Tampereen ammattikorkeakoulu

Käyttöohjeavustin RAG-teknologialla

Henri Sjöblom

OPINNÄYTETYÖ
Kesäkuu 2025

Tietotekniikan tutkinto-ohjelma
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma
Ohjelmistotekniikka

SJÖBLOM, HENRI:
Käyttöohjeavustin RAG-teknologialla

Opinnäytetyö 33 sivua, joista liitteitä 4 sivua
Kesäkuu 2025

Tekoälyn kyvykkyys ymmärtää ja tuottaa luonnollista kieltä on kehittynyt paljon suurten kielimallien ansiosta. Nämä mallit ovat valtavalla määrällä dataa koulutettuja tekoälymalleja, jotka pystyvät yleistämään oppimansa uusiin tehtäviin. Esimerkiksi OpenAI:n GPT-kielimallit ovat mahdollistaneet uusia sovelluksia niin liiketoiminnassa kuin arkipäivän tehtävissä kuten asiakaspalvelun automatisoinnissa ja ohjelmoinnin avustamisessa. Suurten kielimallien käytössä ilmenee kuitenkin haasteita. Mallit voivat hallusinoida eli tuottaa virheellistä tietoa, mikä on erityisen haitallista tarkkuutta vaativissa sovelluksissa kuten käyttöohjeavustimissa. RAG (Retrieval-Augmented Generation) -teknologia tarjoaa ratkaisun vähentämällä hallusinaatioiden riskiä hankkimalla ajantasaisempaa tietoa ulkoisista lähteistä ennen vastausten muodostamista.

Opinnäytetyön tavoitteena oli kehittää Python-ohjelmointikielellä käyttöohjeavustin, joka hyödyntää RAG-teknologiaa ja mikropalveluarkkitehtuuria. Käyttöohje annetaan sovellukselle PDF-tiedostona, joka luetaan tietokantaan. Suuri kielimalli tuottaa vastauksen hyödyntämällä tietokannasta haettua kysymykseen liittyvää tietoa.

Työssä onnistuttiin pilkkomaan RAG-sovellus mikropalveluiksi. Sovellus pystyy vastaamaan käyttöohjeeseen liittyviin kysymyksiin yksityiskohtaisemmin ja tarkemmin kuin perinteinen suuri kielimalli, ja sen tuottamat vastaukset ovat ajantasaisempia. Sovellus kieltäytyy lähtökohtaisesti vastaamasta kysymyksiin, ellei kielimalli katso saavansa riittävästi dataa käyttöohjeesta. Siten hallusinoinnin riski on huomattavasti pienempi kuin pelkällä perinteisellä suurella kielimallilla. Mikropalveluarkkitehtuuri nopeutti ja joustavoitti kehitystä sekä lisäsi sovelluksen skaalautuvuutta, sillä eri toiminnallisuudet voitiin kehittää ja ylläpitää itsenäisesti.

Tulevaisuudessa sovellusta voidaan kehittää lisäämällä tuki eri suurille kielimalleille sekä personoimalla käyttäjäkokemusta autentikoinnin ja datan keräämisen avulla. Kehityksessä on huomioitava eettiset ja tietosuojanäkökohdat erityisesti käyttäjädatan käsittelyssä.

Asiasanat: tekoäly, koneoppiminen, python

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in ICT Engineering
Software Engineering

SJÖBLOM HENRI
A User Manual Assistant with RAG Technology

Bachelor's thesis 33 pages, appendices 4 pages
June 2025

Artificial intelligence's (AI) capability to understand and generate natural language has greatly improved due to large language models (LLMs). These are AI models trained on massive datasets, enabling them to generalize their learning to new tasks. However, the deployment of these large models is not without challenges. A significant issue is model hallucination, where the models can sometimes generate incorrect information. This is particularly problematic for tasks that need accurate details, such as helping with user manuals. Retrieval-Augmented Generation (RAG) technology helps solve the problem of hallucinations by using up-to-date information from outside sources before creating answers.

The goal of this project was to build a user manual assistant using Python, RAG technology, and a microservice architecture. The application ingests user manuals in PDF format and stores them in a database. A large language model then creates answers using related information from this database.

