Olli Varila

Images-mikropalvelun toteutus ja optimointi

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

24.4.2025

Tiivistelmä

Tekijä:	Olli Varila
Otsikko:	Images-mikropalvelun toteutus ja optimointi
Sivumäärä:	40 sivua
Aika:	24.4.2025
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Lehtori Jorma Räty
	Senior Cloud Developer Kimmo Ahokas

Insinöörityössä toteutettiin Sanoma Media Finland Oy:lle Images-mikropalvelu. Työn tavoitteena oli tuottaa moderneilla teknologioilla uusi kuvapalvelu joukolle Suomen suurimpia uutispalveluita. Kuvapalvelun tavoitteena oli noudattaa uutisalustan parhaita käytäntöjä ja vastata nykypäiväisiin kuvatarpeisiin. Sovelluskehitys toteutettiin iteratiivisesti ketterää kehitystä ja DevOps-käytäntöjä soveltaen.

Palvelun toteutuksen lisäksi työssä keskityttiin palvelun ja kuvien optimointiin sekä suorituskykytestaukseen. Uusi palvelu tukee AVIF-kuvaformaattia, jonka käyttöönoton jälkeen toteutusta optimoitiin suorituskyvyn parantamiseksi. Suorituskyvyn mittaamiseksi toteutettiin suorituskykytesti, jonka avulla vertailtiin skaalaussuodattimia.

Lopputuloksena syntyi uusi mikropalvelu, jonka ylläpitäminen ja kehittäminen on helppoa ja luotettavaa. Käytetyt teknologiat tekevät ratkaisusta skaalautuvan, joka mahdollistaa yli 300 miljoonan kuvan lähettämisen käyttäjille päivittäin. Palvelu täyttää annetut vaatimukset ja tarjoaa hyvät puitteet kuvajournalismin kehittämiselle tulevaisuudessa. AVIF-kuvaformaatin käyttöönotto nosti käytettyjen kuvien laatua, ja optimoinnin avulla saavutettiin hyväksyttävä tasapaino kuvien laadun, koon ja suorituskyvyn välillä. Skaalaussuodattimien vertailussa ei huomattu merkittäviä eroja suorituskyvyssä.

Avainsanat: mikropalvelu, kuvankäsittely, TypeScript, AWS, AVIF

Abstract

Author:	Olli Varila
Title:	Implementation and Optimization of ‘Images’ Microservice
Number of Pages:	40 pages
Date:	24 April 2025
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication technology
Professional Major:	Software Engineering
Supervisors:	Jorma Räty, Senior Lecturer
	Kimmo Ahokas, Senior Cloud Developer

This thesis covers the implementation of the Images microservice for Sanoma Media Finland Oy. The goal was to implement a new image service built with modern technologies for one of the biggest news platforms in Finland. The objective for this service was to follow the best practices of the platform and to fulfill the current requirements for images. The development was done by applying agile development and DevOps practices.

The study also focused on optimization of the service and the images used with the help of performance tests. Pictures were optimized by taking AVIF format into use, followed by optimization of the service to improve its performance. A performance test was implemented to provide a framework for measuring performance. Finally, resize kernels were evaluated using this framework.

The outcome is a working service that is maintainable, robust and easy to develop. The technologies used make the service highly scalable, allowing it to serve over 300 million pictures to users every day. The implementation fulfills the requirements given and provides excellent capabilities for the development of picture journalism in the future. New AVIF image format was used to improve the quality of the images used. Optimization resulted in finding an acceptable balance between quality, size and performance. The resize kernel evaluation showed that there are no significant performance differences in this case.

Keywords: microservice, image processing, TypeScript, AWS, AVIF

Sisällys

Lyhenteet

AWS: *Amazon Web Services*. Suosittu pilvipalvelujen tarjoaja, jonka alustan ja palvelujen avulla voidaan rakentaa pilvilaskentaan perustuvia sovelluksia, jotka ovat muun muassa turvallisia ja luotettavia.

SNDP: *Sanoma News Delivery Platform.* Sanoma Media Finland Oy:n uutisalusta, jonka avulla hallinnoidaan uutissivustojen sisältöä ja rakennetta sekä tarjoillaan sisältöä loppukäyttäjille.

CDN: *Content Delivery Network.* Globaalisti hajautettu ryhmä verkkopalvelimia, joiden tehtävä on lähettää verkkosivujen sisältöä päätelaitteille esimerkiksi selaimille. Palvelu nopeuttaa verkkosivujen lataamista, koska data haetaan geograafisesti lähimmästä palvelimesta.

JSON: *JavaScript Object Notation.* Kevyt dataformaatti, jota ihmiset ja koneet pystyvät lukemaan ja kirjoittamaan helposti. Merkittävä osa tietojärjestelmien välisestä kommunikaatiosta käydään tämän formaatin avulla.

YAML: *Yet Another Markup Language.* Helposti luettava dataformaatti, jota käytetään usein konfiguraatioiden määrittämiseen.

HTML: *Hypertext Markup Language.* Kieli, jota käytetään verkkosivujen sisällön näyttämiseen selaimissa.

I/O: *Input/Output.* Tarkoittaa datan kirjoittamista tai lukemista. Yleensä dataa luetaan jostakin I/O-laitteesta, kuten näppäimistöstä tai verkkokortista. Näin ollen I/O mahdollistaa ohjelman kommunikaation ulkomaailman kanssa.

1 Johdanto

Insinöörityössä kehitettiin yritystoimeksiantona uusi kuvapalvelu nimeltä Images. Toimeksiantaja on Sanoma Media Finland Oy, joka on Sanoma-konserniin kuuluva media-alan yhtiö. Yrityksen portfolioon kuuluvat esimerkiksi Helsingin Sanomat, Ilta-Sanomat ja aluemediat, joiden verkkosivuja ja mobiilisovelluksia pyöritetään SNDP-alustan avulla.

Tavoitteena oli luoda uusi kuvapalvelu, joka tukee vanhojen ominaisuuksien lisäksi uutta AVIF-kuvaformaattia ja tarjoaa kyvykkyydet kuvajournalismin kehittämiselle tulevaisuudessa. Projekti on osa SNDP-alustan arkkitehtuuriuudistusta, jossa siirrytään kohti modernia mikropalveluarkkitehtuuria. Palvelu toteutettiin moderneilla teknologioilla, ja se noudattaa alustan nykypäiväisiä parhaita käytäntöjä.

Palvelun toteutuksen lisäksi suoritettiin AVIF-kuvaformaatin käyttöönotto ja optimoitiin käytettäviä kuvia sekä kuvapalvelun suorituskykyä. Optimoinnilla pyrittiin löytämään sopiva tasapaino suorituskyvyn, laadun, tiedostokoon ja kustannusten välillä. Optimoinnin avuksi toteutettiin suorituskykytesti ja sen automaatio. Suorituskykytestauksella mitataan muutosten vaikutusta suorituskykyyn ja pyritään havaitsemaan mahdolliset regressiot mahdollisimman aikaisessa vaiheessa.

2 Kuvapalvelun lähtökohdat

Verkkosivuilla on käytetty kuvia jo vuodesta 1993 alkaen, kun ensimmäinen selain alkoi tukemaan HTML-kuvaelementtejä [1]. Kuvilla on merkittävä vaikutus sivun ilmeeseen, ja ne ovat usein olennainen osa sisältöä. Kuvapalvelulle on siis aina ollut tarvetta, mutta teknologia kehittyy, ja joskus uusien teknologioiden käyttöönotto vaatii järjestelmän uusimista.

Kuvia usein optimoidaan käyttämällä responsiivisia kuvia (engl. responsive images). Responsiivisen kuvan leveys ja formaatti valitaan dynaamisesti kontekstin mukaan. Jos esimerkiksi selain tukee AVIF-formaattia, ja kuvaelementin leveys on 400 pikseliä, selain pyytää palvelimelta 400 pikseliä leveän kuvan AVIF-formaatissa. Responsiivisten kuvien avulla voidaan priorisoida paremmin pakkautuvia kuvaformaatteja ja ladata pienin sopiva kuva, jolloin keskimäärin ladatut tavut ja kuvien latausaika laskee. Responsiiviset kuvat implementoidaan HTML-tasolla, mutta niitä varten tarvitaan kuvapalvelu, joka kykenee tuottamaan useita versioita samasta kuvasta.

2.1 SIDP

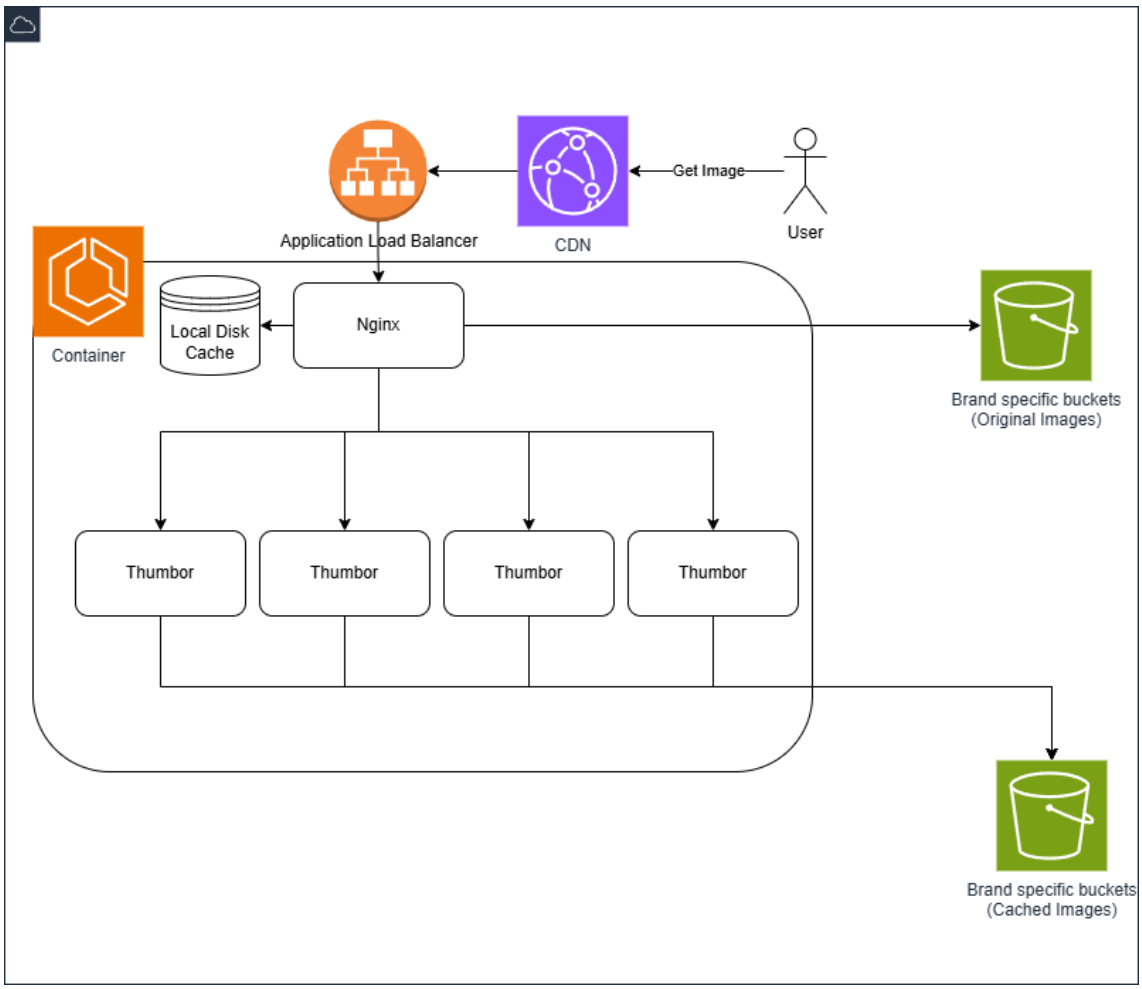
SIDP eli Sanoma Image Delivery Platform on SNDP-alustan korvattava kuvapalvelu. Palvelu koostuu kahdesta komponentista: web-palvelimesta, joka lähettää skaalattuja versioita alkuperäisistä kuvista päätelaitteille, sekä rajapinnasta, jonka kautta lisätään uusia kuvia. Kuvien generointiin käytetään kontissa pyörivää Nginx-web-palvelinta, joka generoi kuvia thumbor-kirjaston avulla. Kuvia haetaan brändikohtaisista tietovarastoista, joihin rajapinta tallentaa niitä. Rajapinta on tehty Java-ohjelmointikielellä ja Spring-ohjelmistokehyksellä.

