskeemat. (JSON Schema, n.d.-f) Ohjelmointikoodissa 9 lisätään edelliseen skeemaesimerkkiin (Ohjelmointikoodi 8) yhteinen skeema nimiominaisuuksille. Nimi-aliskeemaa käytetään vain tässä skeemassa, joten $defs on hyvä valinta tähän tarkoitukseen (Ohjelmointikoodi 9).

Ohjelmointikoodi 9. $defs-viittaus aliskeemaan skeeman sisällä (mukaillen JSON Schema, n.d.-f).

```
{
  "$id": "https://example.com/document/employee",
  "type": "object",
  "properties": {
    "employeeId": {
      "type": "string",
      "pattern": "[A-Za-z0-9]{5,15}"
    },
    "firstName": {
      "$ref": "#/$defs/name"
    },
    "lastName": {
      "$ref": "#/$defs/name"
    },
    "contactInfo": {
      "$ref": "/document/contact-info"
    },
    "emergencyContact": {
      "$ref": "/document/contact-info"
    }
  },
  "required": [
    "employeeId",
    "firstName",
    "lastName",
    "contactInfo"
  ],
  "$defs": {
    "name": {
      "type": "string"
    }
  }
}
```

$defs, parantaa skeemojen luettavuutta ja ylläpidettävyyttä sekä auttaa vähentämään duplikaatteja. Skeeman monimutkaisempia osia voidaan määritellä $defs-osiossa selkeästi nimettyinä, jolloin niihin voi viitata tarpeen mukaan. Tämä helpottaa skeeman hahmottamista, sillä lukijat voivat ensin tarkastella sen rakennetta yleisellä tasolla ennen syventymistä yksityiskohtiin. Vaikka aliskeemaan voidaan viitata myös ulkoisesti, on suositeltavaa käyttää $ref-avainsanaa ulkoisiin skeemoihin viitatessa ja $defs-osiossa määriteltyjä aliskeemoja sisäisiin viittauksiin. (JSON Schema, n.d.-f)

3.2.3 Rekursiiviset skeemat

$ref-avainsanaa voidaan käyttää luomaan rekursiivisia rakenteita, joissa skeema viittaa itseensä. Tämä mahdollistaa esimerkiksi hierarkkisten tietorakenteiden, kuten organisaatiokaavioiden tai sisäkkäisten kohteiden, määrittelyn. (JSON Schema, n.d.-f) Ohjelmointikoodit 10 ja 11 havainnollistavat hakemistorakennetta, jossa kansio voi sisältää alikansioita. Rakenne voidaan määritellä seuraavasti:

Ohjelmointikoodi 10. Rekursiivinen tietorakenne JSON-skeema (mukaillen JSON Schema, n.d.-f).

```
{
  "$id": "https://example.com/document/folder",
  "type": "object",
  "properties": {
    "name": {
      "type": "string"
    },
    "subfolders": {
      "type": "array",
      "items": { "$ref": "#" }
    }
  }
}
```

Tässä "$ref": "#" tarkoittaa, että alikansioiden kentän (subfolders) arvo noudattaa samaa rakennetta kuin koko skeema, mikä mahdollistaa rekursion. Ohjelmointikoodissa 11 JSON-data noudattaa rekursiivista skeemaa ja kuvaa kansiorakenteen, jossa voi olla alikansioita.

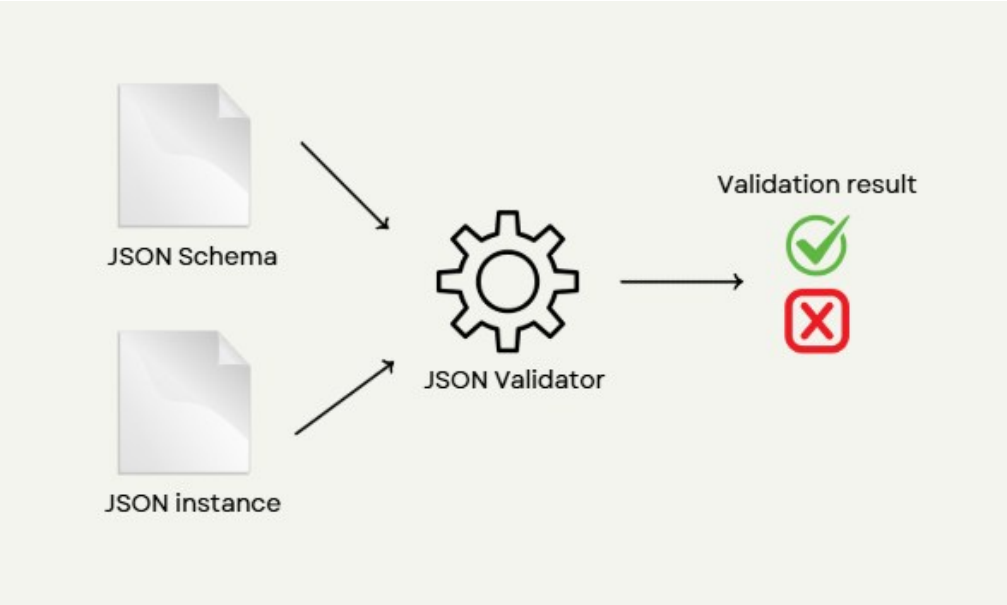
Ohjelmointikoodi 11. Rekursiivinen tietorakenne JSON-data (mukaillen JSON Schema, n.d.-f).

```
{
  "name": "Root Folder",
  "subfolders": [
    {
      "name": "Subfolder 1",
      "subfolders": [
        {
          "name": "Subfolder 1.1",
          "subfolders": []
        }
      ]
    },
    {
      "name": "Subfolder 2",
      "subfolders": []
    }
  ]
}
```

3.3 JSON-skeeman validointityökalut

JSON-dokumenttien validointi JSON-skeemaa vasten edellyttää erillistä validointityökalua. Validointityökalu tarkistaa, noudattavatko JSON-dokumentit määriteltyä JSON-skeemaa. Kuvassa 1 esitetään JSON-skeeman, JSON-datan ja JSON-validointityökalun vuorovaikutus. Validointityökalu tarkistaa datan skeeman mukaisuuden ja antaa tulokseksi joko hyväksytyn (pass) tai hylätyn (fail) tuloksen. JSON-skeeman validointityökalut toteuttavat JSON-skeema määrittelyn ja mahdollistavat JSON-skeemojen helpon integroinnin kaiken kokoisiin projekteihin. (JSON Schema, n.d.-a)

Kuva 1. JSON-skeeman validointityökalu (mukaillen JSON Schema, n.d.-a).



JSON-skeeman validointityökaluja on olemassa monia. Niiden ominaisuudet, yhteensopivuus ja ohjelmointikielikohtaiset ratkaisut vaihtelevat huomattavasti. JavaScript ja Node.js projekteissa yksi suosituimmista ja nopeimmista virallisista validointikirjastoista on Ajv JSON schema validator. Tämä kirjasto tukee uusimpia JSON-skeema standardeja ja mahdollistaa mukautettujen validointisääntöjen lisäämisen, mikä tekee siitä erinomaisen valinnan monenlaisiin ohjelmistokehitysprojekteihin. (GitHub, 2025)

Ajv:n laaja käyttö näkyy sen merkittävässä suosiossa Npm-sivustolla – sitä ladataan viikoittain 126,3 miljoonaa kertaa. Toiseksi suosituin vaihtoehto, JSON Schema validator, saavuttaa 21,3 miljoonaa viikkolatausta. (Npm, 2022; Npm, 2024) Muille ohjelmointikielille on saatavilla vastaavia validointikirjastoja, josta löytyy kattava listaus JSON Schema Org verkkosivuilta (JSON Schema, n.d.-g).

3.4 JSON-skeeman hyödyt

JSON-skeema tarjoaa useita etuja, jotka tekevät siitä tehokkaan välineen datan hallintaan ja validointiin. Gbadebon (2023) mukaan JSON-skeeman merkittävimmät hyödyt ovat data sopimukset, datan validointi, tiedonhallinta, parantunut datan laatu sekä dokumentaatio. Näiden hyötyjen ansiosta JSON-skeema voi merkittävästi parantaa järjestelmien luotettavuutta, yhteensopivuutta ja ylläpidettävyyttä.

Data sopimukset ovat keskeinen osa JSON-skeeman hyötyjä järjestelmien välisissä integraatioissa ja kommunikoinnissa – JSON-skeema toimii sopimuksena datan tarjoajien ja kuluttajien välillä. Se määrittelee datan muodon ja rajoitteet, mikä vähentää väärinkäsityksiä, oletuksia ja erimielisyyksiä datan odotuksissa. Ilman selkeitä määritelmiä eri järjestelmät voivat tulkita samaa dataa eri tavoin, mikä voi johtaa virheisiin ja häiriöihin prosesseissa. (Gbadebo, 2023) JSON-skeeman selkeä määrittely ohjaa eri tiimejä suuntaamaan kohti yhtenäistä visiota (Itelman & Cruz Viotti, 2024, luku 6).

Datan validointi on yksi JSON-skeeman merkittävimmistä eduista. JSON-skeema tarjoaa tarkasti määritellyn validointiprosessin ja tarkistaa, että JSON-data vastaa ennalta määriteltyjä sääntöjä ja rajoitteita. Skeemojen avulla voidaan varmistaa, että data noudattaa odotettua rakennetta ja tietotyyppejä, säilyttää datan eheyden ja estää virheellisen tai odottamattoman datan käytön. Tämä on erityisen tärkeää, koska se estää virheiden syntymistä ja varmistaa järjestelmän toimivuuden. Lisäksi JSON-skeema mahdollistaa automaattisen testauksen, joka tukee reaaliaikaista validointia ja datan eheyden tarkastamista. (Gbadebo, 2023)

Tiedonhallintaa (Data Governance) voidaan järjestelmällistää JSON-skeeman avulla, sillä JSON-skeema tukee organisaatioita tiedonhallinnan ja säädösten noudattamisen varmistamisessa. Se auttaa organisaatioita ylläpitämään kontrollia datan rakenteen ja laadun yli. Tämä on erityisen tärkeää toimialoilla, joilla on tiukat tiedonhallintaa koskevat vaatimukset. (Gbadebo, 2023; Srivastava, 2023) Tiedonhallinnon tavoitteena on varmistaa tietovarantojen laatu, saatavuus, eheys, turvallisuus ja käytettävyys. (Itelman & Cruz Viotti, 2024, luku 7).

Parantunut datan laatu on seuraus JSON-skeeman tarjoamasta validoinnista. JSON-skeeman validoinnin avulla voidaan merkittävästi parantaa datan laatua ja johdonmukaisuutta sovelluksissa ja järjestelmissä. Tämä vähentää datavirheitä ja johtaa

luotettavampiin prosesseihin. Datan tulee toimittaa oikeassa formaatissa, oikeaan aikaan ja oikealle vastaanottajalle. (Gbadebo, 2023)

Dokumentaatio tehostuu JSON-skeeman avulla. JSON-skeema määrittelee JSON-datan odotetun rakenteen ja siihen liittyvät rajoitteet. Se toimii dokumentaationa, jonka avulla API:n kuluttajat voivat selkeästi ymmärtää datavaatimukset ja noudattaa niitä datan tuottamisessa tai käsittelyssä. Tämä tekee sovellusten ja järjestelmien käytöstä entistä sujuvampaa ja ennakoitavampaa. Lisäksi JSON-skeeman selkeys helpottaa kehittäjien työtä, vähentää kommunikaatiohaasteita ja nopeuttaa uusien tiimin jäsenten perehtymistä. (Gbadebo, 2023)

3.5 JSON-skeeman käyttökohteita eri sovelluksissa

Gbadebon (2023) ja Srivastavan (2023) mukaan JSON-skeemaa käytetään laajasti eri sovelluksissa monenlaisiin tarkoituksiin eikä sen käyttö rajoitu pelkästään JSON-datan validointiin. JSON-skeeman etuja hyödyntävät useat käyttökohteet, jotka keskittyvät muun muassa API-datan validointiin, dynaamisiin lomakkeisiin, automaattisiin testeihin ja datan muuntamiseen. Skeeman rakenne ja selkeys sekä koneluettavuus tekevät siitä tehokkaan ratkaisun sekä kehittäjille että järjestelmien väliselle tiedonvaihdolle.

API-datan validointi on prosessi, jossa varmistetaan, että API:lle lähetetty ja sieltä vastaanotettu data on tarkkaa, täydellistä ja turvallista. JSON-skeemaa käytetään laajasti API-datan validointiin, jotta varmistetaan, että API-päätepisteiden kautta lähetetty ja vastaanotettu data noudattaa ennalta määriteltyä rakennetta ja täyttää asetetut rajoitteet. API-spesifikaatiot, kuten OpenAPI ja AsyncAPI, hyödyntävät JSON-skeemaa monipuolisesti eri validointitarkoituksiin, kuten pyyntöjen otsikot (request headers), pyyntö- ja vastaussisällön (request/response body) sekä kysely- ja polkuparametrien (query and path parameters) validointiin. (Gbadebo, 2023) Srivastava (2023) toteaa, että API-datan validointi parantaa API:en luotettavuutta ja auttaa estämään yhteensopivuusongelmia eri järjestelmien välillä.