The RAG application was successfully implemented as a microservice-based system, providing more accurate and up-to-date answers than a regular LLM. It refuses to answer without enough manual information, greatly reducing hallucinations. The microservice architecture also made development faster, more flexible, and the application more scalable.

Future enhancements include diverse LLMs, personalization, data ethics.

Key words: ai, machine learning, python

SISÄLLYS

1 JOHDANTO ... 6
2 RAG JA MIKROPALVELUARKKITEHTUURI ... 8
2.1 RAG ... 8
2.1.1 Suuret kielimallit ... 8
2.1.2 RAG-tekoäly ... 9
2.1.3 Arkkitehtuuri ... 9
2.1.4 RAG-teknologian tarpeellisuus ... 10
2.2 Mikropalveluarkkitehtuuri ... 11
3 KÄYTTÖOHJEAVUSTIMEN TOTEUTUS ... 13
3.1 Mikropalveluarkkitehtuuri ... 13
3.2 RAG-palvelu ... 15
3.3 Generointipalvelu ... 17
3.4 Tiedonhakupalvelu ... 19
3.5 Ingestion-palvelu ... 21
3.6 Chroma-vektoritietokanta ... 23
3.7 Testaaminen ... 24
3.8 Käyttöönotto ... 25
4 POHDINTA ... 26
LÄHTEET ... 28
LIITTEET ... 30
Liite 1. RAG-palvelun tiedostorakenne ... 30
Liite 2. Generointipalvelun tiedostorakenne ... 31
Liite 3. Tiedonhakupalvelun tiedostorakenne ... 32
Liite 4. Ingestion-palvelun tiedostorakenne ... 33

ERITYISSANASTO

REST-API	Ohjelmistorajapinta
Chroma	Vektoritietokanta
Mikropalvelu-arkkitehtuuri	Sovellus pilkotaan pienempiin osiin eli mikropalveluiksi
Monoliittinen-arkkitehtuuri	Sovellus koostuu yhdestä osasta
Prompti	Kehote, joka syötetään suurelle kielimallille
RAG	Retrieval-augmented generation
Feature Engineering	Piirteiden poimimista raakadatasta ja niiden muuntamista muotoihin, jotka soveltuvat koneoppimismallille.

1 JOHDANTO

Tekoälyn kehityksessä on pitkään pyritty parantamaan tekoälyn kyvykkyyttä ymmärtää ja tuottaa luonnollista kieltä. Suurten kielimallien ansiosta tekoäly otti ison kehitysloikan, sillä ne ovat generatiivisia eli pystyvät yleistämään oppimaansa uusiin tehtäviin. (Hämäläinen 2024, 81). Mallit kuten OpenAI:n GPT-4 ovat mahdollistaneet uusia sovelluksia laajasti liiketoiminnasta arkipäivän toimintoihin. Esimerkiksi kielimalleja käytetään automatisoimaan asiakaspalvelua sekä koodinkehityksessä ohjelmoijan apuna.

Suurten kielimallien käytössä on kuitenkin merkittäviä haasteita. Kielimallit voivat hallusinoida, jolloin malli tuottaa virheellistä tai väärennettyä tietoa. Tämä on erityisen kriittistä sovelluksissa, joissa tarkkuus ja luotettavuus ovat olennaisia kuten käyttöohjeavustimissa. Esimerkiksi, jos kielimallia ei ole koulutettu tiedolla uusimpien puhelimien käytöstä, voi se antaa virheellisiä ohjeita laitteen käytöstä. Virheelliset ohjeet voivat johtaa käyttäjän tyytymättömyyteen ja pahimmillaan jopa vaaratilanteisiin.

RAG (retrieval-augmented generation) -teknologian avulla kielimallien ongelmia voidaan vähentää. Teknologiassa perinteiselle generatiiviselle mallille annetaan uutta tietoa ulkoisista lähteistä ennen vastauksen tuottamista, mikä mahdollistaa tarkempien ja ajantasaisempien vastausten tuottamisen. RAG:n avulla käyttöohjeavustin voi hakea uuden puhelinmallin käyttöohjeista oleellista tietoa ja käyttää sitä luodakseen tarkkoja vastauksia käyttäjän kysymyksiin, jolloin hallusinointia voidaan vähentää merkittävästi.