2.2 Miksi tarvitaan uusi palvelu?

SIDP on tehty responsiivisia kuvia varten noin 10 vuotta sitten. Toteutuksen jälkeen palvelua on pääosin vain ylläpidetty. SNDP-alusta ja teknologiat ovat kehittyneet, jonka seurauksena SIDP ei enää noudata parhaita käytäntöjä eikä täytä alustan nykypäiväisiä tarpeita. Tarpeisiin kuuluu esimerkiksi mahdollisuus

kuvajournalismin kehitykseen ja tuki AVIF-kuvaformaatille. Uusien ominaisuuksien kehittäminen vanhaan palveluun on hankalaa, koska kuvien generointiin käytetään Thumbor-palvelua kustomoidun Thumbor AWS -lisäosan kanssa. Palvelun päivittäminen uusimpaan versioon vaatisi lisäosan ylläpitoa, ja ratkaisu perustuisi edelleen monimutkaisiin konfiguraatiotiedostoihin, mikä ei ole yhtä joustavaa kuin kuvien ohjelmallinen generointi. Lisäksi AWS on lopettanut kontin pohjakuvan tukemisen, joten sovelluksesta ei pysty julkaisemaan uutta versiota sen nykyisessä tilassa.

Myös palvelun arkkitehtuuri on uudistuksen tarpeessa. 10 vuoden aikana AWS on julkaissut runsaasti uusia hallinnoituja palveluita, joiden käyttäminen kuuluu SNDP-alustan parhaisiin käytäntöihin, koska ne helpottavat ylläpitoa. SIDP-toteutuksessa on käytetty Nginx web-palvelinta, koska thumbor on vain omassa prosessissaan pyörivä rajapinta, jolle pyyntöjä ohjataan osoitteen perusteella. Sovellus hakee ja kirjoittaa kuvia Nginx-palvelimen läpi, minkä takia pyyntöjä kierrätetään palvelun sisällä useaan kertaan yhden pyynnön aikana (kuva 1). Lisäksi nykyisessä ratkaisussa jokaiselle brändille on oma verkko-osoite ja tietovarasto, jolle ei ole enää tarvetta.



Kuva 1. SIDP delivery -arkkitehtuurikuva.

Kaikkien mainittujen muutoksien tekeminen vaatisi fundamentaalisia muutoksia järjestelmän arkkitehtuuriin ja toteutukseen. Lyhyellä aikavälillä muutosten tekeminen nykyiseen palveluun saattaa olla nopeampaa, mutta pitkällä aikavälillä uutta palvelua pitäisi olla helpompaa kehittää, koska teknistä velkaa on huomattavasti vähemmän.

2.3 Vaatimukset

Tässä luvussa määritellään uuden palvelun vaatimukset. Määritykseen käytetään arkkitehtonisia piirteitä ja toiminnallisia vaatimuksia [2, s. 92-102]. Tunnistettujen vaatimusten lisäksi järjestelmän tulee noudattaa SNDP-alustan arkkitehtuurivisiota. Visioon kuuluu joustavuus, skaalautuvuus, ylläpidettävyys, kyvykkyys kehittää palvelua tehokkaasti ja hyvä kehittäjäkokemus.

2.3.1 Arkkitehtoniset piirteet

Täydellistä arkkitehtuuria ei ole olemassa. Sen sijaan on tähdättävä vähiten huonoon arkkitehtuuriin. Mahdollisia arkkitehtonisia piirteitä on paljon, ja liian monen priorisoiminen kasvattaa kompleksisuutta, mikä johtaa hyvin geneeriseen arkkitehtuuriin. Kun piirteitä aluksi valitaan mahdollisimman vähän, on arkkitehtuuria helpompi muuttaa toteutusta iteroidessa. [2, s. 102-103.] Tämä huomioiden kuvapalvelun tärkeimmät arkkitehtoniset piirteet ovat saatavuus (engl. availability), suorituskyky, skaalautuvuus, konfiguroitavuus, laajennettavuus ja ylläpidettävyys.

Saatavuus, suorituskyky ja skaalautuvuus kulkevat käsi kädessä. Palvelun kuuluisi olla korkeasti saatavilla eli kuvien pitää näkyä sivustoilla lähes aina. Palvelun suorituskyky on tärkeä prioriteetti, koska liian hidas kuvien lataus heikentää käyttäjäkokemusta. Media-alalla suuret käyttöpiikit ovat yleisiä, kun esimerkiksi merkittäviä uutisia julkaistaan. Kuvapalvelun pitää reagoida äkillisiin muutoksiin käyttäjämäärissä ja pystyä toimimaan oikein näissä tilanteissa.

Konfiguroitavuudella tarkoitetaan käyttäjien kykyä muuttaa sovelluksen toimintaa asetuksia säätämällä. Laajennettavuus taas kuvaa, miten helppoa uusien toiminnallisuuksien lisääminen on järjestelmään. Nämä ovat tärkeitä prioriteetteja, koska konfigurointi ja mahdollisuus jatkokehitykseen ovat vaatimuksia palvelulle.

Ylläpidettävyys kertoo, miten helppoa järjestelmän päivittäminen on. Ylläpitotyötä tulisi olla vähän ja sen pitäisi tapahtua tehokkaasti. Arkkitehtuuria suunnitellessa voidaan vaikuttaa järjestelmän ylläpidettävyyteen valitsemalla helposti ylläpidettäviä teknologioita, joita ovat esimerkiksi AWS-hallinnoidut palvelut.

2.3.2 Toiminnalliset vaatimukset

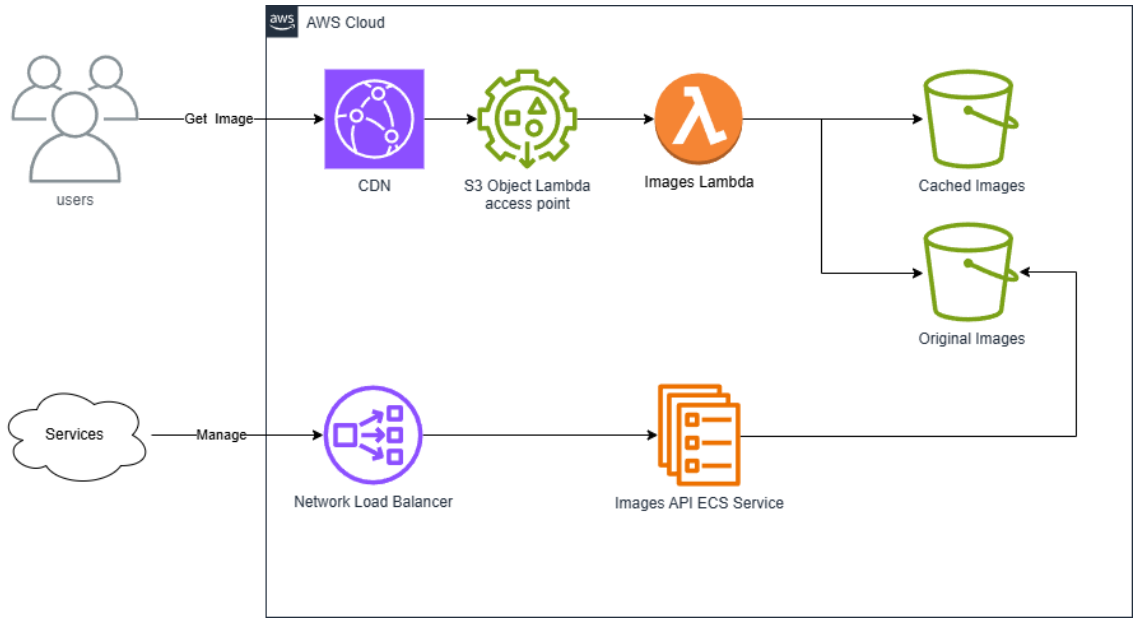
Palvelun pitää tukea samoja kuvaformaatteja, leveyksiä ja tyyppejä kuin nykyinen palvelu. Korvattavan palvelun kuvaformaatteihin kuuluu esimerkiksi WebP, SVG

ja GIF, JPG ja PNG. Kuvatyyppejä on neliö, normaali ja somekuva, jossa on brändin vesileima. Lisäksi halutaan ottaa käyttöön uusi AVIF-kuvaformaatti.

Hallinnointirajapinnan tulisi olla lähes identtinen versio korvattavasta rajapinnasta. Kuvien lähetykseen käytetään multipart/form-data-formaattia ja BASIC-autentikaatiota autentikaatioon. Ainoa uusi ominaisuus rajapinnassa on kuvien poistaminen.

2.4 Arkkitehtuuri

Kuvassa 2 on Images-palvelun arkkitehtuurikuva. Arkkitehtuurissa käytetään mahdollisimman paljon hallinnoituja ja palvelimettomia (engl. serverless) palveluja pelkän laskentatehon vuokraamisen sijaan. Hallinnoidut ratkaisut joustavat vähemmän, mutta helpottavat ylläpitoa, koska ajonaikainen infrastruktuuri on palveluntarjoajan vastuulla. Kehittäjän tehtäväksi jää hallinnoidun palvelun konfiguraatio ja suoritettavan koodin julkaisu. Lisäksi joustamattomuus tarkoittaa yleensä sitä, että palvelu on suunniteltu tiettyjä yleisiä käyttötapauksia varten, joissa se on erittäin hyvä.



Kuva 2. Images-palvelun arkkitehtuurikuva.

Arkkitehtuurikuvasta nähdään, että käyttäjien pyynnöt menevät edelleen CloudFront CDN:n kautta, mutta pyynnön ottaa vastaan web-palvelimen sijasta Lambda-funktio, joka generoi uusia kuvia, mikäli kuvaa ei löydy välimuistista. Kuvien tietovarastona käytetään edelleen S3-palvelua, mutta brändikohtaisista tietovarastoista on luovuttu. Kuvien hallinnointirajapintaa kutsutaan Network Load Balancerin läpi, ja se tallentaa uusia kuvia ulkuperäisten kuvien tietovarastoon. Rajapinnan pitää myös pystyä tyhjentämään välimuisti kuvien poistamista varten. Tämä on kuitenkin todella harvinainen käyttötapaus, jonka takia se on jätetty pois kuvasta.

3 Teknologiat

Tässä luvussa esitellään kuvapalvelun toteutuksen tärkeimmät teknologiat. Jokaisesta teknologiasta annetaan korkean tason kuvaus ja perustellaan sen valinta. Toteutusvaiheessa perehdytään teknologioiden käyttöön enemmän.

3.1 Pilvipalvelut

Suuri osa SNDP-alustan palveluista pohjautuu AWS-pilvipalveluihin. Saman tarjoajan käyttäminen kuvapalvelussa helpottaa järjestelmän integraatiota alustan kanssa. Lisäksi toteutus on nopeampaa, kun aikaisempaa kokemusta voidaan hyödyntää.

3.1.1 Amazon S3

Amazon Simple Storage Service on oliotietovarasto, joka on suosittu tapa säilyttää staattista dataa, kuten esimerkiksi kuvia tai dokumentteja pilvessä. Palvelu skaalautuu lähes äärettömästi eli tietovarastosta ei käytännössä lopu tila koskaan.

Tallennettua dataa on mahdollista salata monella eri tapaa ja se on tallennettu luotettavasti. Datan säilyvyys (engl. durability) S3-palvelussa on 99,999999999 % eli käytännössä 10 000 000 tiedostoa kohden voi olettaa, että kymmenessä tuhannessa vuodessa yksi tiedosto keskimäärin korruptoituu tai katoaa.

Palvelu tarjoaa monia eri tallennusluokkia, joilla on omat käyttötarkoituksensa ja hintansa. Esimerkiksi usein käytetylle datalle suositellaan S3 Standard -luokkaa, jonka hinta on 0.021-0.023 gigatavua kohti kuukausittain, kun taas harvemmin käytetylle datalle suositellaan joko S3 Standard - Infrequent Access tai jotakin S3 Glacier -luokista, jotka on tarkoitettu pitkäaikaiseen datan säilytykseen. Arkistoluokissa datan säilyttäminen on huomattavasti edullisempaa, mutta datan hakeminen on hitaampaa ja kalliimpaa. Palvelussa on myös S3 Intelligent – Tiering, jonka avulla tiedostot siirretään niiden käytön perusteella sopivaan luokkaan. [3.]

Images käyttää S3-palvelua tallennusratkaisuna, koska data on binääriformaatissa eikä sitä vasten tarvitse suorittaa kyselyitä. Muita vaihtoehtoja olisi Amazon Elastic Block Storage tai tietokanta. EBS ei toimi, koska se ei tue AWS Lambdaa, jota halutaan käyttää palvelussa. Tietokantaan datan tallentaminen ei ole hyvä idea, koska dataan ei kohdisteta kyselyjä ja tietokannan hintaan kuuluu laskentateho, joka menisi tässä tapauksessa hukkaan. Lisäksi tietokantoihin yleensä liittyy ylläpitoa ja hallinnointia, josta kertyy lisäkustannuksia.