Dynaaminen datan luominen on mahdollista JSON-skeeman avulla, jolloin luotu data noudattaa määriteltyä skeemaa. API-määrittelyn perusteella voidaan luoda esimerkkivastauksia API-pyynnöille käyttäen JSON-skeemaa. Lisäksi näitä esimerkkivastauksia voidaan käyttää luomaan mock-palvelimia. (Gbadebo, 2023)

Dynaamisen lomakkeen luominen on yksi JSON-skeeman käyttökohteista. JSON-skeeman deklaratiivisen rakenteen avulla on mahdollista luoda monimutkaisia lomakkeita. Lomakedata ja siihen liittyvä validointi voidaan esittää skeemamäärittelynä, jolloin lomakkeet voidaan luoda dynaamisesti skeeman pohjalta ja niiden vastauksia voidaan tarkistaa skeemaa vasten. Dynaamisissa lomakkeissa voidaan käyttäjän syötteen perusteella kohdistaa useita mukautettuja validointisääntöjä. (Gbadebo, 2023; Srivastava, 2023)

Datan muuntaminen ja ETL (Extract, Transform, Load) hyötyvät JSON-skeeman käytöstä. Datan muuntaminen tarkoittaa prosessia, jossa tiedot muunnetaan yhdestä formaatista toiseen, jotta ne ovat yhteensopivia eri järjestelmien ja sovellusten kanssa. JSON-skeemaa käytetään varmistamaan, että eri lähteistä integroidun tai ETL-prosessin läpikäyneen datan rakenne säilyy johdonmukaisena ja eheänä. Se auttaa määrittämään ja varmistamaan datan oikeellisuuden muuntamisen aikana vähentäen dataintegraation haasteita. (Gbadebo, 2023; Srivastava, 2023)

Konfiguraation validointi on myös yksi JSON-skeemojen käyttökohteista. JSON-skeemaa käytetään ohjelmistosovellusten konfiguraatiotiedostojen validointiin. Tämä varmistaa, että konfiguraatiotiedostot ovat oikeassa muodossa ja sisältävät voimassa olevia asetuksia. JSON-skeemojen avulla voidaan estää väärät konfiguroinnit, jotka voisivat aiheuttaa virheitä sovelluksessa. (Gbadebo, 2023; Srivastava, 2023)

API-dokumentaatio ja erityisesti API-datan rakenne voidaan luoda JSON-skeeman pohjalta. API-dokumentaatio tarjoaa ohjelmistokehittäjille selkeän ohjeistuksen siitä, miten rajapinnan kanssa tulee toimia ja millaista dataa se odottaa. Selkeä API-dokumentaatio auttaa vähentämään virheitä ja parantaa integraatiota, sillä kehittäjät voivat nopeasti tarkistaa, vastaako heidän lähettämänsä data odotettua rakennetta. (Srivastava, 2023)

Datamallinnus tarkoittaa tietojen rakenteiden ja tietosisältöjen kuvaamista. JSON-skeeman avulla voidaan mallintaa ja suunnitella tietorakenteita. Sen avulla voidaan määritellä datan ominaisuudet ja niiden väliset suhteet, mikä tarjoaa selkeän pohjan datan hallinnalle. Tämä tukee tietomallien kehitystä ja optimointia järjestelmän tarpeiden mukaan. (Srivastava, 2023)

Metatiedon hallintaa voidaan tehostaa käyttämällä JSON-skeemaa aineistojen metatiedon määrittelyyn. Tämä tarjoaa lisätietoa datasta, kuten sen tietotyypeistä ja rajoitteista, mikä parantaa datan ymmärrettävyyttä ja hallintaa. Skeemat auttavat myös

varmistamaan, että metatiedot ovat johdonmukaisia ja luotettavia koko järjestelmän laajuisesti. (Srivastava, 2023)

Automaattinen testaus on keskeinen osa ohjelmistorajapintojen laadunvarmistusta. JSON-skeema voidaan integroida automaatiotesteihin varmistamaan, että API-palveluiden JSON-vastaukset noudattavat odotettua rakennetta. Automaattiset testit tunnistavat virheet nopeasti ja tehokkaasti, ja mahdolliset virheet voidaan lokittaa virheraporttiin. Tämä auttaa ylläpitämään API:in luotettavuutta ja vakautta. (Srivastava, 2023)

Itelman & Cruz Viotti (2024, luku 2) mukaan JSON-skeemaa hyödyntävät muun muassa suuret teknologiayritykset, kuten Amazon, Microsoft ja Google sekä julkiset toimijat, kuten NASA, Yhdysvaltain kauppaministeriö, kansallinen turvallisuusvirasto (NSA) ja Ison-Britannian hallitus. JSON-skeema on myös keskeinen osa useita määrittelyjä, kuten OpenAPI, AsyncAPI, RAML ja W3C Web of Things.

4 API-kehitys ja validointi

API:t (Application Programming Interfaces) eli ohjelmointirajapinnat ovat yksi nykypäivän tehokkaimmista teknologioista, jotka mahdollistavat ohjelmistojen ja palveluiden välisen viestinnän. Ne ovat olennaisia verkkopalveluille, mobiilipalveluille ja taustajärjestelmien integraatioille. Voi sanoa, että API:t ovat välttämättömiä nykyaikaisessa ohjelmistokehityksessä. (Pedro, 2024, luku 1) API-rajapintojen pääroolit ovat tietojen siirtäminen eri järjestelmien välillä, toiminallisuuksien yhdistäminen, sekä tietojen validointi ja muokkaaminen.

Tässä luvussa käsitellään API-kehitystä ja keskeisiä tekijöitä, jotka vaikuttavat ohjelmistorajapintojen suunnitteluun ja toteutukseen. Aluksi tarkastellaan API-tyyppejä ja yleisimpiä rajapintamalleja, kuten REST, GraphQL ja gRPC, sekä käsitellään, kuinka käyttötarkoitus vaikuttaa valittavaan API-ratkaisuun. Tämän jälkeen tarkastellaan erityisesti REST API:n näkökulmasta miten validointi, dokumentointi ja versiointi tukevat rajapintojen luotettavuutta ja ylläpidettävyyttä – ja kuinka JSON-skeemojen käyttö edistää näiden toimintojen tehokkuutta.

4.1 API-tyypit käyttöoikeuksien mukaan

API-kutsuja käytetään niin saman sovelluksen sisällä kuin eri järjestelmien välillä. Rajapinnat voidaan jaotella eri käyttöoikeuksien mukaan näihin kolmeen: sisäinen API, kumppani API ja julkinen API. (Moilanen ym., 2018, s. 57–58; Postman, n.d.-a) **Sisäinen API** käytetään sovelluksen sisäiseen kommunikointiin esimerkiksi käyttöliittymän ja palvelimen välillä. Ne eivät ole avoimia organisaation ulkopuolelle. Sisäinen API voi olla private API, julkisessa verkossa oleva sisäiseen käyttöön tarkoitettu API, tai internal API, sisäverkossa käytettävä API. (Moilanen ym., 2018, s. 57–58) **Julkinen API** tarjoaa julkisen pääsyn organisaation tietoihin, toiminallisuuksiin tai palveluihin, joita kolmannet osapuolet voivat integroida omiin sovelluksiinsa. **Kumppani API** mahdollistaa kahden tai useamman organisaation tietojen tai toiminallisuuksien jakamisen yhteistyöprojektissa. Ne eivät ole julkisesti saatavilla, vaan niihin pääsee vain valtuutetut kumppanit tunnistautumismekanismeilla. (Postman, n.d.-a) **Avoin API** (Open API) on joko julkinen rajapinta tai kumppani rajapinta. Avoin API esimerkkejä on HSL:n Reittiopas API ja Ilmatieteen laitoksen API. Näiden lisäksi on vielä **Avoimen datan API** (Open Data Interface), joka tarjoaa avoimella lisenssillä tietoa muiden käyttöön. (Moilanen ym., 2018, s. 57–58) Taulukko 2 kuvaa API-luokittelua.

Taulukko 2. API-luokittelu (mukaillen Moilanen ym., 2018, s. 57–58).

Ohjelmointirajapinnan tyyppi		Julkisesti saatavilla	Käyttö maksaa	Data avointa
Avoimen datan API (Open Data Interface)		Kyllä	Yleensä aluksi ilmaista, mutta vaatimusten kasvaessa saataa olla maksullista.	Kyllä, sisältöön liitetty lisenssi määrittää oikeudet.
Avoin API (Open API)	Julkinen API (Public API)	Kyllä	Useasti kyllä. Voi sisältää myös ilmaiskerroksen (freetier) muodossa tai toisessa.	Ehkä
	Kumppani API (Partner API)	Ei	Ei, "kuuluu usein pakettiin".	Ei
Sisäiset APIt (private API eli julkisessa verkossa oleva sisäisen käytön tai internal API sisäverkossa sisäiseen käyttöön)		Ei	Ei	Ei

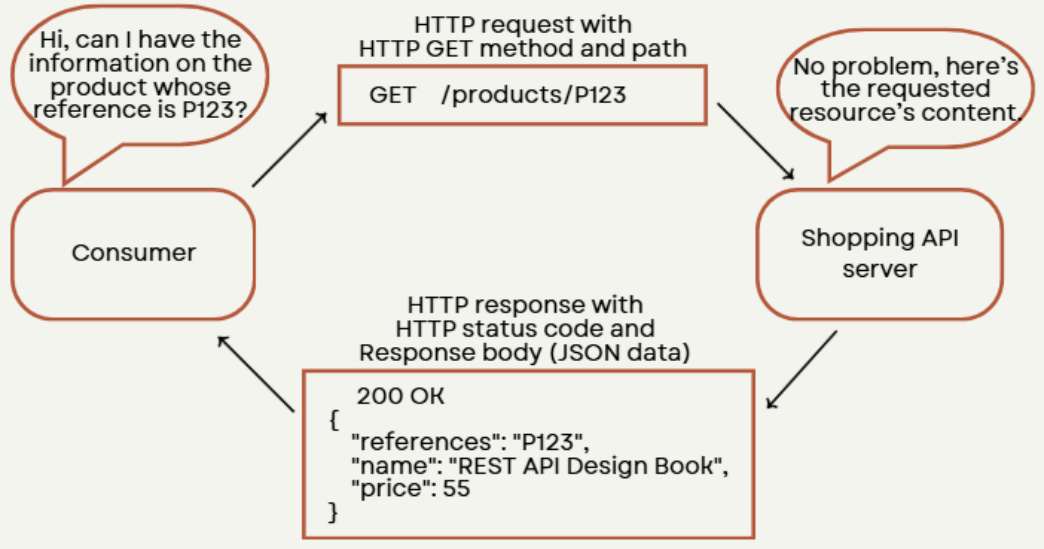
4.2 Yleisimmät API-arkkitehtuurityylit ja tiedonsiirtoprotokollat

API-suunnittelu alkaa sen käyttötarkoituksen määrittelystä, johon kaikkien sidosryhmien on sitouduttava. Ohjelmointirajapinnan käyttötarkoitus vaikuttaa myös sopivan arkkitehtuurityylin valintaan. (Postman, n.d.-b) Reselmanin (2020) mukaan yleisimpiä tapoja toteuttaa API-arkkitehtuurityylit ovat REST, SOAP, GraphQL ja gRCP.

REST (Representational State Transfer) on suosituin arkkitehtuurityyli API-rajapinnoille. REST on helppo käyttää ja laajasti tuettu. Sen HTTP-protokollaan perustuva toimintamalli tekee siitä erityisen hyvin yhteensopivan verkkoselainten, mobiilisovellusten ja kolmansien osapuolten integraatioiden kanssa. REST-rajapinnat hyödyntävät HTTP-menetelmiä, kuten GET tietojen hakemiseen, POST uusien resurssien luomiseen, PUT ja PATCH tietojen päivittämiseen sekä DELETE resurssien poistamiseen. (Fielding, 2000) REST-arkkitehtuuri mahdollistaa resurssien haun käyttämällä URL-osoitteita ja tunnisteita. Vaikka JSON on yleisin tiedonvälitysmuoto, voidaan käyttää myös muita formaatteja, kuten XML, CSV, ja jopa RSS. (Reselman, 2020) Kuva 2 havainnollistaa REST API:n eri komponentteja. Kuluttaja (Consumer) kommunikoi käyttöliittymän kautta palvelimen (Shopping API server) kanssa. API-pyyntö sisältää API-päätepisteen (API-endpoint), HTTP-metodin, kuten tässä GET-pyynnön tuotetietojen hakua varten, sekä parametrit, kuten products ja P123, osana

API-päätepistettä. API-vastaus (API response) puolestaan sisältää tilakoodin (status code) esimerkiksi 200 onnistuneesta API-kutsusta sekä vastaussisällön (response body), mikä tässä tapauksessa sisältää haetut tuotetiedot. (Lauret, 2019, luku 3)

Kuva 2. REST API -kutsu HTTP-protokollaa käyttäen (mukaillen Lauret, 2019, luku 2).