Tämän työn tavoitteena on kehittää käyttöohjeavustin, joka hyödyntää RAG-tekoälyä. Sovellus tehdään Pythonilla ja se tarjoaa API:n sovelluksen käyttöön. Sovellukselle annetaan käyttöohje PDF-tiedostona, josta sovellus tallentaa datan tietokantaan. Käyttäjä voi esittää kysymyksiä tekoälylle, jolloin tietokannasta haetaan kysymykseen liittyvää olennaista dataa tekoälyn avuksi. Uuden datan avulla tekoäly osaa antaa tarkempia ja yksityiskohtaisempia vastauksia sekä riski hallusinaatioon on pienempi.

Sovelluksessa käytetään mikropalveluarkkitehtuuria, jossa sovellus pilkotaan pienemmiksi itsenäisiksi osiksi. Jokainen osa vastaa tietystä toiminnallisuudesta ja kommunikoi muiden sovellusten kanssa API:n kautta.

Kappaleessa kaksi tutustutaan RAG:n ja mikropalveluiden teoriaan, ja kappaleessa kolme on sovelluksen toteutus. Viimeisessä kappaleessa pohditaan työstä saatuja oppeja sekä jatkokehitysmahdollisuuksia.

2 RAG JA MIKROPALVELUARKKITEHTUURI

2.1 RAG

Chen ym. (2017) esittelivät RAG-mallin (eng. retrieval-augmented generation) ensimmäistä kertaa artikkelissa ”Reading Wikipedia to Answer Open-Domain Questions”. Artikkelin järjestelmässä hyödynnettiin Wikipedia-artikkeleita tiedonhakuun. Tekijät havaitsivat, että RAG-mallit tuottavat tarkempaa, monipuolisempaa ja faktapitoisempaa kieltä tekstin tuottamisessa verrattuna parhaisiin perinteisiin kielimalleihin. (Chen ym. 2017, 1)

Lewis ym. (2020) ottivat ensimmäisenä käyttöön termin ”retrieval-augmented generation” artikkelissa “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks” Artikkelissa ehdotettiin RAG-menetelmää ratkaisuksi tietointensiivisiin tehtäviin, joissa kaikkea saatavilla olevaa tietoa ei voida syöttää mallille suoraan. RAG:n avulla vain hakijan tunnistama olennaisin tieto haetaan ja syötetään malliin. (Lewis ym. 2020, 2).

RAG-mallilla on useita etuja verrattuna perinteiseen laajaan kielimalliin. Lewis ym. (2020) näkevät etuina hallusinaation vähentymisen ja faktoihin perustuvien vastauksien mahdollistaminen (Lewis ym. 2020, 2).

2.1.1 Suuret kielimallit

Kielimallit ovat laskennallisia malleja, joilla on kyky ymmärtää ja tuottaa luonnollista kieltä. Suuret kielimallit ovat kehittyneitä kielimalleja, joissa on poikkeukselliset oppimiskyvyt ja ne koostuvat valtavasta määrästä parametreja. (Chang ym. 2024, 4).

Transformer-arkkitehtuurin esitteli Vaswani ym. (2017) ja se perustuu itsehuomioon (eng. self-attention) (Vaswani ym. 2017, 2). Suuret kielimallit perustuvat transformer-arkkitehtuuriin ja ne ovat mullistaneet luonnollisen kielen käsittelyn kyvyllään käsitellä peräkkäistä dataa tehokkaasti. Ne pystyvät hahmottamaan riippuvuuksia pitkästä tekstistä, vaikka yhdistettävät asiat olisivat kaukana toisistaan. (Chang ym. 2024, 4-5)

Yksi suurten kielimallien keskeinen ominaisuus on kontekstuaalinen oppiminen (eng. in-context learning), jossa malli opetetaan tuottamaan tekstiä annetun kontekstin tai promptin perusteella. Tämän ansiosta suuret kielimallit voivat tuottaa yhtenäisempiä ja kontekstiin sopivampia vastauksia, mikä tekee niistä erityisen sopivia vuorovaikutteisiin ja keskusteleviin sovelluksiin. (Chang ym. 2024, 5).