3.1.2 AWS Lambda

AWS Lambda on palvelu, jonka avulla voidaan suorittaa koodia pilviympäristössä ilman palvelinten vuokraamista ja hallinnointia. Yhtä instanssia kutsutaan Lambda-funktioksi ja se on kirjaimellisesti koodifunktio, joka suoritetaan jokaisella pyynnöllä. Esimerkkikoodissa 1 on yksinkertaisen Lambda-funktion toteutus.

```
export const handler = async (event, context) => {
  const response = {
    statusCode: 200,
    body: JSON.stringify(“Hello world!”)
  };
  return response;
}
```

Esimerkkikoodi 1. Yksinkertainen Lambda-funktio JavaScript-kielellä.

AWS Lambda hoitaa instanssien skaalaamisen, hallinnoinnin ja infrastruktuurin ylläpidon. Funktiot skaalautuvat siten, että vain tarpeellinen määrä instansseja on päällä. Koska funktiot ovat eristettyjä prosesseja, tämä tekee palvelusta hyvin kestävän, kun virhe yhdessä ohjelmassa ei vaikuta muiden instanssien suoritukseen. Palvelussa on myös joustava hinnoittelu, jolloin maksetaan vain suoritetuista pyynnöistä. Tämä tekee Lambdasta ideaalin myös sovelluksille, jotka saavat vähän liikennettä. [4.]

Lambdaa ei käytetä hallinnointirajapinnan toteutuksessa, koska sen kutsumiseen siinä tapauksessa tarvitaan joko API Gateway tai Application Load Balancer ja nämä palvelut rajoittavat pyynnön koon 10 megatavuun. Kuvaskaalaimessa

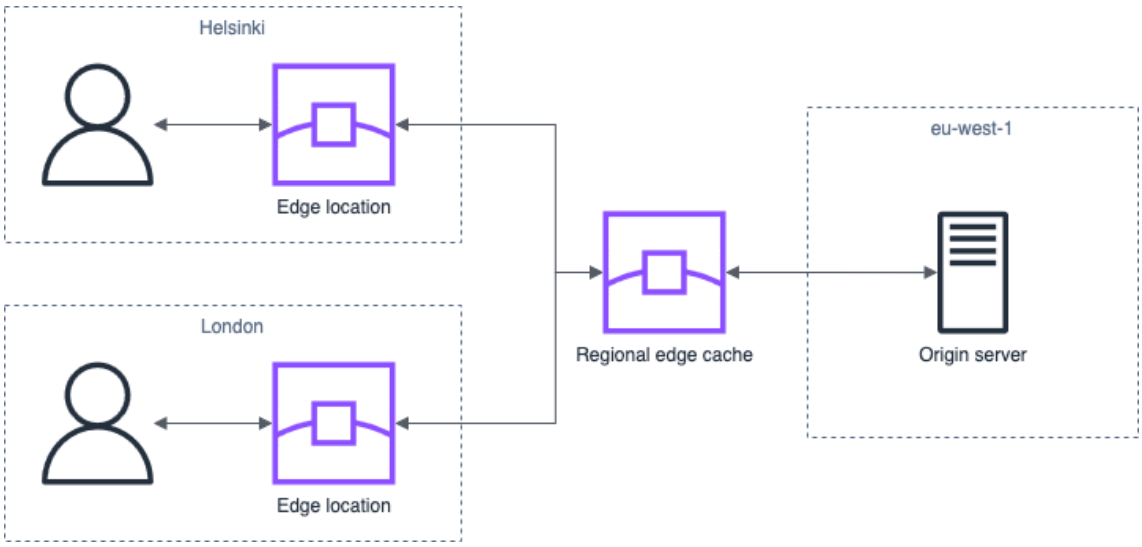
Lambda-funktio on paritettu S3 Object Lambda Access -pisteen kanssa, jonka avulla funktio toimii välittäjänä (engl. proxy) tietovarastolle.

3.1.3 Amazon ECS

Amazon Elastic Container Service on yksi AWS-konttipalveluista. Sen avulla voi helposti julkaista, hallinnoida ja skaalata konttiteknologiaan perustuvia sovelluksia. Kyseessä on hallinnoitu palvelu, jossa AWS on vastuussa infrastruktuurista. ECS-palvelussa on klustereita, jotka sisältävät palvelumääritelmiä (engl. Service definition), joissa on taas tehtävämääritelmiä (engl. Task definition). Määritelmien avulla hallinnoidaan esimerkiksi ajettavia kontteja, oikeuksia sekä varattua laskentatehoa ja muistia. [5.]

3.1.4 Amazon CloudFront

Amazon CloudFront on globaali CDN-palvelu, jolla tehostetaan staattisen ja dynaamisen sisällön lähetystä päätelaitteille. Palvelu on hajautettu verkko palvelimia, joka tallentaa sisältöä välimuistiin. Lisäksi käyttäjän pyyntö ohjataan ensin geograafisesti lähimpään sijaintiin. Kuvassa 3 näkyy tyypillinen pyynnön kulku CloudFrontin läpi. CloudFrontissa alkuperä (engl. origin) on sovellus, josta sisältö haetaan, mikäli välimuistista löydy vastausta. [6.]



Kuva 3. Pyynnön kulku CloudFront-palvelun läpi.

CloudFrontin avulla voidaan vähentää alkuperäiselle palvelulle saapuvien pyyntöjen määrää huomattavasti, koska palveluun tallennetut kuvat eivät käytännössä muutu koskaan. Yli 99 % vastauksista pitäisi löytyä välimuistista, eli vain 1 % pyynnöistä saapuu alkuperäiselle palvelulle. Nopeuden lisäksi tämä on myös edullisempaa, koska CloudFront veloittaa vain datan siirrosta.

3.1.5 AWS Identity and Access Management

AWS IAM on web-palvelu, jonka avulla määritetään ja hallinnoidaan oikeuksia AWS-resursseihin. Oikeuksien jako tapahtuu hienojakoisesti yleensä resurssi- ja operaatiotasolla. Niitä annetaan identiteetille, jota käytetään autentikaatioon ja autorisaatioon. Sovelluksen identiteettinä käytetään yleensä IAM-roolia, kun taas kehittäjä autentikoituu IAM-käyttäjän avulla. [7.]

Palvelun käyttäminen lisää tietoturvaa ja sen käyttäminen on pakollista AWS-kontekstissa. Pakollisuus johtuu siitä, että IAM kieltää kaikki operaatiot oletuksena, ja ilman tarpeellisia oikeuksia palvelu ei voi kommunikoida muiden resurssien kanssa.

3.2 TypeScript

TypeScript on Microsoftin kehittämä JavaScriptin päälle rakennettu ohjelmointikieli. Se on kehitetty ratkaisuksi JavaScriptin skaalautuvuusongelmiin isoissa koodikannoissa. [8, s. 8.] Kielen staattinen tyypitys tekee koodista ymmärrettävämpää, ja merkittävä osa virheistä havaitaan jo koodin käännösvaiheessa, mikä vähentää ajonaikaisia virheitä.

Kielen ominaisuudet määritetään ECMAScript-standardissa, josta ajonaikaiset ympäristöt tukevat tiettyä versiota. Standardin versiot ovat kumulatiivisia, eli uusi versio sisältää kaikki aikaisemmat ominaisuudet. Vuonna 2025 erittäin hyvin tuettu versio on ES6, mutta esimerkiksi Googlen V8-moottori, jota Node.js ja Chromium-pohjaiset selaimet käyttävät, tukee hyvin ES2022-standardia [10; 11]. Yleensä kirjoitetussa koodissa halutaan käyttää viimeisimpiä kieliominaisuuksia, jolloin koodi muunnetaan käännösvaiheessa yleisemmin tuettuun versioon standardista. Vaihtoehtoisesti koodin mukaan voidaan pakata lisäkoodia täydentämään puuttuvat ominaisuudet.

TypeScriptiä käytetään web-kehityksessä verkkosivuilla ja palvelimilla ympäri maailmaa [9]. Niinpä se on myös laajassa käytössä Sanoma Media Finland Oy -yrityksessä. Kun suurin osa kehittäjistä ymmärtää koodia, sen ylläpito ja jatkokehitys on vaivattomampaa. TypeScript on myös hyvä valinta, koska SNDP-alustalla on kehitetty yhteisiä työkaluja TypeScript-kehitystä varten, kuten valmiita konfiguraatioita. Lisäksi AWS Lambda tukee JavaScriptiä hyvin.

3.3 Sharp

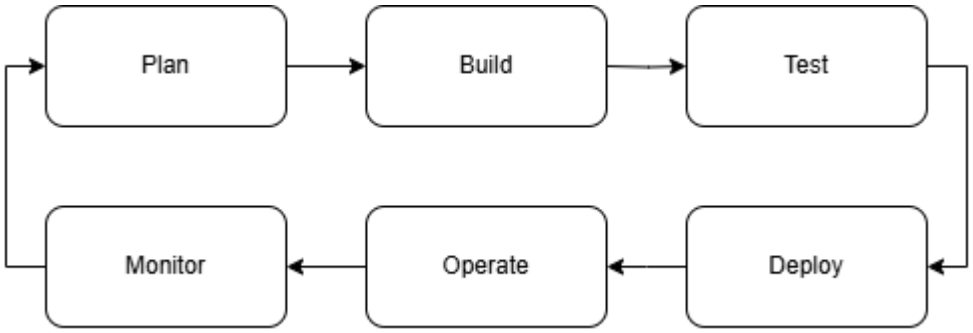
Sharp on JavaScript-ekosysteemin suosituin kuvankäsittelykirjasto, jota tällä hetkellä ladataan noin 37 miljoonaa kertaa viikossa [12]. Kirjasto toimii abstraktiona libvips-kirjastolle, joka on kirjoitettu C++-ohjelmointikielellä. Natiivin koodin suoritus nopeuttaa kuvien käsittelyä, koska käännetty koodi on yleensä optimoidumpaa kuin JavaScript. Lisäksi natiivi koodi rinnakkaistuu paremmin verrattuna JavaScriptiin, jota suoritetaan Node.js-ympäristössä yleensä yhdellä säikeellä.

3.4 Express.js

Express.js on yksi JavaScript-ekosysteemin suosituimmista web-kehyksistä. Se on myös todettu toimivaksi ratkaisuksi muissa SNDP-alustan palveluissa. Kehyksen minimalistisuus tekee siitä sopivan valinnan hallinnointirajapintaan, jossa on vähän reittejä. Lisäksi hallinnointirajapinnan toteutuksessa tärkein arkkitehtoninen piirre on ylläpidettävyys, ja kehyksen suosio yleensä viittaa siihen, että sitä ylläpidetään tulevaisuudessa. [13.]

4 Toteutus

Kuvapalvelu toteutettiin iteratiivisella prosessilla, joka pyrkii soveltamaan ketterän kehityksen ja DevOps-prosessin käytäntöjä, joita on kuvattu kuvassa 4. Kehitysprosessissa suunnittelua ja toteutusta tehdään inkrementaalisesti pienissä vaiheissa ja ominaisuuksia pyritään julkaisemaan mahdollisimman usein, jotta voidaan kerätä palautetta ja aloittaa prosessi uudestaan.



Kuva 4. DevOps-kehitysprosessi

Kehityksessä käytetään myös Trunk Based Development -metodologiaa, joka on lähestymistapa koodin versionhallintaan. Ideaalisesti kehittäjä tekee ”Trunkkiin” eli päähaaraan paljon ja usein pieniä muutoksia, joiden avulla helpotetaan koodin katselmointia ja kasvatetaan tuotosten määrää. Muutoksia ei kuitenkaan tehdä suoraan päähaaraan vaan kehittäjän luomasta ominaisuushaarasta yhdistämispyyntöjen (engl. pull request) avulla. [14.] Projektissa versiohallintaan käytetään git-versiohallintatyökalua ja GitHub-palvelua ja kehitys tehdään Visual Studio Code -koodieditorilla.

Julkaisuprosessin automaatioon käytetään Github Actions- ja AWS Codepipeline -palvelujen yhdistelmää. GitHub Actions suorittaa jokaisen yhdistämispyynnön yhteydessä yksikkötestejä, staattista analyysia ja muita tarkistuksia, joiden läpäiseminen on vaatimus prosessin etenemiselle. Päähaaraan yhdistämisen jälkeen AWS CodePipeline -palvelu tunnistaa muutokset ja käynnistää julkaisuprosessin, jossa käännetään koodi, tehdään tarvittavat muutokset infrastruktuuriin ja otetaan uusi version sovelluksesta käyttöön. Lisäksi julkaisuprosessin aikana suoritetaan integraatio- ja suorituskykytestit.

4.1 Kuvaskaalain

Kuvaskaalain lähettää päätelaitteille kuvia pyynnön reitin perusteella. Reitti kertoo, miltä kuvan pitäisi näyttää, ja koostuu säädettävistä parametreista, joita ovat leveys, tyyppi ja formaatti. Esimerkkejä mahdollisista reiteistä on listattu taulukossa 1.