GraphQL on kasvattanut suosiotaan API-rajapintana ja on alun perin Facebookin kehittämä, mutta nykyisin avoimen lähdekoodin teknologia. GraphQL on suosittu erityisesti moderneissa verkko- ja mobiilisovelluksissa, koska se vähentää tarvetta tehdä useita peräkkäisiä API-kutsuja. GraphQL mahdollistaa joustavat kyselyt, joissa haetaan vain tarvittava data. Tämä parantaa suorituskykyä erityisesti monimutkaisissa tiedonhakurakenteissa. Vaikka GraphQL on tehokas ja joustava, sen yhdistäminen synkronisiin (reaaliaikainen) ja asynkronisiin (tapahtumapohjainen) toimintoihin voi olla haasteellista. (Reselman, 2020)

gRPC (Google Remote Procedure Call) on Googlen kehittämä avoimen lähdekoodin ohjelmistokehys. Se on erityisen suosittu mikropalveluissa, joissa tarvitaan nopeaa, tehokasta ja skaalautuvaa tiedonsiirtoa. Toisin kuin REST ja GraphQL, jotka käyttävät tekstipohjaista tietomuotoa, gRPC käyttää binääristä Protobuf-muotoa (nopeampi kuin JSON). Protocol Buffers vaatii, että asiakas/käyttöliittymä ja palvelin käyttävät samaa skeemamäärittelyä. gRPC tukee tehokasta striimausta, mikä on erityisen hyödyllistä ajankohtaisessa tiedonvaihdossa, kuten osakemarkkinatiedon käsittelyssä. gRPC on hyvä valinta mikropalveluarkkitehtuureihin ja sisäisiin API-kutsuihin. (Reselman, 2020)

SOAP (Simple Object Access Protocol) on tiedonsiirtoprotokolla, joka määrittelee tarkasti viestien rakenteen ja siirtomekanismit XML-muodossa. SOAP käyttää enemmän kaistanleveyttä kuin REST, mutta se tukee kehittyneitä tietoturva- ja transaktio-ominaisuuksia, jolloin se on hyvä valinta esimerkiksi rahoituspalveluissa. SOAP on legacy-protokolla, ja vaikka sitä käytetään edelleen vanhoissa järjestelmissä, uudemmat arkkitehtuurit suosivat moderneja tiedonsiirtomenetelmiä. (Reselman, 2020)

API:n käyttötarkoitus ohjaa arkkitehtuurivalintoja: gRPC on erinomainen valinta sisäisiin mikropalveluihin tehokkaan binäärimuotoisen viestinvälityksen ja suorituskyvyn ansiosta. Se tukee myös reaaliaikaisia päivityksiä ja vähentää viiveitä. GraphQL puolestaan soveltuu monien tietolähteiden hallintaan ja joustavaan tiedonhaun mekanismiin, sillä asiakas voi tarkasti määritellä tarvittavat tiedot, vähentäen turhaa tiedonsiirtoa. Jos API:n on oltava laajasti integroitavissa eri järjestelmiin ja sovelluksiin, REST-arkkitehtuuri tarjoaa standardoidun HTTP-pohjaisen ratkaisun, joka on helppo toteuttaa ja laajasti tuettu. Valinta näiden formaattien välillä riippuu arkkitehtuurin tarpeista ja vaatimuksista. (Fielding, 2000; Postman, n.d.-b; Reselman, 2020) Kun arkkitehtuurivalinta on päätetty kaikkien sidosryhmien kesken, API:n tavoitteet tulisi kuvata selkeästi luonnollisella kielellä, jotta ne ovat ymmärrettävissä myös niille, jotka eivät ole teknisiä asiantuntijoita. Tämä luo vahvan perustan API:en suunnittelulle ja kehitykselle. (Postman, n.d.-b) Tässä opinnäytetyössä tarkastellaan API-kehitystä erityisesti REST-arkkitehtuurissa, missä JSON-skeemat ovat erityisen hyödyllisiä.

4.3 API-dokumentaatio

API-dokumentaatio on tekninen ohjeistus, joka sisältää kattavat ohjeet ohjelmistorajapinnan tehokkaaseen käyttöön. Se sisältää kaiken tarvittavan tiedon API:n toiminnasta, kuten yksityiskohtaiset kuvaukset kustakin päätepisteestä, parametrit, pyyntö- ja vastausmuodot, tilakoodit sekä esimerkit ja ohjeet API:n käytöstä. Hyvä dokumentaatio parantaa kehittäjäkokemusta ja nopeuttaa API:n käyttöönottoa, niin sisäisten kehittäjien kuin ulkoisten kumppanien osalta. Se myös vähentää tukipyyntöjen määrää. Hyvä dokumentaatio helpottaa myös ylläpitoa, sillä kehittäjät ymmärtävät paremmin API:n rakenteen ja voivat tehdä nopeammin päivityksiä.

OpenAPI-kuvausta js SwaggerHub-työkalua, jotka hyödyntävät JSON-skeemoja, käytetään laajalti dokumentoinnin automatisointiin. Näiden työkalujen avulla voidaan luoda ajantasaisia ja helposti ylläpidettäviä dokumentaatioita, jotka parantavat kehittäjien

käyttökokemusta. Dokumentaation automaattinen päivitys ja validointi mahdollistavat tehokkaan työskentelyn ja virheiden minimoinnin API:n kehityksessä. (Vasudevan, 2023) Lisäksi API-dokumentaatio auttaa API:n testaamisessa, sillä selkeä ja kattava dokumentaatio tukee testauksen ja virheenkorjauksen prosessia, jolloin kehittäjät voivat paremmin varmistaa API:n oikean toiminnan ennen sen käyttöönottoa (Lauret, 2019, luku 12).

4.4 API-validointi ja datan eheys

Innocent (2023) toteaa artikkelissaan: ”API-validointi tarkoittaa prosessia, jolla varmistetaan, että API:lle lähetetty ja API:n palauttama data on virheetöntä, kattavaa ja turvallista.” Kattavalla datalla tarkoitetaan, että kaikki olennaiset tiedot, kuten lomakkeella vaadittavat kentät, ovat mukana. Tämä varmistaa, että API toimii odotetusti ja käsittelee erilaisia syötteitä ja tilanteita oikein (Innocent, 2024). API-validoinnin tehokkuutta voidaan parantaa seuraavien käytäntöjen avulla:

Skeemavalidoinnin avulla varmistetaan, että API:n käsittelemä data noudattaa ennalta määriteltyä rakennetta. Tämä sisältää datan rakenteen, tietotyyppien, pakollisten kenttien ja sallittujen arvojen tarkistamisen. JSON-skeema on erinomainen työkalu skeemavalidointiin, sillä se mahdollistaa validointisääntöjen ohjelmallisen määrittelyn sekä dokumentoinnin. (APItoolkit, 2024; Innocent, 2024)

Datavalidointi keskittyy datan sisältöön ja varmistaa, että arvot ovat oikeita ja sallituissa rajoissa. Esimerkiksi sähköpostiosoitteesta on hyvä tarkistaa oikea muoto (sisältää @-merkin) ja numerokentän arvon tulee olla määritellyn vaihteluvälin (esimerkiksi ikä 18–99 välillä). Datavalidointi suoritetaan mahdollisimman varhaisessa vaiheessa API-kutsua, jotta virheellinen data havaitaan ajoissa. (APItoolkit, 2024; Innocent, 2024)

Turvallisuusvalidointi suojaa API:a yleisiltä haavoittuvuuksilta, kuten SQL-injektioilta (tietokantakyselyiden manipulointi haitallisilla syötteillä), XSS-hyökkäyksiltä (haitallisen koodin suorittaminen selaimessa) ja CSRF-hyökkäyksiltä (hyökkääjän suorittamat valtuuttamattomat pyynnöt käyttäjän nimissä). SQL-injektio, jonka syötteenä yritetään antaa haitallinen komento, kuten { ”username”: ”’; DROP TABLE users;--” }, tulee estää validoinnilla. Tietoturvan parantamiseksi suositellaan käyttämään kirjastoja ja kehyksiä, joissa on sisäänrakennettuja tietoturvaominaisuuksia. (Harley, 2024, luku 5; Innocent, 2024)

Suorituskykyvalidoinnin tarkoituksena on varmistaa, että API toimii luotettavasti myös suuren kuormituksen alla. Kuormitustestien avulla voidaan tunnistaa mahdolliset pullonkaulat, varmistaa API:n vakaus eri olosuhteissa ja testata sen skaalautuvuus. Hyvin toimivat API:t parantavat käyttäjäkokemusta ja edistävät asiakasuskollisuutta (Innocent, 2024)

Edellä mainittujen neljän keskeisen käytännön – skeemavalidointi, datavalidointi, turvallisuusvalidointi ja suorituskykyvalidointi – lisäksi on olemassa suositeltavia käytäntöjä, jotka täydentävät API:n toimivuuden, turvallisuuden ja suorituskyvyn validointia. Yksi näistä on **varhainen validointi.** Datan validointi kannattaa suorittaa heti sen saavuttua API:in, esimerkiksi API-yhdyskäytävän (API-gateway) tasolla. Tämä estää virheellisen datan käsittelyn ja leviämisen järjestelmässä. Varhainen validointi tukee turvallisuusvalidointia parantaen järjestelmän suojatumista mahdollisilta uhilta. **Validointiprosessien automatisointi** parantaa API:n luotettavuutta ja vähentää manuaalista työtä. JSON-skeeman validointi integroituna API-testauksen automatisointiin auttaa pitämään datan eheänä sekä validointisäännöt ajantasaisina. API:n kehittyessä myös **säännöllinen validointisääntöjen päivitys** on oleellista, jotta ne vastaavat uusia vaatimuksia. Säännöllinen tarkastelu varmistaa validointisääntöjen ajantasaisuuden. Ja kun validointisääntöjä päivitetään, **versionhallinta** auttaa estämään integraatioiden rikkoutumisen ja mahdollistaa sujuvan siirtymisen uuteen versioon. Näiden käytäntöjen lisäksi, on suositeltavaa implementoida **yhtenäiset virheilmoitukset**. Virheilmoitusten tulee olla selkeitä ja johdonmukaisia, jotta virheen syy on helposti ymmärrettävissä ja korjattavissa. Virheilmoitusten tulee ilmaista mikä kenttä ei läpäissyt validointia ja miksi. Toteuttamalla nämä käytännöt voidaan varmistaa, että API käsittelee dataa luotettavasti ja turvallisesti, säilyttäen datan eheyden. (APItoolkit, 2024; Innocent, 2024)

Datan eheys tarkoittaa sen tarkkuutta, luotettavuutta, laatua sekä oikeellisuutta järjestelmien välillä siirrettäessä. Se on perusta, joka varmistaa, että data säilyttää alkuperäisen merkityksensä koko elinkaarensa ajan. (APItoolkit, 2024; Harward Business School) Organisaation datan eheyden säilyttäminen on jatkuva prosessi. Datan eheyden uhkia voivat olla esimerkiksi inhimillinen virhe, epäjohdonmukaisuudet eri formaateissa, tai jos kerätty data on virheellistä tai siitä puuttuu tietoa. (Harward Business School)

4.5 API-rajapintojen ylläpito ja versionhallinta

Lauret (2019, luku 9) mukaan on suositeltavaa jo ohjelmistorajapinnan suunnitteluvaiheessa huomioida sen laajennettavuus. Tämä helpottaa jatkokehitystä, vähentää yhteensopivuusongelmien riskiä sekä parantaa API:n uudelleenkäytettävyyttä ja skaalautuvuutta. Ylläpidon tärkeimpiin tehtäviin kuuluvat ohjelmiston päivitykset, uusien vaatimusten mukaiset muutokset virheenkorjaukset, käyttäjätuen tarjoaminen sekä yhteensopivuuden ja tietoturvan varmistaminen sen elinkaaren aikana.

API-kokonaisuuden monimuotoisuus (variety) syntyy tarpeesta palvella erilaisia käyttötarkoituksia ja käyttäjäryhmiä sekä suunnittelukäytäntöjen muuttuessa ajan myötä. Eri API:t voivat ratkoa saman ongelman eri tavoilla, mikä vaikeuttaa kokonaisuuden hallintaa ja lisää ylläpidon kuormaa. Yhtenäiset käytännöt ja selkeät ohjeistukset vähentävät tarpeetonta vaihtelua ja tukevat API-tuotteiden yhdenmukaisuutta. Myös sanastojen (vocabulary) hallinta on suositeltavaa; kun eri rajapinnoissa käytetään samoja käsitteitä ja rakenteita, vähenee riski päällekkäisille tai ristiriitaisille toteutuksille. JSON-skeemat ovat tässä keskeinen työkalu, sillä ne mahdollistavat rakenteiden standardoinnin, validoitavuuden ja uudelleenkäytettävyyden. Skeemojen hyödyntäminen helpottaa muutosten hallintaa ja parantaa API:n ylläpitoa. Versiointi on olennainen osa ylläpitoa, sillä se mahdollistaa ohjelmointirajapintojen kehittämisen ja laajentamisen ilman, että API:n käyttäjät joutuvat kohtaamaan katkoksia tai yhteensopivuusongelmia. (Medjaoui ym., 2018, luku 10)

API-versiointi on prosessi, jossa hallitaan ja seurataan API:in tehtyjä muutoksia sekä viestitään niistä API:n käyttäjille. API-kehityksessä muutoksia tapahtuu, kun lisätään uusia toiminnallisuuksia tai korjataan nykyisiä. Jotkin muutokset eivät vaikuta käyttäjiin lainkaan, toiset niin sanonut ”breaking changes” voivat aiheuttaa yhteensopivuusongelmia. API tulee versioida, kun muutokset vaativat käyttäjiä päivittämään koodiaan eli kun kyseessä on ”breaking change”. API-versiointi varmistaa, että nämä muutokset otetaan käyttöön hallitusti, jotta käyttäjien luottamus säilyy ja API pysyy turvallisena, virheettömänä ja suorituskykyisenä. (Pedro, 2024, luku 16; Postman, n.d.-c)

API-versiointiin on useita eri tapoja. On suositeltava valita API versiointi strategia jo API-suunnittelu vaiheessa ja käyttää sitä kaikissa API-rajapinnoissa. Yksi tapa hallita versiopäivityksiä on käyttää versionumeroita URL-polussa (esimerkiksi https://example.com/v1/products). Toinen vaihtoehto on, että versio määritellään kyselyparametrinä (esimerkiksi https://example.com/products?version=v1). Kolmas

vaihtoehto on lähettää versiotieto API-pyynnön otsikossa. Nämä versiointistrategiat yhdistetään versiointimalliin, kuten semanttiseen versiointiin tai päivämääräpohjaiseen versiointiin. Semanttinen versiointi käyttää kolmen numeron muotoa (esimerkiksi 3.2.1), jossa ensimmäinen numero viittaa merkittävään päivitykseen, joka voi sisältää rikkovia muutoksia, toinen numero tarkoittaa päivitystä, joka tuo uusia aiempien versioiden kanssa yhteensopivia ominaisuuksia, kolmas numero edustaa virheenkorjauksia tai ohjelmistopäivityksiä. (Pedro, 2024, luku 16; Postman, n.d.-c)

Huolellinen API-suunnittelu, dokumentointi, validointi, testaus ja versionhallinta ovat keskeisiä tekijöitä ohjelmistorajapinnan pitkäaikaisessa ylläpidossa ja käytettävyyden varmistamisessa. Yhtenäiset suunnittelukäytännöt ja sanastojen hallinta auttavat vähentämään API:en päällekkäisyyksiä ja ristiriitaisuuksia parantaen kokonaisuuden hallintaa sekä ylläpidettävyyttä. Myös aikaisemmin mainitut turvallisuusvalidointi ja suorituskykyvalidointi sekä monitorointi ovat oleellisia varmistamaan rajapinnan sujuva toiminta sen elinkaaren aikana. (Medjaoui ym., 2018, luku 4; Medjaoui ym., 2018, luku 10)

5 Tutkimusstrategia

Tutkimus toteutettiin laadullisena tapaustutkimuksena, jossa tarkasteltiin kuuden organisaation käytäntöjä ja kokemuksia JSON-skeemojen implementoinnista eri sovelluksissa ja järjestelmissä. Tutkimuksen lähestymistavaksi valittiin tapaustutkimus, koska se mahdollistaa syvällisen tarkastelun JSON-skeemojen käytön vaikutuksista ohjelmistokehityksessä eri konteksteissa.