2.1.2 RAG-tekoäly

RAG:ia voi ajatella teknologiana, jonka avulla luodaan kyselykohtainen syötekonteksti, sen sijaan että käytettäisiin samaa kontekstia kaikkiin kyselyihin. Tämä auttaa käyttäjädatan hallinnassa, sillä näin voidaan lisätä käyttäjäkohtaiset tiedot vain kyseistä käyttäjää koskeviin kyselyihin. (Huyen, 2024)

Huyen (2024) mukaan kontekstin muodostaminen foundation-malleille (eng. foundation models) vastaa feature engineeringiä klassisille koneoppimismalleille. Tarkoitus on sama eli antaa mallille tarvittava tieto syötteen käsittelyyn (Huyen 2024, luku 6). Feature engineering tarkoittaa piirteiden (eng. features) poimimista raakadatasta ja niiden muuntamista muotoihin, jotka soveltuvat koneoppimismallille. Muuntamiseen voidaan käyttää esimerkiksi datan normalisoimista tai skaalausta. (Zheng & Casari 2018, luku 2)

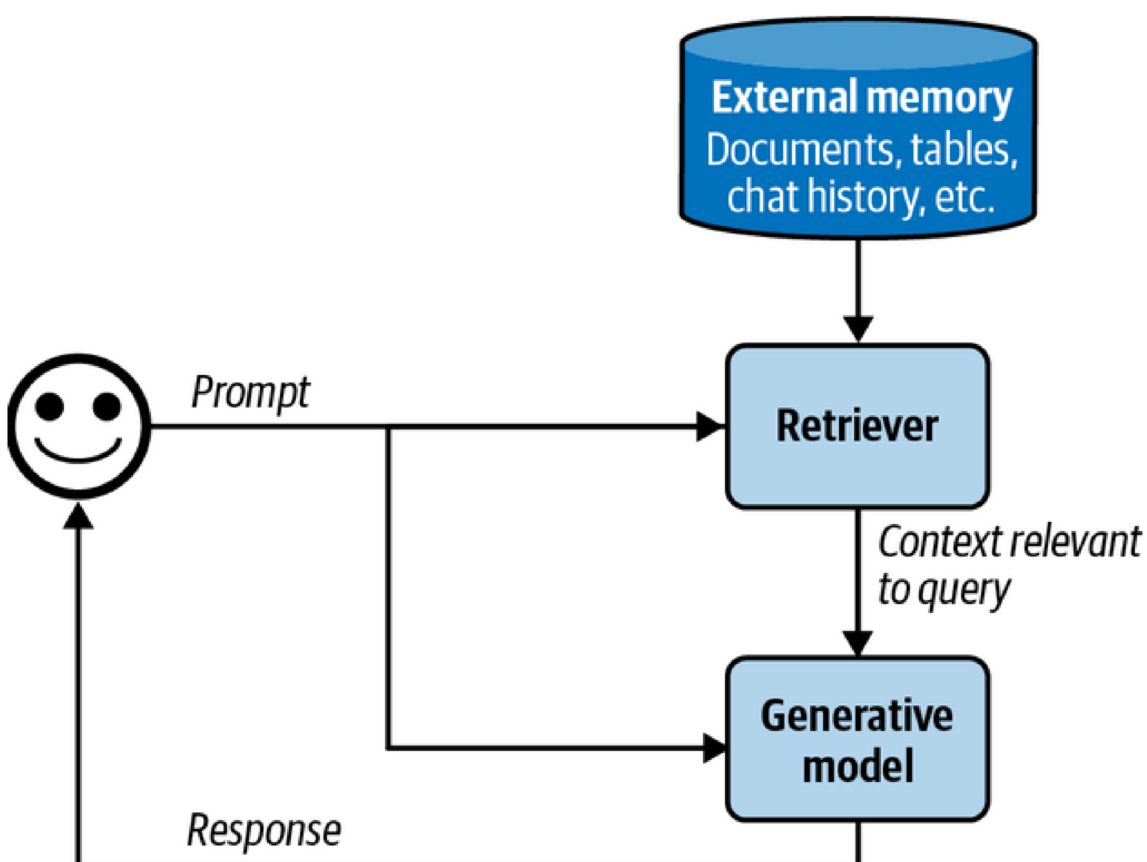
2.1.3 Arkkitehtuuri

RAG-järjestelmässä on kaksi osaa hakija (eng. retriever) ja generaattori (eng. generator). Hakija hakee tietoa ulkoisista muistilähteistä. Generaattori tuottaa vastauksen haetun tiedon perusteella. (Huyen 2024, luku 6)