Taulukko 1. Esimerkkejä reiteistä, joiden perusteella kuvat generoidaan.

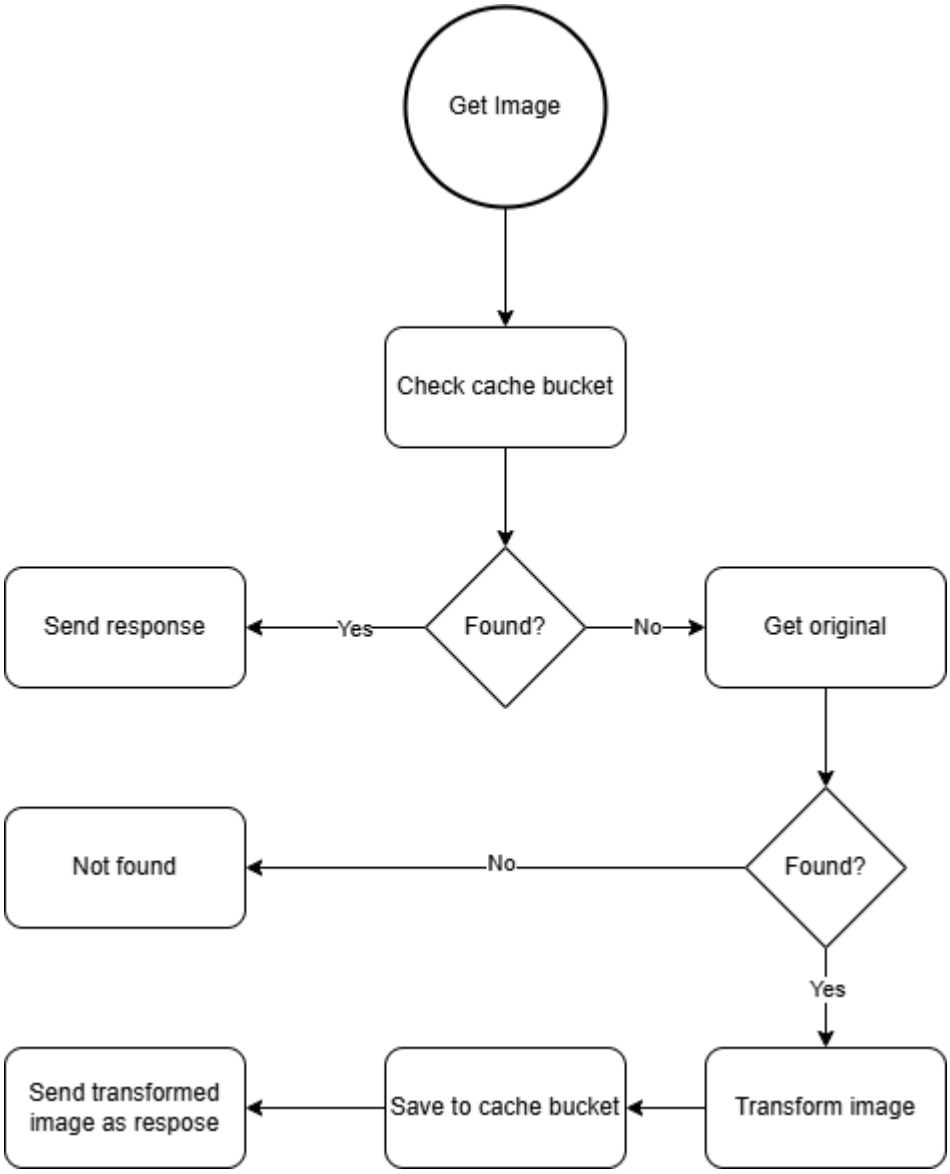
Reitti	**Kuvaus**
/image-id/square/320.webp	image-id tunnuksella tallennetusta kuvasta 320x320 kokoinen kuva Webp-formaatissa
/image-id/normal/978.webp	image-id tunnuksella tallennetusta kuvasta 978 pikseliä leveä kuva alkuperäisessä kuvasuhteessa Webp-formaatissa.
/image-id/some/hs-default.jpg	image-id tunnuksella tallennetusta kuvasta HS vesileimalla varustettu kuva Jpg-formaatissa.
/image-id.svg	image-id tunnuksella tallennettu svg kuva.

Reitin formaatti uudistettiin siirtämällä kuvan tunnus alkuun, mikä helpottaa kuvien poistamista. Reittejä yhtenäistettiin kuvatyyppien avulla ja niiden alusta poistettiin ”/img”-etuliite, koska se oli kaikissa reiteissä. SVG-reitit ovat poikkeus ja sisältävät vain tunnuksen ja tiedostopäätteen. SVG-kuvia ei skaalata, koska ne ovat vektorikuvia ja niiden data on tekstimuodossa. Kuvareittien kielioppi voidaan määrittää formaalisti BNF-muodossa, joka näkyy esimerkkikoodissa 2. Tämä kielioppi on kuitenkin vain kuvaa antava ja pitää ottaa huomioon esimerkiksi, että GIF-formaatissa voi pyytää vain kuvia, joiden alkuperäinen formaatti on GIF.

```
path ::= svg | square | normal | some
svg ::= id '.svg'
square ::= id '/square/' width '.' ext
normal ::= id '/normal/' width '.' ext
some ::= id '/some/' watermark '.' ext
ext ::= 'jpg' | 'png' | 'webp' | 'gif' | 'avif'
watermark ::= string # i.e. 'hs-default'
width ::= number # fixed set
id ::= string
```

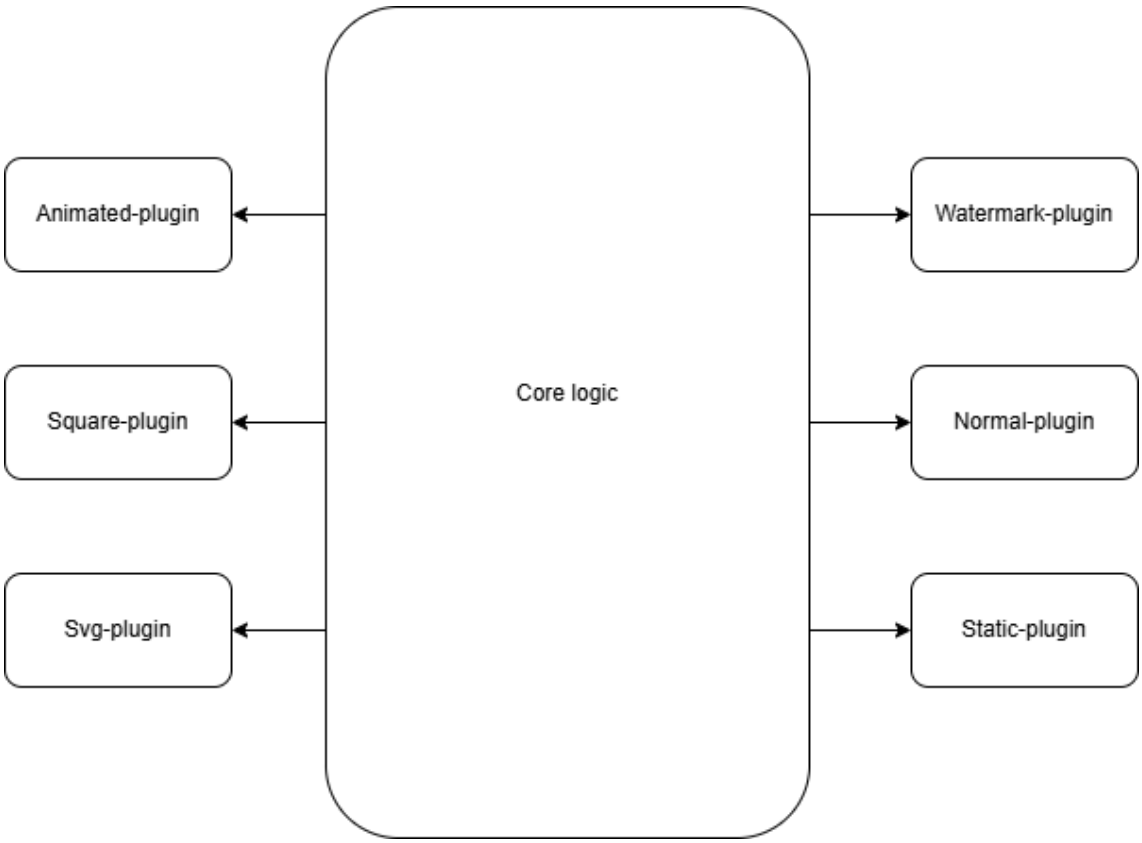
Esimerkkikoodi 2. Kielioppi kuvareitille BNF-muodossa.

Yksittäisen pyynnön korkean tason logiikka näkyy kuvassa 5. Välimuistitietovarastolla varmistetaan, että kuva generoidaan vain kerran siitä huolimatta, että CloudFrontin palvelimet eivät jaa välimuistia (kuva 3). Esimerkiksi kun käyttäjä Helsingistä lähettää pyynnön, kuva generoidaan ja tallennetaan Suomen palvelimen välimuistiin. Sen jälkeen, kun toinen käyttäjä Lontoosta hakee samaa kuvaa, sitä ei löydy alueen välimuistista, jolloin pyyntö menee alkuperäiselle palvelimelle ja generoidaan taas uusi kuva. [6, s. 22-25.]



Kuva 5. Kuvankäsittelypyynnön logiikka vuokaaviossa.

Palvelun toimintalogiikan tunnistamisen jälkeen on helppoa suunnitella sovellusarkkitehtuuri sen ympärille. Kuvasta 5 voidaan päätellä, että pyynnön käsittelyssä muuttuu vain logiikka, jolla kuva generoidaan. Generointilogiikka saattaa myös kehittyä tulevaisuudessa, jos esimerkiksi halutaan luoda uusia kuvatyyppejä. Tämän vuoksi sen abstraktoiminen ja eristäminen muusta logiikasta on hyödyllistä. Sovelluksen lopullinen sisäinen arkkitehtuuri näkyy kuvassa 5, johon päädyttiin sovelluksen toteutusta iteroidessa.



Kuva 6. Kuvaskaalaimen sisäinen arkkitehtuuri.

Sovelluksen arkkitehtuuria kutsutaan Plugin-arkkitehtuuriksi tai Mikrokernel-arkkitehtuuriksi. Tällaisessa sovelluksessa on keskusjärjestelmä, joka hoitaa yleisen prosessoinnin. Toiminnallisuutta voidaan laajentaa pienempien lisäosien avulla. [2, s. 221-238.] Funktio tai moduulitasolla tällaista arkkitehtuuria voidaan rinnastaa strategia-malliin. Kuvaskaalaimessa keskusjärjestelmä hoitaa kuvien hakemisen, kirjoittamisen sekä vastausten lähettämisen. Lisäosien tehtäväksi jää kuvien generointi.

4.1.1 Kuvien haku ja tallennus

Kuvaskaalain hakee kuvia kahdessa vaiheessa pyynnön käsittelyn aikana (kuva 5). Toiminnallisuutta varten implementoidaan TypeScript-moduuli, joka toimii rajapintana haku- ja kirjoitusoperaatioille. Moduulissa käytetään @aws-sdk/client-s3-kirjastoa, jonka avulla kutsutaan S3-rajapintaa komentojen avulla. Tiedoston hakuun käytetään GetObject-komentoa, jolle annetaan parametriksi

tietovaraston nimi ja tiedoston avain. Välimuistitietovarastoon tallennetaan tiedostoja PutObject-komennolla, joka ottaa lisäparametrina tallennettavan datan ja metadataa kuten tiedostotyypin. Esimerkkikoodista 3 nähdään kuvan hakeminen kirjaston avulla.

```
const command = new GetObjectCommand({
  Bucket: 'example-bucket',
  Key: 'image.png',
});

const response = await client.send(command);

// Process response
```

Esimerkkikoodi 3. Tiedoston hakeminen S3-kirjastolla.

S3-tietovaraston avaimen formaatti replikoi usein tiedostojärjestelmän rakennetta, mutta todellisuudessa S3-tietovarastossa ei ole hakemistoja, vaan se on lähempänä avain-arvo-tietokantaa. Välimuistitietovarastoon tallennettavan kuvan avain on pyynnön koko reitti, kuten ”/image-id/square/320.webp”, kun taas alkuperäisten kuvien tietovarastoon tallennetaan kuvia niiden alkuperäisellä tunnuksella, johon on lisätty ”/images/”-etuliite.