Erikssonin & Koistisen (2014, s. 1–3) mukaan tapaustutkimus on tutkimusmenetelmä, jossa keskeistä on yksi tai useampi tarkoin määritelty tapaus, kuten yksilö, organisaatio tai ilmiö. Se tarjoaa moniulotteisen ja joustavan lähestymistavan, joka voi hyödyntää sekä laadullisia että määrällisiä aineistoja. Tapaustutkimus on sopiva lähestymistapa, kun jokin seuraavista toteutuu: mitä-, miten- ja miksi-kysymykset ovat keskeisiä, tutkijalla on vain vähän kontrollia tapahtumiin, aiheesta on tehty vähän empiiristä tutkimusta, tutkimuskohteena on jokin tämän ajan elävässä elämässä oleva ilmiö. (Eriksson & Koistinen, 2014, s. 4–5) Tässä opinnäytetyössä tapaustutkimus on pääasiassa kuvailevaa, mutta osittain myös selittävää.

Tutkittavan tapauksen tai tapausten valinta ja tutkimuksen kuluessa tapahtuva täsmentäminen ovat tapaustutkimuksen tärkeimpiä työvaiheita. Tapaustutkimuksessa voidaan hyödyntää monenlaisia aineistolähteitä, kuten haastatteluja, media-aineistoja, tilastoja ja havainnointia sekä erilaisia dokumentteja, kuten esitteet, muistiinpanot ja päiväkirjat. (Eriksson & Koistinen, 2014, s. 30–33) Tämän tutkimuksen aineisto kerättiin julkisesti saatavilla olevista käyttötapauksista virallisilta JSON Schema -sivuilta sekä organisaatioiden omilta verkkosivuilta. Aineistossa käsitellään JSON-skeemojen käyttöönottoa ja niiden vaikutuksia organisaatioihin. Näiden tapaustutkimusten avulla pyritään ymmärtämään, missä konteksteissa JSON-skeemoja on hyödynnetty sekä miten JSON-skeemat tukevat ohjelmistokehitystä, parantavat järjestelmien ylläpidettävyyttä ja tehokkuutta, sekä edistävät datan eheyden varmistamista. Tapaukset valittiin siten, että ne kattavat erilaisia käyttötarkoituksia ja organisaatioita eri toimialoilta sekä erilaisia API-ratkaisuja. Tämä valinta tukee opinnäytetyön rajauksia ja auttaa vastaamaan tutkimuskysymyksiin.

Tapaustutkimuksen analysointiin käytettiin laadullista sisällönanalyysiä. Tämä menetelmä auttaa tunnistamaan toistuvia teemoja ja käsitteitä aineistosta (Eriksson & Koistinen, 2014, s. 34–37). Analyysi perustuu tapauskohtaisiin havaintoihin ja vertailuihin, jotka tehdään sen perusteella, miten JSON-skeemat ovat vaikuttaneet organisaatioiden toimintatapoihin, kuten virheiden vähenemiseen, tuottavuuden kasvuun ja tietojen eheyteen. Tavoitteena oli

luoda syvällinen ymmärrys siitä, kuinka organisaatiot ovat hyödyntäneet JSON-skeemoja eri tilanteissa, ja kuinka niiden käyttöönotto on vaikuttanut datan validointiin, ohjelmistojen kehitykseen ja ylläpitoon.

6 Tapaustutkimukset

Tässä luvussa tarkastellaan, miten kuusi organisaatiota on omaksunut JSON-skeeman käytön. Tapaustutkimus on pääosin kuvailevaa ja myös selittävää. Tapaukset tarjoavat käytännön esimerkkejä siitä, miten JSON-skeemojen implementointi on vaikuttanut ohjelmistokehityksen eri osa-alueisiin. Kukin tapaustutkimus esittelee keskeiset havainnot, haasteet ja saavutetut tulokset.

Lopuksi tapaustutkimuksista kootaan sisällönanalyysi, jossa tunnistetaan toistuvia trendejä ja keskeisiä teemoja. Näiden tapausten avulla pyritään syventämään ymmärrystä siitä, missä konteksteissa ja käyttötapauksissa JSON-skeemat ovat osoittautuneet erityisen hyödyllisiä ohjelmistokehityksessä. Yhdessä tietoperusta ja tapaustutkimukset muodostavat vastauksen tutkimuskysymyksiin.

6.1 Case GitHub – Luotettavampi dokumentaatio ja API-validointi

GitHub on suosittu verkkopohjainen kehitysalusta, jota käytetään laajasti versionhallintaan, yhteistyöhön, automaatioon sekä projektin hallintaan. Kehittäjät voivat jakaa ja arvioida koodia sekä seurata koodimuutoksia Git-versionhallintajärjestelmän avulla. GitHubin dokumentaatio (docs.github.com) kattaa laajan valikoiman sisältöä alustan toiminnallisuuksista, joka oli aikaisemmin hajautettu kahdelle staattisille sivustoille. Vuonna 2020 dokumentaatio yhdistettiin dynaamiseen sovellukseen, jossa JSON:ia käytetään sisällön tuottamiseen ja järjestelmän sisäiseen viestintään. Ilman JSON-skeemojen käyttöä ei ollut keinoa varmistaa, että järjestelmässä tapahtuvat muutokset eivät rikkoisi dokumentaatiota tai aiheuttaisi virheitä. (Berman, 2023)

Ratkaisuksi tähän GitHubin kehittäjät ottivat käyttöön JSON-skeemat validoidakseen kaikki JSON-tiedostot, sovelluksen kontekstidatan ja API-pyyntöjen sisällöt. Jokaisen datamallin muutoksen yhteydessä skeemoja päivitetään, mikä varmistaa, että sovellus toimii oikein jokaisen muutoksen myötä. Tätä validointia käytetään kolmella tavalla. Ensimmäinen tapa on API-kutsujen validointi: jokainen tuotantoympäristössä suoritettu API-pyyntö tarkistetaan JSON-skeemaa vasten ennen datan tallentamista. Toinen tapa on automaatio: kun ulkoista dataa muunnetaan JSON-muotoon, se validoidaan ennen tuotantoon siirtymistä. Kolmas tapa on jatkuva integraatio (CI/CD): skeemat varmistavat, että mm. YAML- ja JSON-tiedostot ovat oikein muodostettuja ja virheettömiä ennen tuotantoon siirtymistä. (Berman, 2023)

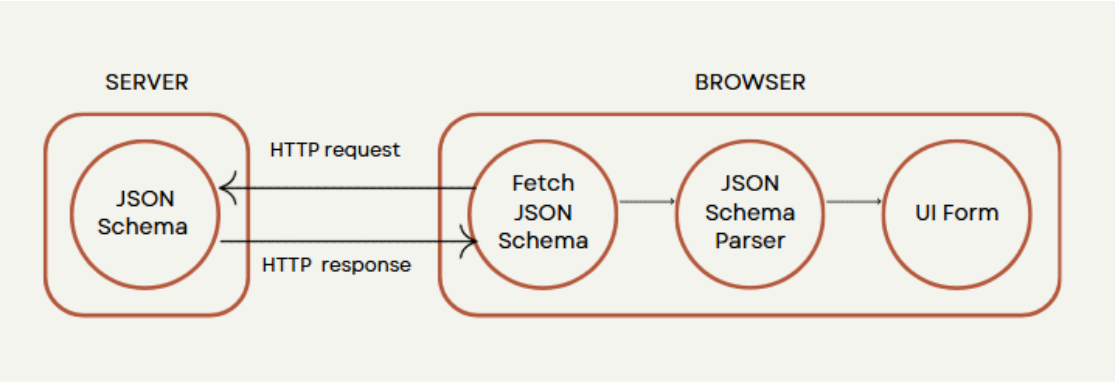
JSON-skeemat ovat parantaneet tuottavuutta, löydettävyyttä ja luotettavuutta. Kehittäjät voivat nopeasti hahmottaa JSON-datan rakenteen ilman manuaalista tarkastelua. Jatkuvan integraation testit varmistavat, että muutokset eivät aiheuta ongelmia ennen tuotantoon siirtymistä. GitHubin dokumentaatiotiimi julkaisee muutoksia yli 20 kertaa päivässä, ja JSON-skeemat ovat kriittinen osa järjestelmän toiminnan varmistamista. (Berman, 2023)

6.2 Case Remote.com – Skaalautuva ja virheetön lomakehallinta

Remote.com käsittelee työsopimus- ja palkkalomakkeita yli 100 maassa, joissa jokaisessa on erilaiset säädökset ja vaatimukset. Alkuun käytettiin kovakoodattuja lomakkeita, mutta liiketoiminnan laajetessa, niiden hallinta ja skaalaaminen osoittautui haastavaksi. Esimerkiksi eri maiden työajat ja viikonloput vaihtelevat merkittävästi. Joissakin maissa viikonloppu on perjantai ja lauantai, ja perinteisen klo 9–17 työaikakäytännön lisäksi on monia muita työaikamuotoja. Nämä vaikuttavat palkanmaksuun ja siksi virheelliset tiedot työaikakäytännöistä voivat aiheuttaa suuria ongelmia. Haasteita aiheutti myös kansainvälisten säädöksellisten vaatimusten jatkuva muutos, mikä teki työsuhdetietojen ja vaatimusten hallinnasta entistä monimutkaisempaa (Hutton, 2023)

Remote päätti ottaa JSON-skeemat käyttöön lomaketietojen validointiin sekä lomakkeiden dynaamiseen luomiseen. Dynaamiset lomakkeet ovat joustavia lomakkeita, jotka rakennetaan ohjelmallisesti JSON-skeemojen määrittelemien sääntöjen perusteella. Näin voidaan hallita erilaisia tietovaatimuksia yhden päätepisteen kautta, riippuen esimerkiksi siitä, mistä maasta, työsuhteesta tai muista tiedon ominaisuuksista on kyse. JSON-skeemojen avulla lomakkeiden hallinta siirrettiin palvelimelle, jolloin samat validointisäännöt voitiin jakaa sekä palvelimen että käyttöliittymän välillä, mikä vähensi JavaScript-koodin määrää sekä duplikaatteja (Kuva 3). JSON-skeemavalidoinnin avulla varmistettiin, että lomakkeiden kautta tuleva data täyttää lakisääteiset ja liiketoiminnalliset vaatimukset. (Hutton, 2023)

Kuva 3. JSON-skeeman integrointi palvelimen ja käyttöliittymän välillä (mukaillen Hutton, 2023).