Alkuperäisessä RAG-artikkelissa Lewis ym. (2020). kouluttivat hakijan ja generaattorin yhdessä. Nykyisissä RAG-järjestelmissä nämä kaksi komponenttia koulutetaan usein erikseen. Monet tiimit rakentavat RAG-järjestelmiään käyttämällä valmiita hakukoneita ja malleja. Koko RAG-järjestelmän hienosäätö päästä päähän voi kuitenkin parantaa suorituskykyä merkittävästi. (Huyen 2024, luku 6)

RAG-järjestelmän suorituskyky riippuu hakijan kyvykkyydestä. Hakijalla on kaksi päätoimintoa: indeksointi ja haku. Indeksointi tarkoittaa datan käsittelyä siten, että se voidaan myöhemmin hakea nopeasti. Kun lähetetään kysely tietojen hakemiseksi, puhutaan hausta (eng. querying). Datan indeksointi riippuu siitä, miten sitä halutaan hakea myöhemmin. (Huyen 2024, luku 6)

Kuviossa 1 on RAG-prosessin arkkitehtuuri. Käyttäjä tekee promptin, joka annetaan hakijalle ja generaattorille. Hakija etsii ulkoisesta tiedonlähteestä promptiin liittyvää dataa ja antaa sen generaattorille. Generaattori tuottaa vastauksen promptin ja hakijalta saadun datan perusteella.



Kuvio 1: RAG-prosessin arkkitehtuuri (Huyen 2024 luku 6).

2.1.4 RAG-teknologian tarpeellisuus

Foundation-mallien alkuvaiheessa RAG nousi yhdeksi yleisimmistä malleista. Sen päätarkoitus oli kiertää laajojen kielimallien syötekontekstirajoitukset. Monet ajattelevat, että riittävän pitkä konteksti tekee RAG:ista tarpeettoman. (Huyen 2024, luku 6)

Huyen (2024) ei usko, että pitkä konteksti poistaa tarpeen RAG-teknologialle: riippumatta siitä, kuinka pitkä mallin kontekstipituus on, on olemassa sovelluksia, jotka tarvitsevat tätä pidemmän kontekstin. Lisäksi käytettävän datan määrä kasvaa jatkuvasti, sillä ihmiset tuottavat ja lisäävät uutta tietoa, mutta harvoin poistavat sitä. Kontekstin pituudet kasvavat nopeasti, mutta eivät tarpeeksi nopeasti todella ison syötekontekstin tarvitsevien sovellusten tarpeisiin. (Huyen 2024, luku 6)

Lisäksi kielimallit eivät osaa hyödyntää pitkää syötekontekstia täydellisesti. Liu ym. (2023) analysoivat kielimallien suorituskykyä tehtävissä, jotka vaativat olennaisen tiedon tunnistamista syötekontekstista. Havaittiin, että suorituskyky voi heikentyä merkittävästi, kun relevantin tiedon sijaintia muutetaan. Johtopäätöksenä nykyiset kielimallit eivät hyödynnä pitkien syötekontekstien tietoa riittävän hyvin. Suorituskyky on usein parhaimmillaan, kun olennainen tieto esiintyy syötteen alussa tai lopussa. Suorituskyky heikkenee huomattavasti, kun mallin täytyy hakea oleellinen tieto pitkän kontekstin keskeltä, vaikka kielimalli olisi suunniteltu pitkille syötekonteksteille. (Liu ym. 2023, 1)