Autentikaation ja autorisaatioon käytetään AWS IAM -rooleja ja oikeuksia. Lambda-funktiolla on rooli, joka toimii sen identiteettinä [4, s. 54]. Roolille annetaan kaikki tarvittavat oikeudet, kuten oikeus kutsua S3-rajapintaa. Oikeudet voidaan määrittää hienojakoisesti ämpäri-, operaatio- ja tiedostotasolla. Lambda-funktiolle on annettu oikeus alkuperäisten kuvien tietovarastosta lukemiseen sekä luku- ja kirjoitusoikeus välimuistitietovarastoon. Koodissa autentikaatio tapahtuu automaattisesti, koska S3-kirjasto hoitaa sen lukemalla ympäristömuuttujia, jotka AWS Lambda määrittää automaattisesti.

4.1.2 Kuvien käsittely

Ensimmäisillä iteraatioilla kuvien käsittely tapahtui yhdessä funktiossa, jolle annettiin parsittu reitti ja alkuperäinen kuvadata. Logiikan lisääntyessä tämä funktio muuttui vaikeasti ymmärrettäväksi. Ongelmalle keksittiin ratkaisuksi Plugin-arkkitehtuuri, jonka avulla kaikki yhteen kuvatyyppiin liittyvä logiikka

saadaan koostettua yhteen moduuliin. Yksi lisäosa tunnistaa tiettyjä reittejä ja generoi kuvia tunnistetun reitin perusteella. Lisäosat määritetään omissa moduuleissaan, joista ne koostetaan listaan, jota iteroimalla ja lisäosa-metodeja kutsumalla selvitetään, mitä lisäosaa käytetään.

Kuvan käsittelyä varten parsitaan pyynnön reitistä parametrit, jotka määrittävät generoitavan kuvan. Koska tämä osa palvelusta on julkinen, ja reitissä voi olla mitä vain käyttäjän syötettä, on syötteen validointi erittäin tärkeää. Reitin parsiminen ja validointi tapahtuu kahdessa vaiheessa. Ensin siitä tunnistetaan attribuutit, kuten tunniste, leveys ja formaatti, jonka jälkeen ne validoidaan. Lopputuloksena syntyy JavaScript-olio, joka sisältää kaikki reitistä tunnistetut attribuutit (esimerkkikoodi 4).

```
// 123/normal/978.webp
const parsedKey = {
  id: '123',
  kind: 'normal',
  width: 978,
  ext: 'webp'
};
```

Esimerkkikoodi 4. Parsittu reitti JavaScript-oliona.

Reitin parsimiseen käytetään url-pattern-kirjastoa, jolla määritetään tekstinä, mitä reitistä tunnistetaan. Validointiin käytetään Zod-kirjastoa, jolla määritetään skeema validille JavaScript-oliolle. Kirjastojen valinnassa otettiin huomioon niiden suosio ja helppokäyttöisyys. Tunnistaminen ja validointi yhdistetään funktion avulla, jolle annetaan parametreina tunnistettava reitti ja skeema.

4.1.3 Konfiguroitavuus

Palvelua konfiguroi todennäköisesti tulevaisuudessa muut kehittäjät, jotka ymmärtävät TypeScriptiä. Staattisesti tyypitetyn kielen käyttö konfiguraatiossa mahdollistaa konfiguraation tarkistuksen kääntämisvaiheessa, joka tekee konfiguraatiosta turvallisempaa kuin esimerkiksi JSON-tiedostojen käyttö.

Konfiguraatio toteutetaan moduulina, jossa määritetään jokaiselle ympäristölle kuvien käsittelyyn liittyvät asetukset, tarvittavat ympäristömuuttujat ja mahdolliset

ominaisuuskytkimet (engl. feature flag). Yksinkertainen rajapinta ja tyypitetyn JavaScript-olion käyttäminen tekee konfiguraation käytöstä helppoa. Siksi moduulista jaetaan vain konfiguraatio-tyyppi ja getter-funktio, joka ENVIRONMENT-ympäristömuuttujan perusteella palauttaa ympäristön konfiguraation.

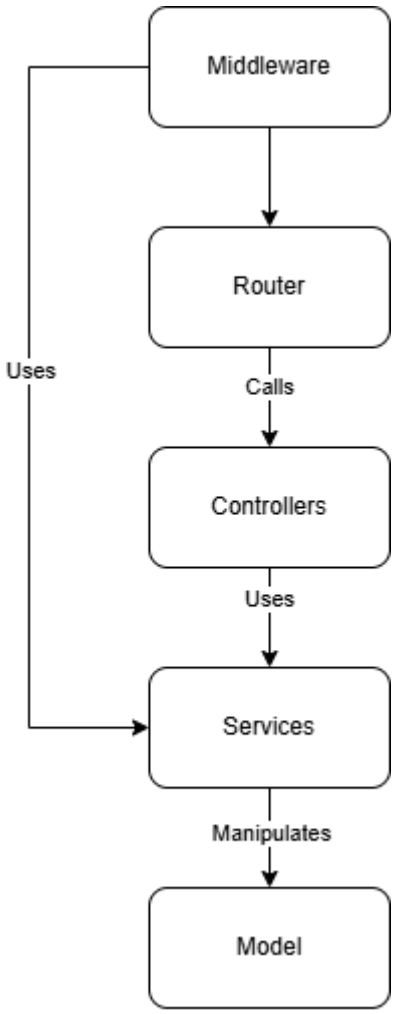
Koska jokaisella ympäristöllä on oma konfiguraatio ja ominaisuuskytkimet, voidaan helposti testata uusia ominaisuuksia alemmissa ympäristöissä. Yksi kytkin on vain totuusarvomuuttuja, joka määrittää, onko ominaisuus päällä vai ei (esimerkkikoodi 5).

```
const featureFlags: FeatureFlags = {
  prod: {
    normalImagePluginEnabled: false
  }
  dev: {
    normalImagePluginEnabled: true
  }
  // Rest of environments…
}
```

Esimerkkikoodi 5. Ominaisuuskytkin tuotanto- ja kehitysympäristöille.

4.2 Hallinnointirajapinta

Tässä luvussa esitellään hallinnointirajapinnan toteutus Express.js-kehyksellä. Kuvassa 7 on rajapinnan sisäinen arkkitehtuuri. Se noudattaa Model-Controller-Service-arkkitehtuuria, joka on variantti suositusta MVC-arkkitehtuurista.



Kuva 7. Hallinnointirajapinnan arkkitehtuuri.

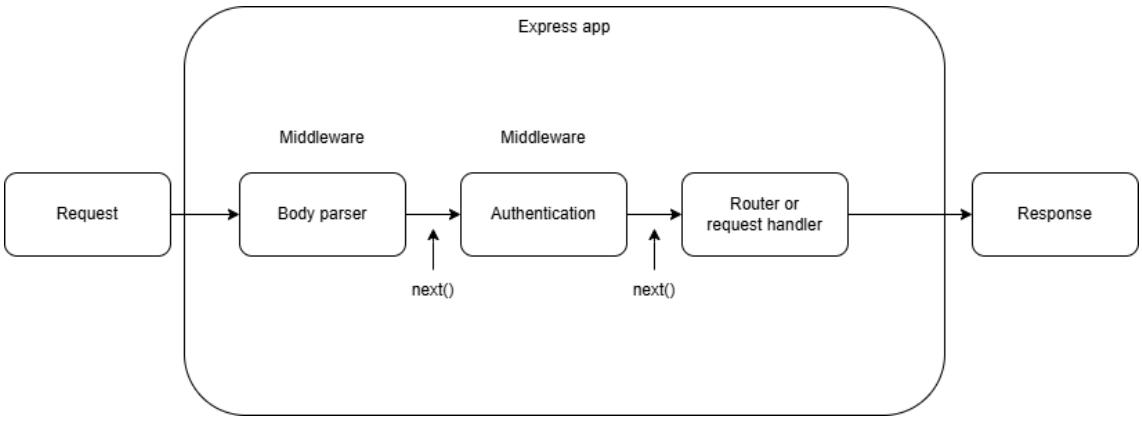
Jokainen kontrolleri ja palvelu on oma moduulinsa, joista jaetaan tarvittavat toiminnallisuudet funktioina. Tämä lisää koodin kytkentää, koska kutsuvat moduulit ovat suoraan riippuvaisia konkreettisista moduuleista. Kuitenkin yksikkötesteissä käytetään Vitest-kehystä, joka tekee kokonaisten moduulien simuloinnista helppoa.

4.2.1 Autentikaatio

Autentikaatioon käytetään BASIC-autentikaatiota, joka on yksinkertainen HTTP-protokollaan rakennettu autentikaatiomalli. Käytännössä pyynnön lähettäjä lisää pyynnön Authorization-otsakkeeseen merkkijonon, joka alkaa sanalla ”Basic ”, jonka jälkeen tulee ”username:password”-merkkijono Base64-enkoodattuna.

HTTP-protokollaa käytettäessä tämä ei ole tietoturvallista, koska Base64-enkoodaus on helppo dekoodata. Siksi BASIC-autentikaation kanssa käytetään myös HTTPS-protokollaa, jolla salataan pyynnön sisältö. [15.] Tietoturvaa tehostetaan AWS Security Groupeilla, joilla määritetään tarkalleen sallitut IP-osoitteet tai muut Security Groupit, jotka voivat kommunikoida palvelun kanssa. Lisäksi palvelu on SNDP-alustan yksityisessä verkossa.

Tyypillisesti autentikaatio tehdään Express.js-sovelluksessa middlewaren avulla. Middlewaret ovat funktioita, joita kehys suorittaa ennen pyynnön lopullista käsittelijää. Kuvasta 8 nähdään suorituksen eteneminen yhden pyynnön aikana.



Kuva 8. Pyynnön käsittelyn eteneminen Express.js-sovelluksessa.

Käyttäjiä säilytetään AWS Systems Manager Parameter Store -palvelussa, joka on käytännössä yksinkertainen avain-arvo-tietovarasto sovellusten konfiguraatioille. Käyttäjät ladataan sovelluksen muistiin @aws-sdk/client-ssm -kirjastolla. Sovellus pitää ladattuja käyttäjiä muistissa 15 minuuttia, jonka jälkeen ne haetaan uudelleen. Näin jokaisella pyynnöllä ei tarvitse hakea käyttäjiä uudelleen, ja käyttäjien lisääminen tai muokkaaminen ei vaadi sovelluksen uudelleenkäynnistämistä.

Käyttäjien hakeminen on käyttäjäpalvelun vastuulla. Moduuli tarjoaa yhden funktion, joka palauttaa listan Credential-olioita (esimerkkikoodi 2).

```
type Credential {
  username: string;
  password: string;
}
```

Esimerkkikoodi 6. Credential-tyyppi

Autentikaattori-middleware hakee käyttäjät ja vertaa niitä pyynnön Authorization-otsakkeesta luettuun käyttäjään. Mikäli käyttäjä löytyy muistista ja salasana on oikein, middleware ohjaa pyynnön eteenpäin.

4.2.2 Kuvien lähettäminen

Rajapinta tarjoaa kaksi tapaa lähettää uusia kuvia, jotka on listattu taulukossa 2. Lisäksi tarjotaan taaksepäin yhteensopivat variaatiot helpottamaan integraatiota.

Taulukko 2. Kuvien lataamiseen tarjottavat reitit.

HTTP-metodi	**Reitti**	**Kuvaus**
POST	/v2/images	Lataa kuva jolle palvelu generoi tunnisteen.
PUT	/v2/images/:imageId	Lataa kuva jonka tunniste on osoitteen reitissä.
POST	/v1/images/:anyproduct/images	Lataa kuva jolle palvelu generoi tunnisteen.
PUT	/v1/images/:anyproduct/images/:imageId	Lataa kuva jonka tunniste on osoitteen reitissä.

Kuvadata lähetetään pyynnön rungossa (engl. body). Rungon formaattina on multipart/form-data. Formaatilla voidaan lähettää erityyppistä tietoa yhdessä pyynnössä asettamalla tiedot osioihin, jotka erotetaan toisistaan pyynnön otsakkeissa määritetyn erottimen avulla (esimerkkikoodi 7). [16.] Tämän vuoksi se sopii kuvien lataukseen, koska kuvadatan lisäksi voidaan lähettää metadataa esimerkiksi JSON-formaatissa.

```
POST /upload HTTP/1.1
Host: example.com
Content-Type: multipart/form-data; boundary=------------------------
boundary123
Content-Length: [calculated size]

--------------------------boundary123
Content-Disposition: form-data; name="json_field"
Content-Type: application/json

{"key": "value"}
--------------------------boundary123
Content-Disposition: form-data; name="text_field"

Some plain text
--------------------------boundary123
Content-Disposition: form-data; name="file_field"; filename="file.txt"
Content-Type: text/plain

[contents of file.txt]
--------------------------boundary123--
```

Esimerkkikoodi 7. Yksinkertainen HTTP-pyyntö, jonka runko on multipart/form-data-formaatissa.