Lisäksi Remote kehitti oman lomakegeneraattorin, joka perustuu JSON-skeemaan ja mahdollistaa räätälöityjen avainsanojen lisäämisen, joilla voi mukauttaa lomakekenttien esitystapaa. Tämä työkalu on avattu lähdekoodina osoitteessa @remoteoss/json-schema-form. Remote API mahdollistaa tietojen käsittelyn myös kolmansien osapuolten kautta. API:n avulla kaikki lomakkeet, jotka kommunikoivat Remote:n järjestelmien kanssa, hyötyvät JSON-skeemojen joustavuudesta ja yhteensopivuudesta. Vaikka JSON-skeema ei ole virallinen standardi, sen hyväksyntä ja käyttöönotto suurten toimijoiden keskuudessa lisää luottamusta teknologian pitkäaikaiseen käyttöön. (Hutton, 2023)

JSON-skeeman käyttöönotto toi merkittäviä hyötyjä. Lomakkeiden luontiaika uusille maille lyheni viikoista päiviin. Virheilmoitukset ja tukipyynnöt vähenivät, koska validointi on yhdenmukainen palvelimen ja käyttöliittymän välillä. Sisäiset tiimit voivat luoda ja muokata lomakkeita ilman kehittäjän apua. Tietoturva parani, koska validointi tapahtuu palvelimella ennen datan käsittelyä. (Hutton, 2023)

6.3 Case Cookpad Mart – Dynaamiset lomakkeet ja parempi tietojen eheys

Cookpad Mart on japanilainen tuoreisiin elintarvikkeisiin keskittyvä ruokaverkkokauppa-alusta, jota ylläpitää Cookpad Inc. Se on laajentanut toimintaansa kansainvälisille markkinoille, kuten Isoon-Britanniaan, Brasiliaan, Libanoniin, Intiaan ja Taiwaniin. Jälleenmyyjät ja paikalliset tuottajat toimivat Cookpad Mart -alustalla myyjinä. Tuotteiden noutopisteinä toimivat jääkaapit on sijoitettu esimerkiksi lähikauppoihin, apteekkeihin, rautatieasemille ja asuinkerrostaloihin. Käyttäjät voivat tilata tuotteita sovelluksen kautta ja noutaa ne jääkaapista, johon tilaus on toimitettu. (Hutton, 2021; Shiode, 2021)

Cookpad Mart kehitti myyjille tarkoitettua hallintanäkymää, joka sisälsi toimintoja tuotteiden rekisteröintiin ja päivittäisten toimitusten hallintaan. Tuotteen rekisteröinnin yhteydessä tulee ilmoittaa perustietoja, kuten nimi, kuva ja hinta, sekä laadunvarmistukseen ja elintarvikemerkintöihin liittyviä tietoja. Näihin kuuluvat esimerkiksi viimeinen käyttöpäivä sekä tieto siitä, onko tuote sulatettu. Vaadittavat tiedot vaihtelevat tuotteen tyypin ja tilan mukaan. Jos myyjälle näytetään kaikki mahdolliset lomakekentät samanaikaisesti, on hankalaa arvioida, mitkä niistä tulisi täyttää. Esimerkiksi perunaa myytäessä viimeistä käyttöpäivää ei tarvitse ilmoittaa, mutta suositeltu käyttöaika on hyvä mainita. Sen sijaan valmiille kastikkeelle, jossa on viimeinen käyttöpäivä, on annettava myös säilyvyysaika, joka takaa laadun lähetyspäivästä alkaen. Lisäksi tuore- ja prosessoidulle kalalle tulee ilmoittaa, onko se kasvatettua, raakana syötäväksi tarkoitettua tai aiemmin pakastettua. Koska eri tuotetyypeille tarvitaan erilaisia tietoja, myyjien oli hankalaa arvioida manuaalisesti, mitkä kentät heidän tulisi täyttää. Tarvittiin mekanismi, joka mukauttaisi lomakkeet tuotetyypin ja -tilan mukaan. Pelkkä käyttöliittymässä tehty lomakekenttien suodatus ei riittänyt estämään virheellisten tietojen tallentamista taustajärjestelmään. Oli toivottavaa, että tuotteet voitaisiin rekisteröidä suoraan taustajärjestelmään siten, että niiden tiedot on jo tarkistettu etukäteen. Näin säästettäisiin aikaa ja vältettäisiin virheiden syntymistä tarkistusprosessissa. (Shiode, 2021) Shiode (2021) mainitsee, että tiimi harkitsi räätälöidyn JavaSript koodin käyttöä monimutkaisten lomakkeiden luomiseen, mutta he olivat huolissaan siitä, että validointi ei olisi yhdenmukaista sekä selain- että palvelinpuolella. Yksi ohjelmistokehityksen periaatteista on DRY (”Dont Repeat Yourself” - älä toista itseäsi) eli saman logiikan moninkertainen kirjoittaminen tulisi välttää. Ratkaisuksi päätettiin luoda yhteinen skeema. JSON-skeema valittiin, koska se kykenee tehokkaasti renderöimään ja validoimaan monimutkaisia lomakkeita.

JSON-skeeman käyttöönotto paransi tuotteiden rekisteröinnin tarkkuutta sekä käyttäjäkokemusta. Se myös vähensi merkittävästi tuotetarkastukseen kuluvaa aikaa ja resursseja. JSON-skeeman vaikutukset ilmenivät useina parannuksina alustan toiminnassa. Ensinnäkin muutostarve väheni, koska vaatimusten muuttuessa riitti, että päivittää JSON-skeema määrittelyä. Validointi pysyi yhdenmukaisena sekä käyttöliittymässä että taustajärjestelmässä. Toiseksi tietojen eheys parani, sillä lomakkeiden käsittely ja tietojen tarkistus tehostuivat sekä käyttöliittymässä että taustajärjestelmässä, mikä vähensi virheellisten tietojen rekisteröintiä. Kolmanneksi tuotesyöttö helpottui, kun tuotetietojen täyttäminen selkeytyi ja virheiden määrä minimoitu. (Hutton, 2021) Shioden (2021) mukaan keskeiset tulokset JSON-skeeman käyttöönoton jälkeen olivat seuraavat. Virheellisten lomaketäyttöjen määrä laski 10 prosentista nollaan. Aiemmin noin 10 % uusista tuotteista sisälsi puutteellisia laadunvarmistustietoja, mutta

JSON-skeeman käyttöönoton jälkeen puutteita ei enää ollut. Lisäksi hän arvioi, että validoinnin toteuttamiseen kuluva aika puolittui JSON-skeeman ansiosta.

Ohjelmistokehittäjät ajattelivat käyttää samaa logiikkaa käyttöliittymällä ja palvelinpuolella, sekä luoda oman JSON-rakenteen, mutta sitten huomasivat joutuvansa kirjoittamaan useita ”if”-lausekkeita. Yksi ohjelmistokehityksen periaatteista on DRY (”Dont Repeat Yourself” - älä toista itseäsi) eli saman logiikan moninkertainen kirjoittaminen tulisi välttää. JSON-skeeman avulla tietovaatimukset määritellään vain kerran, minkä jälkeen samaa määrittelyä voidaan hyödyntää automaattisesti sekä käyttöliittymässä että palvelinpuolella. Kun samaa logiikkaa ei tarvitse toistaa, käyttöliittymän ja palvelimen eroavaisuudet vähenevät. (Hutton, 2021) Liitteessä 2 on esimerkki Cookpad Martin tuoterekisteröinnin JSON-skeeman määrittelystä. Se havainnollistaa, miten lomakerakenne mukautuu eri tuotetyyppien vaatimuksiin ja miten JSON-skeemaa hyödynnetään tietojen validoinnissa ja lomakkeiden dynaamisessa muodostamisessa.

6.4 Case 6 River Systems – Tehokkaampi yhteistyö ja ajansäästö

6 River Systems on noussut toimitusprosessien hallinnan alalla johtavaksi toimijaksi tekoälyn ja robotiikan avulla. Yhdysvalloissa, Kanadassa ja Euroopassa toimivat yritykset käyttävät heidän ratkaisujaan optimoidakseen toimitusketjujaan ja käsitelläkseen miljoonia tilauksia viikoittain. Nopean kasvun myötä 6 River Systems kohtasi tyypillisiä haasteita organisaation, teknologian ja toiminnan kehittämisessä. Varastoasiakasmäärien kymmenkertaistuessa ja tiimien laajentuessa kasvoi tarve selkeälle viestinnälle ja yhdenmukaisille toimintamalleille, jotta kaikki osapuolet, myös toimitusjärjestelmä, pysyisivät yhteensopivina ja koordinoituina. (Gutermuth & Hutton, 2023)

6 River Systems hyödyntää robotiikkaa ja laitteistoja, jotka eivät aina toimi saumattomasti yhteen. Haasteena oli huippuluokan tietojenkäsittely-, tallennus- ja analytiikkatyökalujen integrointi, sillä esimerkiksi robottien käyttämä Robot Operating System (ROS) viitekehys ei ollut yhteensopiva modernien analytiikkatyökalujen, kuten Elasticsearch ja BigQuery kanssa. Nämä ovat nykyaikaisia tehokkaita työkaluja suurten tietomassojen hallintaan ja analysointiin. Lisäksi yleisin ROS-versio perustuu vanhentuneeseen Python 2.7. versioon. Nopea kasvu kerrytti teknistä velkaa, ja sen hallinta vaati kestävien ratkaisujen ja pitkäjänteisten toimintamallien kehittämistä. (Gutermuth & Hutton, 2023)

6 River Systems ratkaisi useita teknisiä ja organisatorisia ongelmia laajentamalla JSON-skeeman käyttöään. JSON-skeema toimii yrityksen automaation perustana, mahdollistaen datan yhdenmukaisuuden ja laadun parantamisen. Keskeiset hyödyt JSON-skeeman käytöstä ovat seuraavat (Gutermuth & Hutton, 2023): Tehokkaampi sisäinen viestintä: Analyytikot ja kehittäjät voivat yhdessä määritellä uusia skeemoja, jolloin tuotantokelpoinen määrittely on heti kaikkien ymmärrettävissä. Luotettavammat ohjelmistokehityspaketit (SDK, Software Development Kit): JSON-skeeman pohjalta voidaan luoda TypeScript-rajapintoja, mikä vähentää virheitä jo ennen tuotantoa. Automaattiset tietokantapäivitykset: Uuden skeeman julkaisun yhteydessä järjestelmä luo tai päivittää tietokantataulut automaattisesti, varmistaen tietojen eheän käsittelyn. Datan jatkuva validointi: Kaikki JSON-skeemaan sopimattomat tapahtumat ohjataan virhetauluun ja niistä ilmoitetaan virheviestillä. Parempi järjestelmän seuranta ja diagnostiikka: JSON-skeema mahdollistaa tapahtumatason nimeämiskäytäntöjen käyttämisen kaikissa mittaus- ja seurantajärjestelmissä. Tämän ansiosta voidaan tunnistaa tarkat suorituskykytiedot sekä käytössä oleva versio, jota järjestelmä käsittelee. Mahdolliset ongelmat voidaan havaita välittömästi graafisessa näkymässä, mikä nopeuttaa vianmääritystä. Automaattinen metatietojen hallinta: JSON-skeemat toimivat pohjana datan hallinta- ja hakutyökaluille, kuten datakatalogeille. Datakatalogi on metatiedon kokoelma, joka toimii jäsenneltynä varastona kaikista organisaation tietoresursseista. Sen avulla, järjestelmän ei tarvitse erikseen kerätä ja tulkita metatietoja. dbt-integraatio ja analytiikan automatisointi: JSON-skeema auttaa automatisoimaan lähdeaineiston luontia dbt:ssä, vähentäen manuaalista työtä. dbt (Data Build Tool) on suosittu avoimen lähdekoodin komentorivityökalu, joka mahdollistaa tietovarastojen tehokkaan muuntamisen ja hallinnan. (Gutermuth & Hutton, 2023)

JSON-skeeman käyttöönotto on tuonut merkittäviä ajansäästöjä koko tekniseen ekosysteemiin. Tyyppiturvallisuus ja automaatio vähentävät virhetilanteita ja nopeuttavat kehitystä. JSON-skeema on mahdollistanut varmennetun datankeruun ja instrumentoinnin tuhansille roboteille, paremman yhteistyön TypeScript-, Python- ja SQL-osaajien välillä, jokaisen datalähetyksen validoinnin ennen tallentamista sekä automatisoidun metatiedon hallinnan. Lisäksi organisaation kasvaessa JSON-skeeman käyttö yhteisenä kielenä on parantanut datan laatua ja näkyvyyttä kaikilla tasoilla tukien kestäviä käytäntöjä. (Gutermuth & Hutton, 2023)

JSON-skeeman käytön seurauksena 6 River Systems on voinut ottaa käyttöön työkaluja, kuten Quicktype ja TypeBox palvelinpuolella sekä JSON-skeemapohjaisia validointityökaluja käyttöliittymäpuolella. Yhdistämällä nämä logiikat heidän tietovarastonsa

pysyvät ajan tasalla. 6 River Systems harkitsi myös Protobufia ja Apache Avroa, mutta ne eivät soveltuneet heidän ympäristöönsä yhtä hyvin kuin JSON-skeema. JSON oli jo valmiiksi käytössä käyttöliittymäkehityksessä, se integroituu saumattomasti Node.js- ja TypeScript-ekosysteemiin ja sen käyttöönotto ei vaatinut suuria muutoksia olemassa olevaan koodipohjaan. JSON-skeema on mahdollistanut skaalautuvan ja automaattisen tiedonhallinnan koko organisaatiossa. Se on parantanut datan laatua, tehostanut kehitysprosessia ja vähentänyt inhimillisiä virheitä. 6 River Systemsin käyttökokemuksen perusteella JSON-skeema on ratkaisu, joka mahdollistaa nopean kasvun ilman, että järjestelmät muuttuvat hallitsemattomaksi. (Gutermuth & Hutton, 2023)

6.5 Case Heroku – Optimoitu API-testauksen ja dokumentoinnin prosessi

Heroku on suosittu Platform as a Service (PaaS) pilvipalvelualusta, joka tarjoaa ohjelmistokehittäjille helpon tavan julkaista, hallita ja skaalata moderneja sovelluksia. Herokun avulla kehittäjät voivat keskittyä sovellusten kehittämiseen ilman huolta infrastruktuurin hallinnasta. (Heroku, n.d.-a) Palvelun ydinteknologia, dyno-järjestelmä, perustuu konttiteknologiaan, jossa sovellusprosessit suoritetaan kevyissä ja eristetyissä Linux-konteissa. Nämä kontit tarjoavat sovellukselle tarvittavan laskentatehon, muistin, käyttöjärjestelmän ja tilapäisen tiedostojärjestelmän. Dynot skaalautuvat automaattisesti sovelluksen resurssitarpeiden mukaan, mikä mahdollistaa joustavan ja luotettavan sovellusten kokoamisen ja ajamisen. (Heroku, n.d.-b)