2.2 Mikropalveluarkkitehtuuri

Mikropalveluarkkitehtuurissa (eng. microservices architecture) ohjelmisto koostuu pienistä itsenäisistä rakennuspalikoista eli mikropalveluista, jotka kommunikoivat keskenään vain viestinvälityksen kautta. Jokaisella mikropalvelulla on oma selkeä tehtävänsä, jolloin monimutkaisuus siirtyy palveluiden yhteensovittamiseen eli orkestrointiin. Jokaisen mikropalvelun odotetaan toteuttavan hyvin rajatun toiminnallisuuden, mikä auttaa palvelujen ylläpidossa ja laajennettavuudessa. Koska jokainen mikropalvelu on riippumaton ja sitä päivitetään omana kokonaisuutenaan, virheiden korjaukset tai pienet parannukset eivät vaikuta muihin palveluihin tai niiden julkaisuihin. (Bucchiarone ym. 2018, 2)

Monoliittisissa (eng. monolith) eli perinteisissä arkkitehtuureissa modulaarisuus perustuu saman koneen resurssien jakamiseen, eivätkä ohjelmiston osat ole itsenäisesti suoritettavia. Huomattavia haasteita monoliittisessa arkkitehtuurissa

on ylläpidettävyys ja kehitettävyys sekä muutosten hallinta. (Bucchiarone ym. 2018, 2)

Suurimpia hyötyjä mikropalveluarkkitehtuurissa on palveluiden riippumattomuus toisistaan, kehittämisen joustavuus sekä skaalattavuus. Palveluita voi kehittää itsenäisesti sekä päivittää ilman, että se vaikuttaa muihin palveluihin. Kehittäminen on joustavampaa, sillä eri palvelut voidaan kehittää eri teknologioilla. Jatkuva integraatio ja jatkuva käyttöönotto (CI/CD) voidaan toteuttaa palvelukohtaisesti, sillä jokainen palvelu voidaan paketoida ja siirtää omana konttinaan. Jokainen palvelu voidaan skaalata erikseen, jolloin eri palveluihin kohdistuva kuorma voidaan ottaa huomioon tuotannossa. (Soldani ym. 2018, 16.)

3 KÄYTTÖOHJEAVUSTIMEN TOTEUTUS

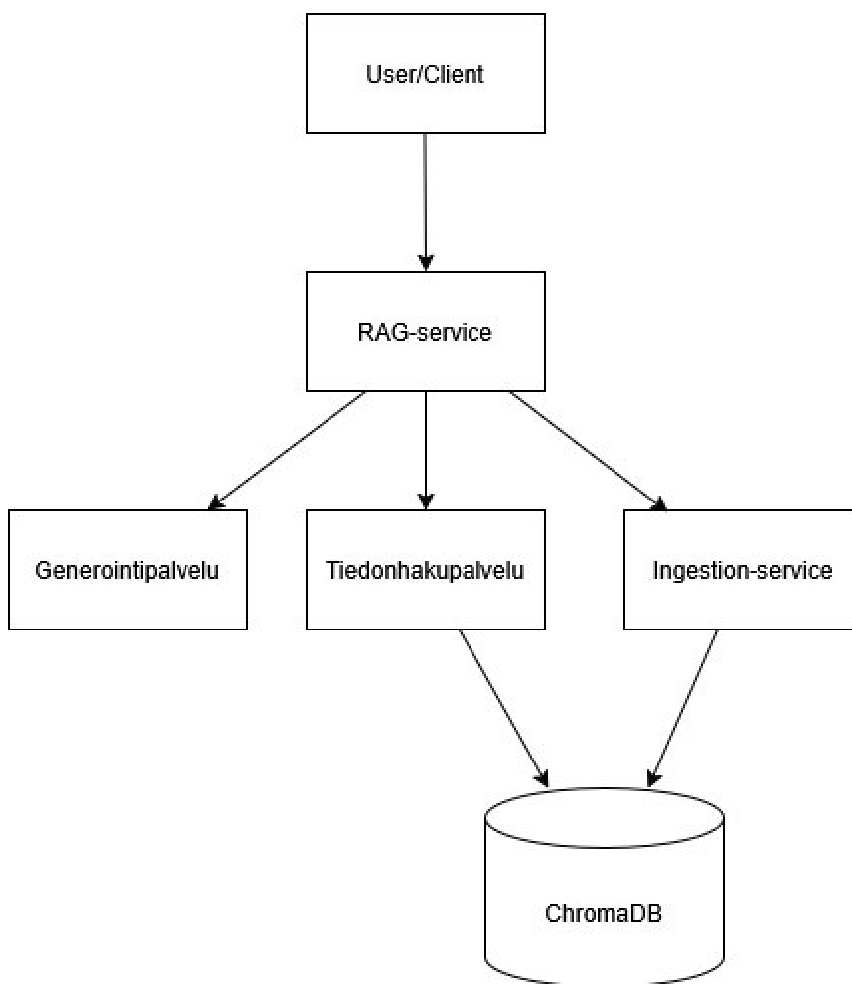
Tässä projektissa toteutettu sovellus on chatbot, joka vastaa käyttöohjeeseen liittyviin kysymyksiin. Sovellus hyödyntää RAG-tekoälyä vastausten antamisessa, jolloin vastausten pitäisi olla parempia kuin pelkän perinteisen suuren kielimallin antamat vastaukset. Sovellus on mikropalveluarkkitehtuurilla toteutettu taustajärjestelmä, joka tarjoaa API:n chatbotin käyttöön.