Express.js on minimalistinen eikä oletusasetuksilla parsi pyynnön runkoa. Parsimiseen voidaan esimerkiksi käyttää middlewarea. Palvelussa rungon parsimiseen käytetään Multer-kirjastoa, jonka jälkeen se annetaan validoitavaksi toiselle middlewarelle. Validointiin ei käytetä Multer-kirjastoa, koska se käyttää validointiin pelkästään pyynnön Content-Type-otsaketta. Lisäksi validoinnin epäonnistuessa kirjasto vastaa 500 virhekoodilla, joka semanttisesti tarkoittaa palvelinvirhettä. Validoinnin epäonnistuminen tarkoittaa, että lähetetty data on väärää, jolloin virhekoodi pitäisi olla 400.

Validoinnin jälkeen pyyntö siirretään lopulliselle käsittelijälle. Lähetysohjain tallentaa kuvan alkuperäisten kuvien tietovarastoon lähetyspalvelun funktioita kutsumalla. Esimerkkikoodissa 3 on kuvattu lähetyslogiikka ilman virheenkäsittelyä.

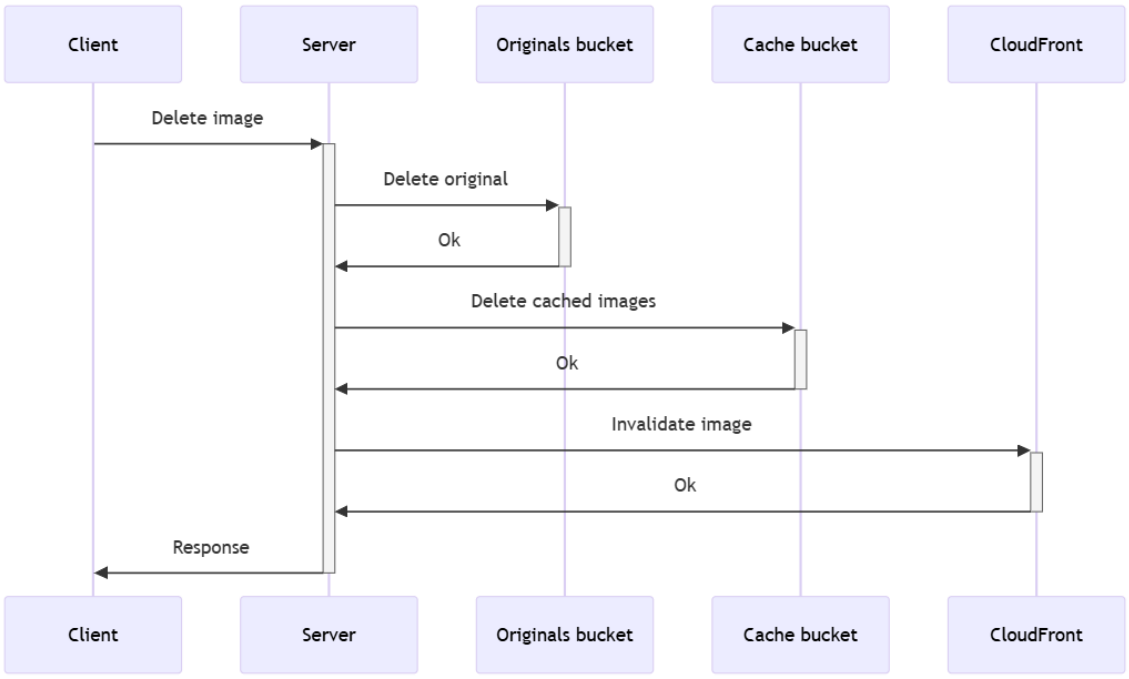
```
const buf = req.file.buffer;
const mimeType = req.file.mimetype;
const imageId = getImageId(params);

const uploadResponse = await uploadFile(imageId, buf, mimeType);

if (!uploadResponse) {
  // error handling logic
}

res.status(201).send(imageId)


// =====================


const createImageId = () => randomUUID().replaceAll('-', '');
const getImageId = (params: UploadParams) => params.imageId ?? cre-
ateImageId();
```

Esimerkkikoodi 8. Tiedostojen lähetyslogiikka ja tunnuksen generointiapufunktio.

Lähetysohjain tehdään geneeriseksi tukemaan PUT- ja POST-operaatiota, koska reiteissä vain tunnuksen generointilogiikka vaihtelee. Näin voidaan käyttää kaikille reiteille samaa ohjainta, joka helpottaa testaamista.

4.2.3 Kuvien poistaminen

Kuva poistetaan lähettämällä pyyntö DELETE-metodilla ”/v2/images/:imageId”-reittiin. Poistaminen tapahtuu kolmessa vaiheessa, jotka nähdään kuvasta 9. Ensimmäiseksi poistetaan alkuperäinen kuva. Sen jälkeen tyhjennetään välimuistitietovarastosta kaikki alkuperäisestä kuvasta generoidut kuvat. Lopuksi luodaan invalidaatio, joka poistaa kuvan CloudFrontista. Poistaminen pitää tapahtua juuri tässä järjestyksessä, jotta vaihdeiden välissä ei tallenneta uusia versioita välimuistiin.



Kuva 9. Kuvan poistamisen sekvenssikaavio.

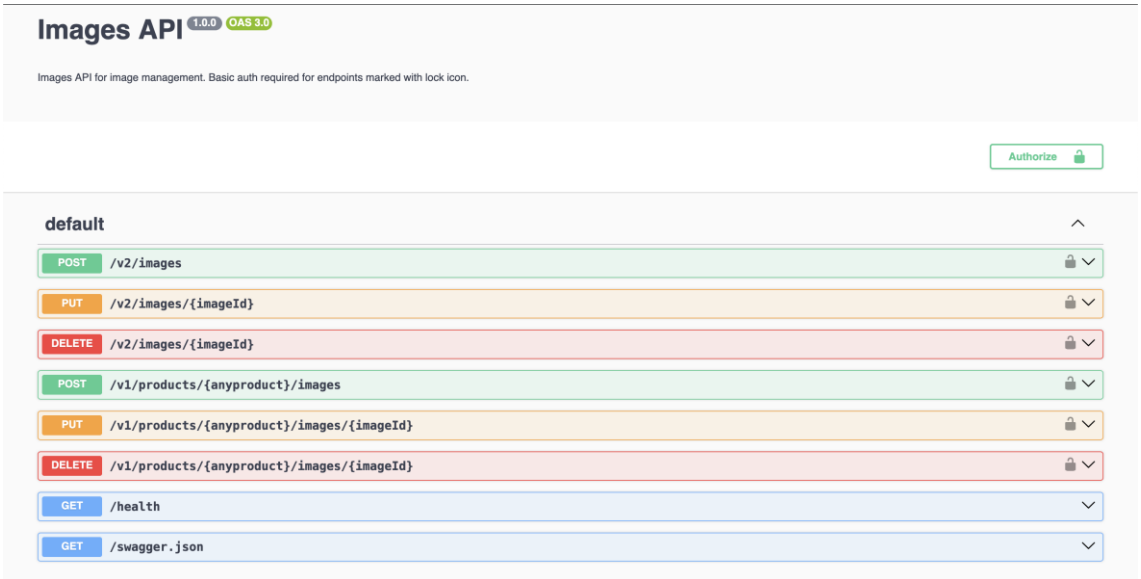
Operaatioita yritetään uudelleen niiden epäonnistuessa, mutta tämä ei takaa poistamisen onnistumista joka kerta. On mahdollista, että alkuperäisen kuvan poistaminen onnistuu, mutta välimuistien tyhjentäminen epäonnistuu. Tästä syystä on tärkeää tyhjentää välimuistit, vaikka alkuperäistä kuvaa ei löydy, jotta järjestelmä ei jää epävakaaseen tilaan, jolloin kehittäjän pitää käydä manuaalisesti poistamassa kuvat välimuistista.

Koska kuvien poistaminen on erittäin harvinaista, riittää poistamisen käyttöliittymäksi yksinkertainen shell-skripti. Tämä on jo huomattava parannus aikaisempaan järjestelmään, jossa kuvien poistaminen oli aina manuaalinen kehittäjän suorittama operaatio, koska käytetty URL-formaatti ei tukenut kuvien helppoa poistamista ja välimuistin tyhjennystä. Skripti ottaa argumentteina ympäristön nimen (dev, test tai prod) ja poistettavan kuvan tunnuksen. Suorituksessa haetaan ensin tunnukset tietovarastosta autentikaatiota varten, jonka jälkeen lähetetään poistopyyntö rajapinnalle.

4.2.4 Interaktiivinen API-dokumentaatio

OpenAPI-spesifikaatio on standardi, joka kuvaa rajapinnan kyvykkyyksiä ilman tarvetta lähdekoodin tai muun dokumentaation tarkastelulle. Rajapinnan ominaisuudet kuvataan OpenAPI-dokumentissa, joka on JSON- tai YAML-formaatissa. Dokumentin perusteella voidaan generoida muun muassa dokumentaatiota, asiakasohjelmia tai testejä. [17.] Yksi näistä generointityökaluista on SwaggerUI, jota käytetään OpenAPI-dokumentin interaktiiviseen visualisointiin.

OpenAPI-dokumenttien generoimiseen on olemassa monia eri kirjastoja. Dokumentteja voidaan esimerkiksi generoida TypeScript-tyyppien tai JavaDoc-kommentteja muistuttavien JsDoc-kommenttien perusteella. Hallinnointirajapinta on kuitenkin tarpeeksi yksinkertainen, että dokumentin kirjoittaminen käsin on helppoa. Rajapinnan OpenAPI-dokumentti tarjoillaan ”/swagger.json”-reitistä ja sen perusteella generoitu käyttöliittymä näkyy kuvassa 10.



Kuva 10. SwaggerUI-käyttöliittymä, joka on generoitu hallinnointirajapinnan OpenAPI-dokumentista.

OpenAPI-dokumentit on määritetty lähes kaikille SNDP-alustan palveluille ja ne on koostettu jaettuun dokumentaatioportaaliin. Osalla rajapinnoista on käyttäjiä SNDP-alustan ulkopuolella, jolloin portaali tarjoaa kevyen tavan oppia

käyttämään palvelua ilman koodin tarkastelua. Lisäksi hallinnointirajapinnan SwaggerUI-käyttöliittymä täydentää erinomaisesti muuta dokumentaatiota, joka on koodikannassa Markdown-tiedostoissa, kommenteissa ja Confluence-palvelussa.

5 Suorituskykytestaus ja optimointi

Tässä luvussa käsitellään kuvaskaalaimen suorituskykytestausta ja optimointia. Tämän lisäksi optimoidaan käytettäviä kuvia ottamalla uusi AVIF-kuvaformaatti käyttöön. Optimoinnin tavoitteena on parantaa kuvien laatua, löytää mahdollisia parannuksia suorituskykyyn ilman suuria muutoksia sovellukseen ja löytää sopiva tasapaino, kuvien laadun, suorituskyvyn ja hinnan välillä.

Kirjoitettua koodia ei optimoida, koska suorituksen jäljityksen perusteella Lambda-funktio käyttää suurimman osan ajasta kuvan generoimiseen tai tiedon siirtämiseen. Funktio on siis todennäköisesti prosessori- tai I/O-sidonnainen, jolloin merkittävimmät hyödyt saavutetaan, jos kuvien generointi nopeutuu, joko CPU-allokaatiota nostamalla tai generointiasetuksia tutkimalla.

5.1 Suorituskykytestin toteutus

Suorituskykytestauksen tehtävänä on tunnistaa regressiot suorituskyvyssä alemmissa ympäristössä ja tarjota kehys suorituskyvyn mittaamiseen ja muutosten vaikutusten vertailuun. Lisäksi suorituskykytestit täydentävät yksikkö- ja integraatiotestejä, mikä helpottaa jatkuvaa kehitystä.

Testit toteutetaan Grafana k6 -työkalulla, jonka tarjoama JavaScript-rajapinta mahdollistaa TypeScriptin käyttämisen [24]. Testissä lähetetään satunnaisia kuvahakuja, joiden reitit generoidaan tuotantodatan perusteella. Todennäköisyydet arvioidaan lokien perusteella ja laskemista varten toteutetaan skripti, joka generoi JavaScript-olioita, joissa on todennäköisyydet kuvatyypeille, leveyksille kuvatyypin mukaan ja formaatille kuvatyypin ja leveyden mukaan.

Grafana k6 testin suoritus jakautuu neljään vaiheeseen (taulukko 3). Testeissä toimii useita virtuaalisia käyttäjiä, jotka tekevät jotain samanaikaisesti. Käyttäjän suorittama tehtävä on JavaScript-funktio, jota kutsutaan jokaisella iteraatiolla.

Taulukko 3. K6-testin suorituksen vaiheet.