Herokulla on useita julkisia API-päätepisteitä (API-endpoints), jotka tulee testata huolellisesti ja dokumentoida API:n käyttäjien varten. Yksi tärkeimmistä asioista julkisen API:n ylläpidossa on varmistaa, että dokumentaatio pysyy aina ajan tasalla. Heroku hyödyntää julkisten API-päätepisteidensä dokumentoinnissa JSON-skeemaa ja on kehittänyt prosessin, joka varmistaa skeematiedostojen ajantasaisuuden. Herokun insinööritiimi käyttää jatkuvan integraation (CI) Rake-tehtävää, joka on Ruby-projekteissa käytettävä automaatiotyökalu. Rake-tehtävä tarkistaa, onko nykyisessä kehitysversiossa tehty muutoksia JSON-skeematiedostoihin. Jos muutoksia havaitaan, ne päivitetään automaattisesti Herokun API-dokumentaatioon Ruby-pohjaisen prmd-työkalun avulla. Tämä automatisointi varmistaa, että asiakkaat ja API-käyttäjät saavat ajankohtaista tietoa. (Young, 2024)

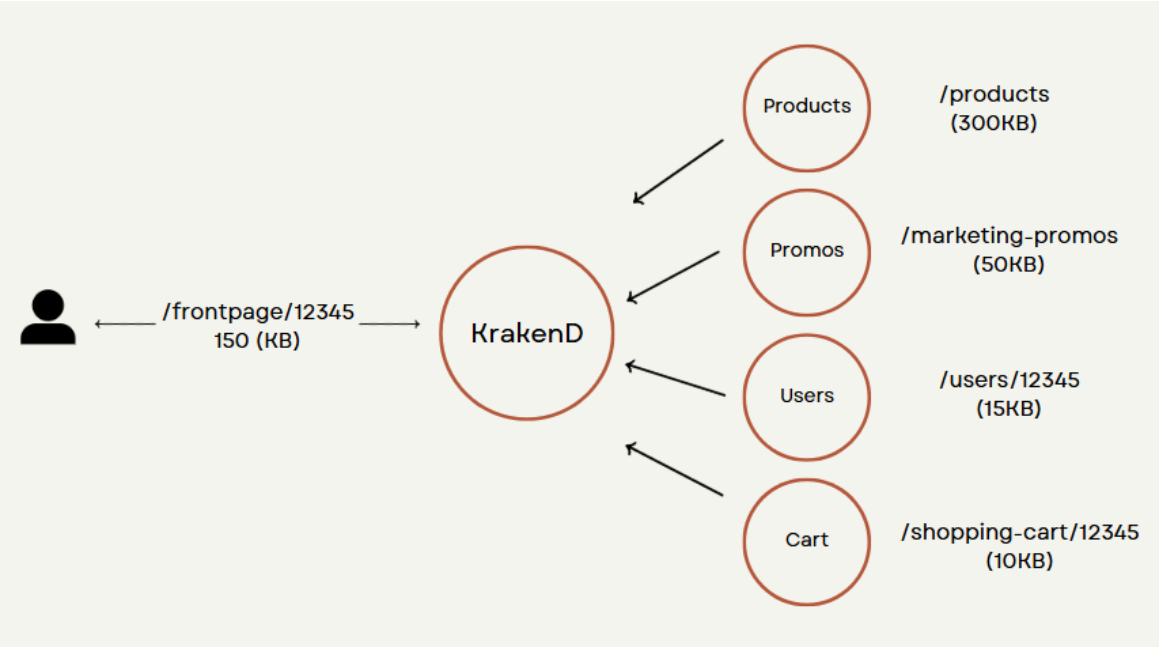
Heroku on kehittänyt API-testausprosessin, joka varmistaa julkisten API-päätepisteiden toiminnan tarkistamalla jokaisen tuotantoympäristössä suoritetun API-pyynnön ja -

vastauksen JSON-skeemaa vasten ennen datan tallentamista. Tähän käytetään Rubyn committee-työkalua. Mikäli testit eivät mene läpi, päivitetään skeemoja tai API-päätepisteet vastaamaan niitä. Heroku on hyödyntänyt JSON-skeemaa myös ohjelmointivirheiden tunnistamiseen. Esimerkiksi testausprosessissa havaittiin, että dynon käyttöarvo saattoi palautua merkkijonona, vaikka sen olisi pitänyt olla numero. Tämä ongelma auttoi tunnistamaan, että JSON-jäsennys ei toiminut odotetulla tavalla. Virheen tunnistaminen johti tarvittaviin korjauksiin, jotka paransivat API:n luotettavuutta ja suorituskykyä. JSON-skeematiedostojen käyttö on mahdollistanut Herokulle tehokkaan API-testauksen ja dokumentoinnin, parantaen näin ohjelmistokehityksen prosessia ja varmistamalla, että API on käyttäjilleen luotettava ja ajantasainen (Young, 2024).

6.6 Case KrakenD – JSON-skeeman hyödyntäminen API-yhdyskäytävässä

KrakenD on korkean suorituskyvyn avoimen lähdekoodin API-yhdyskäytävä (API-gateway), joka toimii liitäntänä useiden mikropalveluiden välillä hoitaen automaattisesti raskaat työtehtävät, kuten aggregointi, muuntaminen, suodatus, purkaminen, rajoittaminen, autentikointi. KrakenD mahdollistaa useiden API-päätepisteiden käytön yhdellä kutsulla. Se ryhmittelee, validoi, pakkaa, muuntaa ja tiivistää API-pyyntöjä ja vastauksia sekä tukee lukuisia liitännäisiä ja välikohtia, joiden avulla voi laajentaa toiminallisuutta, kuten OAuth2-todennusta, turvallisuuskerroksia, katkaisijatoimintoa ja nopeusrajoituksia. (KrakenD, 2018) Kuva 4 havainnollistaa mobiiliapplikaatiota, jossa data kulkee käyttöliittymän ja useiden palvelinpään palveluiden välissä KrakenD API-yhdyskäytävän avulla.

Kuva 4. API-yhdyskäytävä liitäntänä useiden mikropalveluiden välillä (mukaillen KrakenD, 2018).



Koska API-yhdyskäytävä on keskeinen osa API-hallintaa, sen luotettavan toiminnan, tietojen eheyden ja sujuvien rajapintayhteyksien takaaminen on ensiarvoisen tärkeää. JSON-skeema on osoittautunut keskeiseksi työkaluksi tämän varmistamisessa ja parantunut merkittävästi sekä KrankenD kehittäjien että käyttäjien kehittäjäkokemusta. Lombarten (2023) mukaan KrakenD käyttää JSON-skeemaa muun muassa seuraaviin viiteen käyttötarkoitukseen: API-pyyntöjen validointi ajonaikaisesti, API-vastausten varmistaminen ennen niiden palauttamista käyttäjille, konfiguraatiotiedostojen validointi käyttöönotto- ja rakennusprojekteissa, ohjelmiston dokumentointi sekä end-to-end testien määrittelyn kirjoittaminen.

KrakenD varmistaa API-pyyntöjen ja -vastausten oikeellisuuden validoimalla ne JSON-skeemaa vasten jo API-yhdyskäytävässä. Tämä estää virheellisten tai puutteellisten pyyntöjen etenemisen palveluihin ja vähentää turhaa kuormitusta taustajärjestelmissä. Skeemavalidointi tarkistaa, että saapuvat pyynnöt sisältävät kaikki tarvittavat tiedot oikeassa muodossa ja että vastaukset noudattavat sovittua rakennetta. Näin järjestelmä pysyy luotettavana ja API-käyttäjät voivat luottaa siihen, että vastaanotettu data on yhtenäistä ja helposti käsiteltävää. Mikäli jokin pyyntö tai vastaus ei täytä määriteltyjä vaatimuksia, KrakenD hylkää sen jo ennen sen jatkokäsittelyä ja palauttaa selkeän virheilmoituksen. (Lombarte, 2023)

KrakenD käyttää JSON-skeemaa monin tavoin konfiguraatioiden hallinnassa. JSON-skeeman sisällyttäminen konfiguraatiotiedostoihin parantaa niiden selkeyttä ja varmistaa oikeellisuuden, mikä helpottaa järjestelmän ylläpitoa ja kehittämistä. Kun lisäksi otetaan käyttöön JSON-skeemaan perustuva *krakend check --lint* -tarkistus CI/CD-putkessa, konfiguraatiotiedostot validoidaan automaattisesti osana ohjelmiston julkaisuprosessia. Näin mahdolliset konfiguraatiovirheet keskeyttävät julkaisun. KrakenD hyödyntää JSON-skeemaa myös dokumentaation luomisessa. Sen kehitystiimi on integroinut JSON-skeemat suoraan dokumentaatioprosessiin. Tämä mahdollistaa konfiguraatiotietojen rakenteen ja kenttien esittämisen suoraan skeeman pohjalta, mukaan lukien kenttien kuvaukset, esimerkit, oletusarvot, enum-arvot ja JSON-skeeman haku-URL:in. Tämä automatisointi varmistaa, että dokumentaatio pysyy ajan tasalla eikä sisällä vanhentunutta tai virheellistä tietoa. (Lombarte, 2023)

Lisäksi JSON-skeema tukee KrakenD:n testausprosessia end-to-end testauksessa. Kaikkia testitapausten arvoja ei aina voida tarkasti testata. Esimerkiksi jos API-pyyntöä tehdään päätepisteelle kahdesti, tulokset voivat vaihdella, jos pyynnössä on päivämääräkenttä, joka muuttuu jokaisen suorituksen aikana. Vaikka tarkkoja arvoja ei voida aina testata, voidaan kuitenkin varmistaa, että API-vastaus noudattaa määriteltyä JSON-skeemaa ja sen asettamia tietotyyppivaatimuksia. KrakenD:n tapa hyödyntää JSON-skeemaa osoittaa sen monipuolisuuden API-yhdyskäytävien hallinnassa. Sen avulla voidaan varmistaa tietojen eheys, validoida konfiguraatiotiedostoja, optimoida dokumentaatio ja automatisoida testausprosesseja, mikä parantaa järjestelmän luotettavuutta ja ylläpidettävyyttä. (Lombarte, 2023)

6.7 Sisällönanalyysi – tulosten yhteiset trendit

Tapaustutkimukset osoittivat, kuinka kuusi organisaatiota – GitHub, Remote.com, Cookpad Mart, 6 River Systems, Heroku ja Krakend – oli hyödyntänyt JSON-skeemojen implementointia parantaakseen ohjelmistokehityksen luotettavuutta, virheettömyyttä ja tietojen eheyttä. Kaikki tarkastellut organisaatiot olivat saavuttaneet merkittäviä etuja JSON-skeemojen käyttöönotosta, erityisesti datan validoinnin, dynaamisten lomakkeiden ja prosessien automatisoinnin osalta. Lisäksi organisaatioiden kasvaessa ja digitaalisten palveluiden ja ohjelmistojen monimutkaistuessa JSON-skeemat olivat toimineet ratkaisevana tekijänä järjestelmien hallittavuuden ja ylläpidettävyyden varmistamisessa. Nämä havainnot tukevat myös teoriaosuuden näkemyksiä JSON-skeemojen merkityksestä ohjelmistokehityksessä. Vaikka tapauskohtaiset tulokset vaihtelivat, useat yhteiset teemat nousivat esiin auttaen tunnistamaan trendejä ja vaikutuksia:

Datan validointi ja virheiden väheneminen oli kaikkien tapaustutkimusten perusteella JSON-skeemojen merkittävin hyöty organisaatioille, erityisesti API-rajapinnoissa ja lomakepohjaisessa tiedonsyötössä. JSON-skeemat varmistavat datan eheyden määrittelemällä selkeät säännöt sille, millaista dataa voidaan tallentaa ja käsitellä, esimerkiksi määrittelemällä rakenteen ja tyypit, pakolliset kentät sekä arvojen validoinnin, kuten merkkijonojen minimipituuden, sallitut arvot ja numeeriset rajat. Organisaatiot onnistuivat minimoimaan virheet tarkistamalla API-pyynnöt JSON-skeemaa vasten ennen datan tallentamista, mikä paransi koko järjestelmän luotettavuutta.

JSON-skeemat mahdollistivat yhdenmukaisen validoinnin; samat validointisäännöt pätevät niin käyttöliittymällä sekä palvelimella, mikä vähentää koodin määrää, duplikaatteja ja sitä kautta virheilmoitusten sekä tukipyyntöjen määrää. Tämä yhdenmukainen määrittely parantaa ohjelmiston ylläpidettävyyttä. Remote.com ja Cookpad Mart hyödynsivät JSON-skeemoja virheiden minimoimiseksi dynaamisissa lomakkeiden luomisessa ja tietojen syöttämisessä. Remotella JSON-skeeman käyttöönoton jälkeen virheellisten lomaketäyttöjen määrä laski 10 prosentista nollaan. Tietojen eheys oli keskeinen tavoite monen organisaation toiminnassa, ja JSON-skeemojen avulla voitiin varmistaa, että tiedot olivat tarkkoja ja luotettavia ennen niiden siirtymistä tuotantoon.