API:n kautta sovellukselle voi antaa käyttöohjeen pdf-dokumenttina, jonka data luetaan vektoritietokantaan. Käyttäjä voi esittää kysymyksiä käyttöohjeesta, jolloin suuri kielimalli käyttää käyttöohjeesta luettua dataa vastausten antamisessa.

Sovellus tehtiin Visual Studio Code -editorilla ja versionhallinnassa käytettiin Git-ohjelmaa. Pythonin kirjastojen hallinnassa sekä projektin kehityksessä käytettiin uv-työkalua (UV n.d.). Koodin siistimisessä ja muotoilussa käytettiin Ruff-työkalua (Ruff n.d.).

3.1 Mikropalveluarkkitehtuuri

Sovellus koostuu neljästä mikropalvelusta: RAG-palvelu, generation-palvelu, retrieval-palvelu, ingestion-palvelu. Mikropalvelut kommunikoivat keskenään sekä Chroma-vektoritietokannan kanssa API:iden avulla. Käyttäjä kommunikoi sovelluksen kanssa RAG-palvelun kautta. Tieodonhakupalvelu ja Ingestion-palvelu jakavat saman tietokannan monimutkaisuuden vähentämiseksi. Kuviossa 2 on sovelluksen mikropalveluarkkitehtuuri. Sovellus tarjoaa mahdollisuuden tallentaa käyttöohjeen ja kysyä kysymyksiä käyttöohjeesta. Kun käyttäjä tallentaa käyttöohjeen, PDF-tiedosto annetaan RAG-palvelulle, joka välittää käyttäjän antaman käyttöohjetiedoston Ingestion-palvelulle. Ingestion-palvelu lukee tiedoston ja tallentaa datan Chroma-vektoritietokantaan.



Kuvio 2: Sovelluksen mikropalveluarkkitehtuuri.

Käyttäjän tehdessä kysymystä käyttöohjeesta RAG-palvelu välittää kysymyksen tiedonhakupalvelulle, joka hakee kysymykseen soveltuvan datan Chroma-tietokannasta. Tiedonhakupalvelu välittää datan takaisin RAG-palvelulle. Kun RAG-palvelu on saanut kysymystä varten oleellisen datan tiedonhakupalvelulta, välittää RAG-palvelu kysymyksen ja datan generointipalvelulle. Generointipalvelu käyttää suuria kielimalleja OpenAI:n API:n kautta sekä käyttöohjedokumentista saatua dataa vastauksen generointiin. Vastaus palautetaan RAG-palvelun kautta käyttäjälle.

Kaikissa palveluissa on samankaltainen FastAPI-arkkitehtuuri (FastAPI n.d.). App-kansiossa on sovelluksen varsinainen koodi ja tests-kansiossa on testit sovellukselle sisältäen kansiot yksikkö ja integraatiotesteille. App-kansiossa on routers-kansio reittien luontia varten sekä services-kansio, joka sisältää tiedostot palvelulogiikkaa varten. Esimerkiksi RAG-palvelussa palvelulogiikka hoitaa RAG-

prosessin eli hakuvaiheen, generointivaiheen ja vastauksen palauttamisen. Models-tiedosto sisältää Pydantic-mallit datan validointia varten (Pydantic n.d.). Deps-kansio sisältää riippuvuuksien injektoinnin, joka on oleellinen osa FastAPI-arkkitehtuuria.