Vaihe	**Kuvaus**	**Kutsutaan**
Init	Lataa tiedostot ja alusta tarvittavat JavaScript-moduulit	Kerran per virtuaalikäyttäjä
Setup	Testeihin liittyvä alustuskoodi	Kerran
VU-code	Suoritettavaa skenaariota ajetaan toistuvasti	Kerran jokaisella iteraatiolla
Teardown	Mahdollinen validaatio tai raportointi testin lopussa	Kerran

Skenaarioiden avulla määritetään, mitä testissä suoritetaan ja miten iteraatioita ajastetaan. Testiin on määritetty neljä skenaariota, jotka ovat getCachedImages, getNormalImages, getSomeImages ja getSquareImages. Edeltävien skenaarioiden avulla on helppo hallita välimuistista haettujen kuvien määrää, ja se yksinkertaistaa satunnaisten kuvaosoitteiden generoimista.

Testi automoidaan suorittamalla sitä osana julkaisuprosessia AWS Codepipeline -palvelussa. Testi suoritetaan kehitysympäristöä vasten ja ne pitää läpäistä, jotta julkaisuprosessi jatkuu. Läpäisykriteerinä käytetään p95 vasteaikaa, koska se kuvaa, miten suurin osa käyttäjistä kokee järjestelmän toimivan. Käytännössä p95 tarkoittaa, että 95 % mittauksista on alle p95-arvon. Valinta tehdään tuotannon p95 vasteajan perusteella. Arvoksi asetetaan 600 millisekuntia, joka on noin 150 millisekuntia yläkanttiin, jotta pieni hajonta ei aiheuta haittaa.

5.2 Monitorointi ja metriikoiden kerääminen

Palvelujen monitorointi on tärkeä osa kehitysprosessia. Monitorointi havainnollistaa palvelun käyttäytymistä reaaliajassa. Kerätty tieto kuvaa, miten loppukäyttäjät käyttävät palvelua, ja sen perusteella voidaan tehdä johtopäätöksiä sekä suunnitella jatkokehitystä. Monitoroinnin avulla voidaan myös tunnistaa ja diagnosoida ongelmia helpommin ja nopeammin. Metriikoiden keräys voidaan jakaa viiteen yleiskategoriaan (taulukko 4). [23, s. 643-705.]

Taulukko 4. Metriikoiden eri kategoriat ja niiden selitteet.

Kategoria	**Kuvaus**	**Esimerkki**
Saatavuusmetriikat	Kuvaa kuinka usein palvelu tai sen osa toiminnallinen ja käytettävissä	Onnistuneiden pyyntöjen määrä suhteessa epäonnistuneisiin pyyntöihin
Liiketoiminnalliset metriikat	Kertoo miten palvelua käytetään ja tukee liiketoiminnan päätöksentekoa	Linkkien klikkaukset.
Sovellusmetriikat	Sovelluksen suorituskykyyn liittyvät metriikat kuten vasteaika tai virheet	Pyyntöjen vasteajat
Palvelin- tai infrastruktuurimetriikat	Järjestelmän resurssien tilaan ja käyttöön liittyvät metriikat	Sovelluksen CPU:n käyttö
Tiimimetriikat	Kehitysprosessiin liittyvät metriikat	Testikattavuus

Näistä kategorioista tärkeimmät kuvapalvelulle ovat saatavuus-, sovellus- ja infrastruktuurimetriikat. Tiimimetriikoita kerätään myös, mutta niiden keräämiseen on olemassa omat prosessit ja järjestelmät, jotka eivät liity tähän työhön.

Monitorointiin käytetään DataDog-palvelua, joka tarjoaa web-käyttöliittymän metriikoiden visualisointiin. Kyseinen palvelu on käytössä koko SNDP-alustalla ja tarjoaa kokonaiskuvan alustan toimintaan. Metriikoita kerätään AWS Cloudwatch -palvelun ja dd-trace-kirjaston avulla, joita käytetään metriikoiden keräämiseen ja lähettämiseen. Lokeja kerätään Winston-kirjaston avulla, joka kirjoittaa niitä JSON-formaatissa sovelluksen stdout-striimiin, josta ne lähetetään DataDog-palveluun. Koodissa monitorointi tarvitsee vain vähän konfigurointia, koska DataDog tarjoaa integraatioita eri AWS-palveluille kuten Lambdalle.

5.3 AVIF-kuvaformaatin käyttöönotto

AVIF eli AV1 Image File Format on uusi kuvaformaatti, jonka ensimmäinen versio julkaistiin 2019 [18]. AVIF-kuvat pakkautuvat tehokkaammin kuin JPEG tai Webp eli samaan määrään bittejä saadaan pakattua enemmän informaatiota [19; 20]. Formaatin avulla pitäisi siis pystyä parantamaan laatua ilman, että ladattujen tavujen määrä nousee. Tehokkaampi pakkaus on kuitenkin laskennallisesti työläämpää [21]. Kuvia tallennetaan välimuistiin, joten generoinnin hidastuminen ei pitäisi näkyä loppukäyttäjälle. Hidastumisen vaikutusta voidaan myös minimoida säätämällä Sharp-kirjaston skaalausasetuksia tai Lambda-funktion CPU-allokaatiota, kunnes löydetään sopiva suorituskyvyn ja laadun yhdistelmä.

Yleensä käytettyihin teknologioihin vaikuttaa niiden tuki päätelaitteissa. Työn tekemisen aikaan noin 94 % selaimista tukee AVIF-formaattia, joten sen käyttäminen on perusteltua [22].

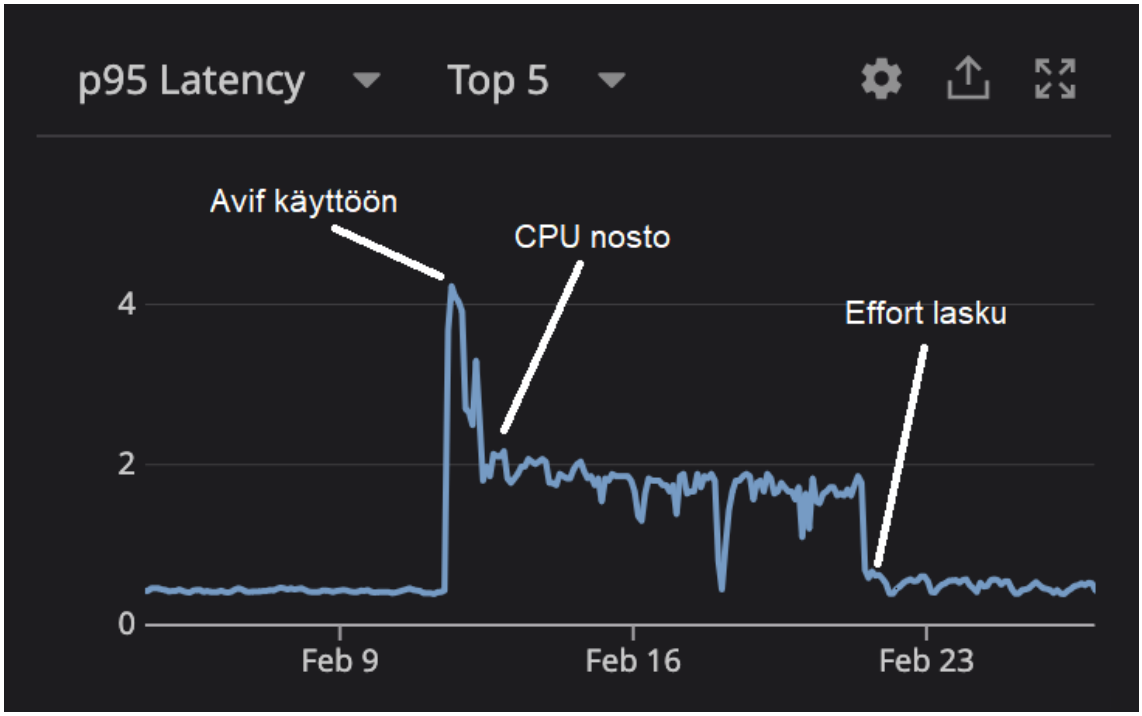
Formaatin käyttöönotto vaatii Lambdassa vain ”.avif”-päätteen hyväksymisen lisäosissa, joissa formaattia halutaan käyttää. Loput muutoksista tehdään frontend-palveluissa, joissa määritetään, mitä kuvia käytetään. Suuri osa palveluista käyttää Reactia, jonka avulla generoidaan responsiivisia HTML-kuvakomponentteja, joissa on käytettävät osoitteet. Esimerkkikoodissa 9 näkyy yksinkertaistettu responsiivisen kuvakomponentin HTML-koodi.

```
<picture>

<source srcset=" https://images.sanoma-
sndp.fi/6a18e75eff4705dfdf6c94ba976066c7/normal/658.avif 658w,
https://images.sanoma-
sndp.fi/6a18e75eff4705dfdf6c94ba976066c7/normal/978.avif 978w"
sizes="(max-width: 980px) 100vw, 640px" type="image/avif">

<source srcset="https://images.sanoma-
sndp.fi/6a18e75eff4705dfdf6c94ba976066c7/normal/658.webp 658w,
https://images.sanoma-sndp.fi/6a18e75eff4705dfdf6c94ba976066c7/nor-
mal/978.webp 978w" sizes="(max-width: 980px) 100vw, 640px" type="im-
age/webp">

<img alt="Kuvateksti" src="https://images.sanoma-
sndp.fi/6a18e75eff4705dfdf6c94ba976066c7/normal/468.jpg">

</picture>
```

Esimerkkikoodi 9. Responsiivinen HTML-kuvaelementti.

Kuvakomponentti sisältää source-elementtejä, joista selain valitsee ensimmäisen tuetun kuvaformaatin. Jos mikään source-tageissa olevista kuvista ei ole tuettu, niin selain käyttää img-elementin kuvaa. Eli jos AVIF on tuettu, niin käytetään sitä ja muissa tapauksissa valitaan Webp tai JPEG.

Käyttöönotto tehtiin ennen suorituskykytestausta, mutta silti manuaalisesti testatessa oli nähtävissä, että AVIF-kuvat ovat hitaampia generoida. Tämä vahvistui, kun muutokset otettiin käyttöön tuotannossa (kuva 11). Hitaus ei kuitenkaan aiheuttanut ongelmia, koska Lambda skaalautuu hyvin ja suositut kuvat löytyvät välimuistista. Suoritusta nopeutettiin nostamalla Lambda-funktion muistia, mikä nostaa CPU-allokaatiota samassa suhteessa. Lisäksi Sharp-kirjaston AVIF-kuvien skaalausasetuksista löytyi effort-parametri, jonka laskeminen neljästä kolmeen teki transformaatiosta noin 2 kertaa nopeampaa (kuva 11). Effort parametrin laskeminen johtaa huonompaan pakkaukseen, mutta manuaalisen testauksen perusteella vaikutukset kokoon jäävät pieneksi tässä tapauksessa.



Kuva 11. P95-vasteajat AVIF-käyttöönoton aikaan tuotannossa.

Optimoinnilla pyritään laskemaan p95-vasteaikaa, koska se vähentää vasteaikojen hajontaa. Pienempi hajonta taas tarkoittaa johdonmukaisempaa käyttäjäkokemusta.

5.4 Skaalaussuodattimien vertailu

Skaalaussuodatin tarkoittaa algoritmia, jota käytetään uudelleennäyttöönottoon (engl. resampling). Algoritmi määrittää, miten kuvan dimensioiden muuttaminen tapahtuu. Esimerkiksi kuvaa pienentäessä algoritmi päättää, miten pikseleitä jätetään kuvasta pois. Algoritmeja on monia ja niillä on erilaisia ominaisuuksia. Tässä luvussa vertaillaan näitä algoritmeja vain suorituskyvyn näkökulmasta.

Sharp-kirjasto tukee työn tekemisen aikaan viittä eri suodatinta kuvien pienentämiseen. Vertailu tehdään luvussa 5.1 toteutetun suorituskykytestin avulla. Kehitysympäristöön määritetään konfiguraatio käytettävälle suodattimelle ja testit ajetaan normaalisti osana julkaisuputkea. Taulukosta 5 nähdään testien tulokset. Vertailussa käytetään pyyntöjen keskimääräistä vasteaikaa ja p95-vasteaikaa, koska niiden avulla saadaan tarkempi kokonaiskuva järjestelmän suorituskyvystä.

Taulukko 5. Skaalaussuodatintestien tulokset.