Järjestelmien skaalautuvuus ja joustavuus olivat ratkaisevia tekijöitä monimutkaisten järjestelmien hallinnassa organisaatioissa, kuten Remote.com, 6 River Systems ja Heroku. JSON-skeemat soveltuvat erinomaisesti dynaamisten lomakkeiden luomiseen ja tarjoavat joustavan ja ohjelmallisesti validoitavan tavan määritellä lomakekenttien rakenne, sallitut

arvot ja riippuvuudet. JSON-skeemat mahdollistavat skeemojen yhdistämisen muihin aliskeemoihin erilaisin ehtoperustein ja viittein. Skeemapohjainen lähestymistapa tukee automaattista lomakkeiden generointia, validointia ja mukautumista käyttäjän syötteiden perusteella, mikä tekee niistä erityisen tehokkaita laajoissa ja monimutkaisissa järjestelmissä.

Tapaustutkimuksissa havaittiin, että JSON-skeemat olivat mahdollistaneet sujuvan laajentumisen kansainvälisiin ympäristöihin sekä tehneet monimutkaisten palveluiden ja järjestelmien hallinnan mahdolliseksi minimoiden virheet. Remotella lomakkeiden rakentaminen dynaamisesti ohjelmallisesti lyhensi lomakkeiden luontiajan uusille maille viikoista päiviin ja paransi järjestelmän skaalautuvuutta. 6 River Systems puolestaan pystyi hallitsemaan suuria tietomassoja ja varmistamaan, että järjestelmä pystyi skaalautumaan kasvavan organisaation tarpeiden mukaan.

Jatkuvan integraation ja käyttöönoton (CI/CD) rooli ohjelmistokehityksessä korostui erityisesti GitHubin ja Herokun tapauksissa. JSON-skeemojen käyttö CI/CD-putkessa varmisti, etteivät muutokset riko dokumentaatiota tai järjestelmän toimintaa. Tämä paransi ohjelmistokehityksen luotettavuutta ja ylläpidettävyyttä useissa organisaatioissa erityisesti, kun virheitä estetään ennen tuotantoon siirtymistä. KrakendD:n tapaustutkimuksessa JSON-skeemaa käytettiin konfiguraatiotiedostojen validoinnin automatisointiin CI/CD-putkessa, mikä takasi tiedostojen oikeellisuuden ja esti virheet julkaisuprosessissa. Lisäksi JSON-skeemat voitiin integroida **automaatiotesteihin**, joiden avulla varmistettiin, että API-palveluiden JSON-vastaukset noudattivat odotettua rakennetta, mikä paransi API:n luotettavuutta ja vakautta.

Sujuvampi yhteistyö ja dokumentaatio parani JSON-skeemojen ansiosta useissa organisaatioissa. JSON-skeemat helpottivat tiimien välistä yhteistyötä ja paransivat kommunikointia eri osapuolten välillä. 6 River Systems:in toiminnot ja tiimit kasvoivat merkittävästi, mikä lisäsi tarvetta selkeälle viestinnälle ja yhteisille toimintamalleille myös toimitusjärjestelmän kanssa. JSON-skeemat mahdollistivat tehokkaamman sisäisen viestinnän; analyytikot ja kehittäjät pystyivät yhdessä määrittelemään uusia skeemoja, jolloin tuotantokelpoinen määrittely oli heti kaikkien ymmärrettävissä. JSON-skeemat toimivat itsessään dokumentaationa, sillä ne määrittelevät tarkasti tietomallin rakenteen, validointisäännöt ja sallitut arvot. Tämä vähentää erillisen dokumentoinnin tarvetta ja helpottaa kehittäjien työtä, vähentää kommunikaatiohaasteita ja nopeuttaa uusien tiimin jäsenten perehtymistä. Kuten myös teoriaosuudessa todettiin, JSON-skeemat voivat toimia sopimuksena datan tarjoajien ja kuluttajien välillä. Ne määrittelevät selkeästi datan

rakenteen ja sallitut arvot, mikä vähentää väärinkäsityksiä ja varmistaa datan yhteensopivuuden. GitHub, Heroku ja KrakenD hyödynsivät JSON-skeemaa API-dokumentaation luomisessa ja sen ajantasaisuuden varmistamisessa. Täsmälliset ohjeet API:n toiminnasta ja sen odotetusta datasta ovat oleellisia ohjelmistokehittäjille.

Lakisääteiset ja liiketoiminnalliset vaatimukset toivat mukanaan erityisiä vaatimuksia tiedon käsittelylle, joihin skeemapohjainen validointi vastasi tarjoamalla selkeän ja automatisoidun tavan vaatimusten hallintaan. Remoten tapauksessa JSON-skeemat tarjosivat keinon hallita yli sadan eri maan työsuhdetietojen nopeasti muuttuvia ehtoja, jotka määräytyvät kunkin maan lainsäädännön ja vaatimusten mukaan. JSON-skeemojen avulla voidaan määritellä tarkat säännöt ja rajoitukset sekä seurata versiohistoriaa. Lisäksi JSON-skeemojen avulla voidaan varmistaa säädösmuutosten hallinta, sillä ne mahdollistavat vaatimusten mukauttamisen ja päivitysten seuraamisen tarkasti. Tämä on erityisen tärkeää toimialoilla, joilla on tiukat vaatimukset tiedonhallinnalle ja säädösten noudattamiselle.

Datan muuntaminen ja ETL-prosessit tehostuivat JSON-skeeman rakenteen avulla. Tapaustutkimuksissa JSON-skeemaa käytettiin varmistamaan, että muunnettu data säilytti johdonmukaisen rakenteen eri järjestelmien välillä. Tämä on keskeistä integraatioiden, tiedonsiirron ja analytiikan kannalta. GitHub hyödynsi skeemoja varmistaakseen, että JSON-tiedostot, kontekstidata ja API-pyynnöt olivat oikein ja yhdenmukaisia. Skeemojen päivitys aina datamallin muutosten yhteydessä takasi sovelluksen toimivuuden. Kun ulkoinen data muunnettiin lähdemuodosta JSON-muotoon, generoitu data validoitiin automaatioputkessa JSON-skeemaa vasten sen oikeellisuuden varmistamiseksi. Lisäksi jatkuvassa integraatiossa (CI/CD) skeemat takasivat, että muun muassa YAML- ja JSON-tiedostot olivat oikeassa rakenteessa ennen tuotantoon siirtymistä. Näin JSON-skeemojen käyttö voi varmistaa datan eheyden ja estää virheitä koko prosessin ajan, parantaen sovelluksen luotettavuutta ja tehokkuutta.

Vaikka edellä kuvatut trendit olivat yleisimpiä, tapaustutkimuksista nousi esiin myös muita merkittäviä etuja, jotka tukevat teoriaosuuden esittämiä näkökulmia. Esimerkiksi Cookpad Mart hyödynsi JSON-skeemoja osana käyttöliittymän saavutettavuuden parantamista, jolloin dynaamisesti luodut lomakkeet mukautuivat paremmin käyttäjien tarpeisiin. JSON-skeemat tukivat versionhallintaa ja mahdollistivat selkeän tavan hallita tietomallien muutoksia. Tietoturvan parantaminen mainittiin muun muassa Remote.com tapauksessa, ja miten skeemavalidointi palvelinpäässä esti haitallisten tai virheellisten datan pääsyn järjestelmiin. Lisäksi JSON-skeemat tehostivat kehitystyötä automaattisen validoinnin avulla, vähentäen manuaalista työtä ja nopeuttaen kehitysprosessia. Nämä yksittäiset

havainnot eivät olleet yhtä yleisiä kuin pääteemat, mutta ne korostavat JSON-skeemojen laajaa sovellettavuutta eri tilanteissa.

7 Johtopäätökset ja pohdinta

Tämä opinnäytetyö tarkasteli JSON-skeemojen roolia ohjelmistokehityksessä, erityisesti datan eheyden varmistamisessa ja monimutkaisten järjestelmien ylläpidossa. Tulokset osoittavat, että JSON-skeemat tarjoavat standardoidun ja tehokkaan tavan kuvata ja validoida JSON-dataa. Ne määrittelevät tarkkaan JSON-datan sisällön, rakenteen, tietotyypit ja rajoitteet. JSON-skeemojen käyttö vähentää olettamusten määrää, tehostaa kehitystyötä ja varmistaa datan yhdenmukaisuuden ja eheyden API-rajapinnoissa.

JSON-skeemat ovat erityisen hyödyllisiä API-rajapintojen datan validoinnissa, skeemapohjaisessa dokumentoinnissa ja dynaamisten lomakkeiden hallinnassa. Lisäksi niiden avulla voidaan hallita konfiguraatioita ja parantaa tietoturvaa. JSON-skeemat voivat myös toimia sopimuksena datan tarjoajan ja kuluttajan välillä tehostaen yhteistyötä eri osapuolten välillä. Mikropalveluarkkitehtuureissa ja integraatioissa JSON-skeemat mahdollistavat sujuvan ja luotettavan tiedonvaihdon.

Tutkimuksen perusteella voidaan todeta, että organisaatiot voivat parantaa tietojen eheyttä ja vähentää virheitä varhaisen skeemapohjaisen validoinnin avulla. Jokaisen API-pyynnön ja -vastauksen validointi JSON-skeemaa vasten ennen datan tallentamista, estää virheellisen datan pääsyn järjestelmiin sekä parantaa API-rajapintojen luotettavuutta. JSON-skeemojen integrointi automaatiotesteihin tukee reaaliaikaista validointia ja datan oikeellisuuden ja eheyden tarkastamista. Lisäksi JSON-skeemat mahdollistavat yhdenmukaisen validoinnin sekä käyttöliittymällä että palvelimella, mikä vähentää koodin määrää, duplikaatteja ja sitä kautta virheiden sekä tukipyyntöjen määrää.

Organisaatioiden kasvaessa sekä järjestelmien vaatimusten kasvaessa ja monimutkaistuessa JSON-skeemat edistävät järjestelmien hallittavuutta ja ylläpidettävyyttä. JSON-skeemat mahdollistavat modulaarisen kehitystyön, jossa erilaisten tunnisteiden, ehtoperusteiden ja viitteiden avulla voidaan yhdistää useita aliskeemoja. Näin ollen skeemat ovat uudelleenkäytettäviä ja helposti ylläpidettäviä. JSON-skeeman deklaratiivinen rakenne mahdollistaa monimutkaisten järjestelmien sekä joustavien dynaamisten lomakkeiden hallinnan. Tutkimus osoitti, että JSON-skeemat ovat lisänneet ohjelmistojen skaalautuvuutta, pitkäaikaista ylläpidettävyyttä ja tehneet uusien ominaisuuksien kehittämisestä nopeampaa ja virheettömämpää. JSON-skeemojen ansiosta useat organisaatiot ovat pystyneet sujuvasti laajentamaan toimintojaan kansainvälisille markkinoille sekä varmistamaan lakisääteisten ja liiketoiminnallisten vaatimusten noudattamisen.

Tutkimuksen luotettavuutta tukee se, että JSON-skeemat ovat laajasti käytettyjä ohjelmistokehityksessä ja niiden toiminta voidaan testata erilaisissa järjestelmäympäristöissä. Koodiesimerkit ja tapaustutkimukset osoittivat käytännössä, kuinka skeemapohjainen validointi ja dokumentaatio parantavat ohjelmistojen ylläpidettävyyttä. Tutkimuksen validiteettia tukee se, että analyysi perustui ajankohtaiseen teoriaan sekä alan käytäntöihin todellisissa kehitysympäristöissä. Näin ollen tutkimuksen tulokset ovat sovellettavissa ohjelmistokehityksen eri osa-alueilla.

Alkuperäinen tavoite oli toteuttaa opinnäytetyö toimeksiantona yritykselle, joka tarjoaa digitaalisia lakitietopalveluja sekä oikeudelliseen tiedonhallintaan liittyviä järjestelmäratkaisuja hyödyntäen JSON-skeemaa. Kuitenkin organisaatiomuutosten ja aikataulullisten haasteiden vuoksi tutkimus toteutettiin hyödyntäen muita tapaustutkimuksia. Tutkimus tuotti kattavan analyysin JSON-skeemojen merkityksestä ohjelmistokehityksessä. Tapaustutkimukset ja koodiesimerkit tukivat teoreettista tarkastelua. Haasteita aiheutti teorian rajaaminen, mutta lopputuloksena syntyi hyvin jäsennelty kokonaisuus.

Jatkokehityksenä voitaisiin syventyä tarkemmin johonkin tässä opinnäytetyössä käsiteltyyn aiheeseen, kuten JSON-skeemojen vaikutuksiin ohjelmistojen tietoturvaan. Esimerkiksi voitaisiin tutkia syvällisemmin, kuinka tehokkaasti skeemapohjainen validointi estää virheellisen tai haitallisen datan, kuten SQL-injektioiden ja muiden haitallisten hyökkäysten, pääsyn järjestelmiin. Tämä voisi sisältää käytännön testejä JSON-skeemojen kyvystä tunnistaa ja torjua tietoturvauhkia sekä analyysin siitä, millaisia parhaita käytäntöjä organisaatioiden tulisi noudattaa skeemojen suunnittelussa ja käyttöönotossa.

Opinnäytetyössä esitettyjä havaintoja voidaan soveltaa API-kehityksessä, ohjelmistojen validointiprosesseissa ja tiedonhallinnan tehostamisessa. Tulokset tarjoavat konkreettisia suosituksia siitä, miten JSON-skeemoja kannattaa käyttää datan eheyden varmistamiseksi ja järjestelmien ylläpidettävyyden parantamiseksi. Organisaatiot voivat hyödyntää tuloksia uusien API-ratkaisujen kehittämisessä ja nykyisten järjestelmien optimoinnissa.