Nimi	**Keskiarvo**	**P95**
Cubic	226.92ms	519.53ms
Nearest	227.85ms	508.17ms
Mitchell	224.51ms	516.04ms
Lanczos 2	223.11ms	509.84ms
Lanczos 3 (oletus)	222.85ms	515.85ms

Tuloksista nähdään, että Sharp-kirjaston tarjoamilla suodattimilla ei ole tässä tapauksessa suurta vaikutusta vasteaikoihin. Samanlaisia tuloksia saadaan myös kehittäjän tietokoneella yhtä kuvaa skaalattaessa. Tulokset voivat johtua siitä, että muut tekijät, kuten kuvien pakkaus ja I/O-nopeus vaikuttavat Lambdan

suorituskykyyn paljon enemmän kuin käytetty suodatin. On myös mahdollista, että tuloksiin vaikuttaa liiallinen melu tai taustalla voi olla virhe konfiguraatiossa tai mittausmenetelmässä. Tulosten perusteella valitaan käytettäväksi Sharp-kirjaston oletusarvo eli Lanczos 3 -algoritmi, koska kuvien laatuun ollaan tyytyväisiä.

6 Yhteenveto

Vaikka SNDP-alusta on suuri kokonaisuus, palvelu oli melko yksinkertainen toteuttaa, koska se on itsenäinen komponentti. Tätä insinöörityötä olisikin mahdollista soveltaa samankaltaisissa tilanteissa, joissa tarvitaan responsiivisia kuvia. Sovellettavuus perustuu myös skaalautuvuuteen ja kustannustehokkuuteen. Koska palvelun resursointi ja kustannukset perustuvat kulutukseen, se on myös sovelias pienille tai kasvaville yrityksille.

6.1 Tulosten tarkastelu

Insinöörityössä toteutettiin uusi kuvapalvelu, joka noudattaa SNDP-alustan parhaita käytäntöjä, joista olennaisimmat ovat ylläpidettävyys, kehitettävyys ja hyvä kehittäjäkokemus. Palvelu tukee vanhan palvelun ominaisuuksien lisäksi uutta AVIF-kuvaformaattia ja tarjoaa kyvykkyydet kuvajournalismin kehittämiseen tulevaisuudessa. Tuotannossa ei ole ensimmäisestä päivästä lähtien ilmennyt merkittäviä ongelmia, mikä kertoo palvelun luotettavuudesta ja skaalautuvuudesta. Kaikki toimeksiantajan 15 uutisbrändiä käyttää palvelua, ja alkuperäisiä kuvia on tällä hetkellä tietovarastossa noin 7,2 miljoonaa kappaletta, jotka vievät noin 3 teratavua tilaa. Uutisbrändien suuri liikenteen määrä näkyy myös kuvapalvelussa. Palvelun CDN:stä haetaan päivittäin noin 360 miljoonaa kuvaa yhteensä noin 20 teratavun edestä.

Palvelun korkean tason arkkitehtuurissa hyödynnetään moderneja AWS-ratkaisuja, jotka ovat skaalautuvia ja ylläpidettäviä. Lisäksi käytössä on suosittuja teknologioita ja riippuvuuksien määrä on pyritty pitämään pienenä. Testikokonaisuus koostuu yksikkö-, integraatio- ja suorityskykytesteistä, joiden ajaminen julkaisuputkessa validoi palvelun toiminnan ja tekee kehityksestä luotettavaa. Monitoroinnin avulla on helppo tarkastella palvelun käyttöä ja diagnosoida mahdollisia ongelmia. Edellä mainittujen ominaisuuksien perusteella voidaan sanoa, että toteutettu palvelu täyttää sille annetut vaatimukset.

Palvelun käyttöönoton jälkeen optimoitiin kuvia ottamalla käyttöön AVIF-kuvaformaatti, jonka jälkeen nostettiin Lambdan CPU-allokaatiota ja säädettiin

AVIF-kuvien generointiasetuksia. Näillä toimenpiteillä saatiin palvelun suorituskyky hyväksyttävälle tasolle. Käyttöönoton lopputuloksena kuvien laatu pitäisi olla parempi sivustoilla. Laadun validoimisessa haasteena on subjektiivisuus, mutta yksittäisiä kuvia vertailtaessa nähdään AVIF-kuvissa selvästi vähemmän artefakteja varsinkin pienillä leveyksillä. Kuvien laatua parannettiin myös nostamalla käytettyjen kuvien leveyksiä.

Palvelulle toteutettiin myös suorituskykytesti, jota käytetään regressioiden havaitsemiseen ja muutosten vaikutusten mittaamiseen. Lopuksi vertailtiin Sharp-kirjaston skaalaussuodattimia toteutetulla kehyksellä. Vertailun lopputuloksena todettiin, että suodattimilla ei ole tässä tapauksessa merkittävää vaikutusta suorituskykyyn, joten Sharp-kirjaston oletusarvo Lanczos 3 on sopiva.

6.2 Vertailua aikaisempaan palveluun

Taulukossa 6 on vertailua palvelujen välillä. Uudessa palvelussa on huomioitu korvattavan palvelun heikkoudet ja pyritty korjaamaan niitä. Siksi vertailusta nähdään, että suurimmat erot ovat juuri ylläpidettävyydessä ja kehittäjäkokemuksessa.

Taulukko 6. SIDP ja Images vertailua.

Ominaisuus	**SIDP**	**Images**
Haavoittuvuudet	Avoimia haavoittuvuuksia	Ei avoimia haavoittuvuuksia
Vasteaika (p95)	700ms	407ms
Jatkuva kehitys mahdollista	Ei	Kyllä
Hyvä testikattavuus	Ei	Kyllä
AVIF-tuki	Ei	Kyllä
Skaalautuvuus (kuvien lähetys päätelaitteille)	Uusien konttien käynnistämiseen menee minuutteja	Lambda-instanssit käynnistyy sekunneissa

Kuvien poistaminen	Manuaalinen operaatio	Yksi komento terminaalissa
Lokaali kehitys	Ei	Kyllä

Palvelujen kustannuksissa ei ole suuria eroja, jos otetaan huomioon uudet ominaisuudet. Ennen AVIF-kuvien käyttöönottoa Images maksoi suunnilleen saman verran kuin SIDP, mutta sen jälkeen hieman enemmän, koska Lambdan muistia nostettiin. Jos SIDP tukisi AVIF-kuvia, se olisi todennäköisesti myös kalliimpi.

6.3 Haasteet ja palvelun jatkokehitys

Suurimmat haasteet ilmenivät palvelun integraatiossa, jota varten piti ymmärtää, miten eri palvelut käyttävät kuvia. Palvelun käyttöönotto tapahtui kuitenkin ilman suuria ongelmia. Alussa ilmeni yksittäisiä ei halutun näköisiä kuvia, jotka oli helppo korjata kuva-asetuksia säätämällä.

Palvelua mahdollisesti kehitetään tulevaisuudessa tukemaan uusia kuvatyyppejä. Sharp-kirjasto tukee esimerkiksi kuvien yhdistämistä, jonka avulla on mahdollista vähentää manuaalista työtä. Monitorointia voisi myös kehittää keräämällä tarkempaa metriikkaa, jonka voisi koostaa kojelaudaksi DataDogissa. Sharp-kirjaston käyttämä libvips-kirjasto on myös jatkuvassa kehityksessä ja on mahdollista, että esimerkiksi AVIF-kuvien käsittely paranee tulevaisuudessa.

Insinöörityössä parannettiin kuvien laatua uuden kuvaformaatin ja leveyden noston avulla. Sharp-kirjastossa on paljon asetuksia, jotka vaikuttavat kuvanlaatuun. Siksi tulevaisuudessa olisi mahdollista selvittää kvalitatiivisin menetelmin, voisiko kuvien laatua parantaa esimerkiksi skaalaussuodattimien avulla.

Tulevaisuudessa kuvaskaalaimessa voisi vertailla jotain muuta teknologiaa, jossa CPU-allokaatio suoraan kehittäjän hallittavissa, koska Lambda-funktio on edelleen prosessori- tai I/O-sidonnainen ja nopeutus kasvattaa hintaa lineaarisesti. Myös kuvien välimuistitietovaraston käytössä on parannettavaa,

koska tämä hetkinen 30 päivän säilytysaika on hatusta vedetty arvo. Säilytysajan nostaminen todennäköisesti laskee Lambdan kustannuksia, koska kuvia generoidaan vähemmän. Optimaalinen säilytysaika löytyy, kun säilytyskustannukset ovat yhtä suuret kuin Lambdan kustannukset.

Lähteet

1 A brief history of images on the web. Verkkoaineisto. <https://web.dev/learn/images/history> Päivitetty 1.2.2023. Luettu 16.2.2025.

2 Ford, Neal & Richards, Mark. 2020. Fundamentals of Software Architecture: An Engineering Approach. O'Reilly Media, Inc.

3 Amazon Simple Storage Service. Verkkoaineisto. Amazon Web Services. <https://docs.aws.amazon.com/pdfs/AmazonS3/latest/userguide/s3-userguide.pdf> Päivitetty 31.3.2025. Luettu 20.2.2025.

4 AWS Lambda. Verkkoaineisto. Amazon Web Services. <https://docs.aws.amazon.com/pdfs/lambda/latest/dg/lambda-dg.pdf> Päivitetty 20.2.2025. Luettu 20.2.2025.

5 Amazon Elastic Container Service. Verkkoaineisto. Amazon Web Services. <https://docs.aws.amazon.com/pdfs/AmazonECS/latest/developerguide/ecs-dg.pdf> Päivitetty 13.20.2025. Luettu 20.2.2025.

6 Amazon CloudFront. Verkkoaineisto. Amazon Web Services. <https://docs.aws.amazon.com/pdfs/AmazonCloudFront/latest/DeveloperGuide/AmazonCloudFront_DevGuide.pdf> Päivitetty 7.4.2025. Luettu 20.2.2025.

7 AWS Identity and Access Management. Verkkoaineisto. Amazon Web Services. <https://docs.aws.amazon.com/pdfs/IAM/latest/UserGuide/iam-ug.pdf> Päivitetty 31.3.2025. Luettu 8.3.2025.

8 Holmberg, Oliver. 2023. Migrating from JavaScript to TypeScript and its advantages. Opinnäytetyö. Metropolia Ammattikorkeakoulu. Theseus-tietokanta.

9 Developer Survey 2024. 2024. Verkkoaineisto. Stack Overflow. <https://survey.stackoverflow.co/2024/technology/> Luettu 21.2.2025.

10 TypeScript config reference. Verkkoaineisto. TypeScript <https://www.typescriptlang.org/tsconfig/#target> Luettu 17.4.2025.

11 ECMAScript compatibility table. Verkkoaineisto. <https://compat-table.github.io/compat-table/es2016plus> Päivitetty 16.12.2024. Luettu 17.4.2025.

12 Fuller, Lovell. Sharp. Verkkoaineisto. <https://www.npmjs.com/package/sharp> Luettu 15.2.2025.

13 Express/Node introduction. Verkkoaineisto. MDN Web Docs. <https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Server-side/Express_Nodejs/Introduction> Päivitetty 11.4.2025. Luettu 21.2.2025.

14 Hammant, Paul & Smith, Steve. Trunk Based Development: Introduction. Verkkoaineisto. <https://trunkbaseddevelopment.com> Päivitetty 9.12.2024. Luettu 9.3.2025.

15 RFC 7617-2015. The 'Basic' HTTP Authentication Scheme. Revision of RFC 2617. IETF.

16 RFC 7578-2015. Returning Values from Forms: multipart/form-data. Revision of RFC 2388. IETF.

17 OpenAPI Specification v3.1.0. 2021. Verkkoaineisto. OpenAPI Initiative, The Linux Foundation. <https://spec.openapis.org/oas/v3.1.0.html> Luettu 5.4.2025.

18 AV1 Image File Format (AVIF). Verkkoaineisto. <https://aomediacodec.github.io/av1-avif/> Päivitetty 8.1.2025. Luettu 15.3.2025

19 Mavlankar, Aditya & De Cock, Jan & Concolato, Cyril & Swanson, Kyle & Moorthy, Anush & Aaron, Anne. AVIF for Next-Generation Image Coding. 2020. Verkkoaineisto. Netflix Technology Blog. <https://netflixtechblog.com/avif-for-next-generation-image-coding-b1d75675fe4> Luettu 15.3.2025.

20 Barman, Nabajeet & Martini, Maria G. 2020. An Evaluation of the Next-Generation Image Coding Standard AVIF.

21 Mumtaz, Muhammad Rafly & Bachmid, Muhammad Avied. 2024. Performace Analysis of Lossy Image Formats with Huffman Encoding Across Different Resolutions

22 Deveria, Alexis. Can I Use AVIF. Verkkoaineisto. <https://caniuse.com/?search=AVIF> Päivitetty 25.3.2025. Luettu 15.3.2025.

23 Yevgeniy, Brikman. 2025. Fundamentals of DevOps and Software Delivery. O'Reilly Media Inc.

24 Grafana k6 dokumentaatio. Verkkoaineisto. Grafana Labs. <https://grafana.com/docs/k6/latest/> Luettu 16.3.2025.