8 Yhteenveto

Kaikkiin opinnäytetyön tutkimuskysymyksiin vastattiin onnistuneesti. Teoriaosuus tarjosi kattavan pohjan JSON-skeeman määrittelystä ja hallinnasta. JSON-skeeman validoinnin mekanismeja ja modulaarisuutta havainnollistettiin skeemaesimerkkien avulla. Tapaustutkimuksista kerätty data sekä laadullinen sisällönanalyysi täydensivät teoriaa tuoden esiin JSON-skeeman merkityksen todellisissa ohjelmistokehitysprojekteissa.

Opinnäytetyön aikana syvensin ymmärrystäni JSON-skeeman määrittelyyn ja soveltamiseen. Ennen opinnäytetyön aloittamista olin kokenut, että JSON-skeemoista on tietoa rajallisesti saatavilla, eikä aihe ollut erityisen keskeinen ohjelmistokehityksen keskusteluissa nykyisen työympäristöni ulkopuolella. Tutkimuksen edetessä ja aineistoa kerätessä kävi kuitenkin ilmi, että JSON-skeemat ovat laajasti käytössä monenlaisissa käyttötarkoituksissa. Erityisesti eri organisaatioiden tapaustutkimukset toivat esiin, kuinka JSON-skeemat tukevat ohjelmistokehityksen skaalautuvuutta, automatisoitua validointia ja lisäävät järjestelmien sekä kehitystyön tehokkuutta. Vaikka JSON-skeema ei ole virallinen standardi, sen hyväksyntä ja käyttöönotto suurten toimijoiden keskuudessa lisää luottamusta teknologian pitkäaikaiseen käyttöön.

Työn tuloksena syntyi selkeä kuvaus siitä, miten JSON-skeemat voivat parantaa datan validointia, varmistaa tietojen eheyttä sekä tehostaa API-kehitystä ja ohjelmistojen ylläpitoa. Tutkimuksen perusteella voidaan todeta, että JSON-skeemat tarjoavat merkittäviä etuja erityisesti API-kehityksessä, mikropalveluarkkitehtuureissa ja integraatioissa, joissa järjestelmät vaihtavat tietoa keskenään.

JSON-skeemojen käyttö on todennäköisesti kasvussa tulevaisuudessa, sillä järjestelmien monimutkaistuessa ja datan määrän kasvaessa tarve tehokkaille validointi- ja dokumentointiratkaisuille korostuu. Tämä luo uusia mahdollisuuksia, mutta myös haasteita ohjelmistokehitykselle. Opinnäytetyön tuloksia voivat hyödyntää ohjelmistokehittäjät, järjestelmäarkkitehdit ja muut IT-alan asiantuntijat, jotka haluavat varmistaa ohjelmistojensa luotettavuuden, joustavuuden ja yhteensopivuuden JSON-skeemojen avulla.

Lähteet

APItoolkit. (2024). How to Ensure Data Integrity and Consistency in APIs. https://apitoolkit.io/blog/how-to-ensure-data-integrity-and-consistency-in-apis/

Berman, J. (9.9.2023). How JSON Schema Was an Obvious Choice at GitHub. *JSON Schema.* https://json-schema.org/blog/posts/github-case-study

Eriksson, P & Koistinen, K. (2014). Monenlainen tapaustutkimus. Kuluttajatutkimuskeskus.

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures* [väitöskirja, Kalifornian yliopisto], Irvine. https://ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

Gbadebo, B. (23.10.2023). What is JSON Schema? *Postman.* https://blog.postman.com/what-is-json-schema/

GitHub. (12.7.2024). *Ajv JSON schema validator.* Github. https://github.com/ajv-validator/ajv

Gutermuth, M. & Hutton, B. (8.5.2023). How 6 River Systems saves time and boosts collaboration with JSON Schema. *JSON Schema.* https://json-schema.org/blog/posts/6-river-systems-case-study

Harley, M. (2024). Pentesting APIs: A Practical Guide to Discovering, Fingerprinting, and Exploiting APIs. Birmingham Packt Publishing.

Harward Business School Online. A Beginner's Guide to Data & Analytics.

Heroku. (n.d.-a) *What is Heroku?* Heroku. Salesforce. Haettu 25.2.2025 osoitteesta https://www.heroku.com/about

Heroku. (n.d.-b). *Heroku Dynos.* Heroku. Salesforce. Haettu 25.2.2025 osoitteesta https://www.heroku.com/dynos

Hutton, B. (6.5.2023). Using JSON at Remote to scale forms and data validations. *JSON Schema.* https://json-schema.org/blog/posts/remote-case-study

Hutton, B. (9.12.2021). JSON Schema deduplicated complex logic and validation at Cookpad. *JSON Schema.* https://json-schema.org/blog/posts/cookpad-case-study-en

Innocent, A. (12.11.2024). API Validation Best Practices: Ensuring Smooth Operations with Apidog. *Apidog.* https://apidog.com/blog/api-validation-best-practices/

Itelman, R. & Cruz Viotti, J. (2024). Unifying Business, Data and Code. O'Reilly Media, Inc.

JSON Schema. (n.d.-a). *Overview. What is JSON Schema?* Haettu 25.1.2025 osoitteesta https://json-schema.org/overview/what-is-jsonschema

JSON Schema. (n.d.-b). *Getting Started. Creating your first schema.* Haettu 25.1.2025 osoitteesta https://json-schema.org/learn/getting-started-step-by-step

JSON Schema. (n.d.-c). *Reference. Type-specific keywords.* Haettu 20.2.2025 osoitteesta https://json-schema.org/understanding-json-schema/reference/type

JSON Schema. (n.d.-d). *Reference. Schema Composition.* Haettu 20.2.2025 osoitteesta https://json-schema.org/understanding-json-schema/reference/combining

JSON Schema. (n.d.-e). *Reference. Applying Subschemas Conditionally.* Haettu 15.2.2025 osoitteesta https://json-schema.org/understanding-json-schema/reference/conditionals

JSON Schema. (n.d.-f). *Reference. Structuring a complex schema.* Haettu 20.2.2025 osoitteesta https://json-schema.org/understanding-json-schema/structuring

JSON Schema. (n.d.-g). *JSON Schema Tooling.* Haettu 15.2.2025 osoitteesta https://json-schema.org/tools

KrakenD. (26.11.2018). *Community Documentation. Introduction.* KrakenD. https://www.krakend.io/docs/overview/

Lauret, A. (2019). The Design of Web APIs. Manning Publications Co.

Lombarte, A. (20.11.2023). How KrakenD API Gateway uses JSON Schema. Technical Insights & Best Practices. *KrakenD.* https://www.krakend.io/blog/json-schema-use-case/

Marrs, T. (2017). *JSON at Work.* O'Reilly Media, Inc.

McIntyre, J. (2014). *Using JSON Schema. Learn and Apply JSON Schema by Example, with Javascript (Node.js) and Python Programs.* Joe McIntyre.

Medjaoui, M., Amundsen, M., Ronnie, M & Wilde E. (2018). Continuous API Management. O'Reilly Media, Inc.

Moilanen, J., Niinioja, M., Seppänen, M. & Honkanen, M. (2018). *API-talous 101.* Alma.

Npm. (2.7.2024). *Ajv JSON schema validator.* Npm. https://www.npmjs.com/package/ajv

Npm. (2.2.2022). *json-schema.* Npm. https://www.npmjs.com/package/json-schema

Pedro, B. (2024). *Building an API Product: Design, Implement, Release, and Maintain API Products That Meet User Needs.* O'Reilly Media, Inc.

Postman. (n.d.-a). *What is an API?* Postman. Haettu 1.2.2025 osoitteesta https://www.postman.com/what-is-an-api/

Postman. (n.d.-b). *API Design.* Postman. Haettu 18.2.2025 osoitteesta https://www.postman.com/api-platform/api-design/

Postman. (n.d.-c). *API Versioning.* Postman. Haettu 20.2.2025 osoitteesta https://www.postman.com/api-platform/api-versioning/

Reselman, B. (2.10.2020). An architect's guide to APIs: SOAP, REST, GraphQL, and gRPC. *Red Hat.* https://www.redhat.com/en/blog/apis-soap-rest-graphql-grpc

Sandoval, K. (9.1.2024). 4 Examples of JSON Schema In Production. *Nordic APIS.* https://nordicapis.com/examples-of-json-schema-in-production/

Shiode, K. (6.4.2021). Introducing JSON Schema to Cookpad Mart's product registration screen. Developers Blog. *Cookpad.* https://techlife.cookpad.com/entry/mart-json-schema

Srivastava, S. (17.11.2023). Understanding JSON Schema: A Guide for Qodex.ai Users. *Qodex.ai.* https://qodex.ai/blog/understanding-json-schema-a-guide-for-qodex-ai-users

Vasudevan, K. (25.9.2023). Why Does API Documentation Matter? *Swagger.* https://swagger.io/blog/api-documentation/what-is-api-documentation-and-why-it-matters/

Young, J. (3.6.2024). Saved by the Schema: Using JSON Schema to Document, Test, and Debug APIs. *Heroku.* https://blog.heroku.com/json-schema-document-debug-apis

Liite 1. Aineistonhallintasuunnitelma

Opinnäytetyön aineiston kuvaus

Opinnäytetyössä käytetään vain julkisissa lähteissä olevia tietoja

Aineistonhankinnan menetelmä on tapaustutkimukset.

Tekijät ja lähteet on merkitty HAMK:in lähdeviittausohjeen mukaisesti.

Analysoitava aineisto on pääosin tekstimuodossa ja kerätty kirjallisuudesta, verkkosivuilta ja blogikirjoituksista.

Aineiston tallennus ja säilytys

Aineisto tallennetaan ja sitä käsitellään opinnäytetyön tekijän omalla salasanalla suojatulla tietokoneella. Aineistosta tallennetaan erilliseen kansioon varmuuskopiot, joita säilytetään erillään analysoitavista tiedostoista. Opinnäytetyön tekijän lisäksi aineistoa käsittelee myös opinnäytetyön ohjaaja.

Henkilötietojen ja arkaluonteisten tietojen käsittely

Opinnäytetyössä ei käsitellä henkilötietoja tai arkaluonteista tietoa.

Opinnäytetyössä ei ollut toimeksiantajaa.

Aineiston jatkokäyttö työn valmistumisen jälkeen

Tutkimusaineistoa ei jatkokäytetä.

Liite 2. Cookpad Mart JSON-skeeman määrittely tuoterekisteröinnille

Cookpad Mart tuoterekisteröinnin JSON-skeemaa määriteltäessä oli tärkeää luoda rakenne, joka mahdollistaa lomakkeiden mukautumisen tuotetyypin mukaan. Alla on esimerkkejä yleisimmistä tuotetyypeistä ja niiden vaatimista kentistä (Shiode, 2021):

- Juurikasvit (sipulit, porkkanat, perunat)
    - Laadunvarmistustyyppi: Huomautus → tekstikenttä (oletuksena ”Kuluta mahdollisimman pian”)
- Broilerinliha
    - Laadunvarmistustyyppi: Viimeinen käyttöpäivä → Päivämääräkenttä
    - Sulatusmerkintä → erillinen kenttä
- Prosessoitu merenelävä
    - Raakaruoan merkintä (syötävä raakana/kypsennettävä)
    - Viljelytiedot (viljely/villikala)
    - Sulatusmerkintä
- Sulatetut tuotteet
    - Laadunvarmistustyyppi: Viimeinen käyttöpäivä → Päivämääräkenttä
- Ei-sulatetut tuotteet
    - Laadunvarmistustyyppi: Takuuajan viimeinen käyttöpäivä (vähimmäisaika toimituspäivästä) → Päivämääräkenttä

Erityisesti prosessoitujen merenelävien kohdalla lomakerakenteesta tuli monimutkainen. Tällaisissa tilanteissa JSON-skeeman avulla voidaan hyödyntää esimerkiksi *dependencies* ja *oneOf* määrittelyjä ehtoperusteiseen validointiin. Skeema määrittelee, että jos tuotteen kenttä thawed on tosi (eli tuote on sulatettu), vaaditaan expiration-tyyppinen quality_guarantee. Jos kenttä thawed on epätosi, käytetään sen sijaan toista guarantee_expiration-määritelmää. Tämän avulla lomake voi muuttua dynaamisesti tuotteen ominaisuuksien mukaan ja estää virheellisen tiedon tallentumisen tuotantojärjestelmään. Vaikka lopullinen JSON-rakenne oli monimutkainen, sen kirjoittaminen oli huomattavasti helpompaa kuin rakentaa lomakelogiikka ja validoida alusta asti manuaalisesti. (Shiode, 2021).

Esimerkki JSON-skeemasta prosessoitujen merenelävien tietojen määrittelyyn. (Shiode, 2021)

```
{
  "required": [
    "raw",
    "thawed",
    "farmed"
  ],
  "properties": {
    "category_id": {
      "const": "Seafood product category ID"
    },
    "thawed": {
      "$ref": "#/definitions/thawed"
    },
    "farmed": {
      "$ref": "#/definitions/farmed"
    },
    "raw": {
      "$ref": "#/definitions/raw"
    }
  },
  "dependencies": {
    "thawed": {
      "oneOf": [
        {
          "properties": {
            "thawed": {
              "const": true
            },
            "quality_guarantee": {
              "$ref": "#/definitions/quality_guarantee/definitions/expiration"
            }
          }
        },
        {
          "properties": {
            "thawed": {
              "const": false
            },
            "quality_guarantee": {
              "$ref": "#/definitions/quality_guarantee/definitions/guarantee_expiration"
            }
          }
        }
      ]
    }
  }
}
```