Kaikissa palvelussa käytetään API:n tekemiseen FastAPI-kirjastoa ja validointiin Pydantic-kirjastoa. Uvicornin avulla luodaan web-palvelin, jolla palvelun API:a ajetaan. Taulukossa 1 on kirjastot, joita käytetään kaikissa palveluissa.

Taulukko 1: Kaikissa palveluissa käytetyt kirjastot

Kirjasto	Selite
fastapi	web-framework API:den tekemiseen
pydantic	datan validiointikirjasto
pydantic-settings	asetustenhallinta ja konfigurointi Pydantic-kirjastoon
uvicorn	web-palvelin -toteutus Pythonille

3.2 RAG-palvelu

RAG-palvelu on käyttäjän rajapinta sovelluksen käyttämiseen. Käyttäjä voi lisätä käyttöohjeen sovellukseen, hakea tallennettujen käyttöohjeiden nimet sekä poistaa kaikki käyttöohjeet sovelluksesta. RAG-palvelulta voi kysyä ingestion-palvelun tilan. RAG-palvelu ei vastaa kysymyksiin, jos ingestion-palvelu prosessoi ohjetta. Tilan avulla esimerkiksi mahdolliset käyttöohjesovellukseen liitettävät frontend-sovellukset voivat näyttää käyttöohjeen latauksen ja estää käyttäjää esittämästä kysymyksiä, jos ingestion-sovelluksella on ohjeen prosessointi kesken. Health-reitin avulla voi tarkistaa onko sovellus toiminnassa. Kuvassa 1 on RAG-palvelun kaikki tarjolla olevat reitit.

RAG Service 1.0.0 OAS 3.1

/openapi.json

Orchestrates RAG pipeline for the user manual assistant chatbot.

chat

POST /api/v1/chat Process a user's chat message

documents

POST /api/v1/documents/upload Upload a PDF document for ingestion via Ingestion Service

GET /api/v1/documents List documents managed by the Ingestion Service

DELETE /api/v1/documents Clear all documents and ingested data via Ingestion Service

ingestion

GET /api/v1/ingestion/status Get ingestion status from Ingestion Service

health

GET /health Health Check

Kuva 1: RAG-palvelun reitit

RAG-palvelussa toteutus perustuu aiemmin kerrottuun palveluiden toteutukseen. Palvelulogiikka on hoidettu kahden palvelun avulla: chat_processor ja http_client. Chat-prosessori hoitaa RAG-logiikan eli kysyy kysymykseen liittyvää dataa tiedonhakupalvelulta ja generoi vastauksen generointipalvelun avulla. HTTP-asiakkaan avulla tehdään pyyntöjä toisille mikropalveluille. Liitteessä 1 on RAG-palvelun tiedostorakenne. Yhteisten kirjastojen lisäksi RAG-palvelussa käytettiin HTTPX-kirjastoa pyyntöjen tekemiseen tiedonhaku ja ingestion -palveluilta. Taulukossa 2 on kaikki RAG-palvelussa käytetyt kirjastot.

Taulukko 2: RAG-palvelussa käytetyt kirjastot

Kirjasto	Selite
fastapi	web-framework API:den tekemiseen
httpx	HTTPX-pyyntöjen tekemiseen
pydantic	datan validointi
pydantic-settings	asetustenhallinta ja konfigurointi Pydantic-kirjastoon
uvicorn	web-palvelin -toteutus Pythonille

3.3 Generointipalvelu

Generointipalvelu tuottaa vastauksen käyttäjän kysymykseen käyttöohjeesta saadun datan avulla ja vastaus palautetaan käyttäjälle RAG-palvelun kautta. Generointipalvelu käyttää suuria kielimalleja OpenAI:n API:n kautta ja mallin voi valita env-tiedoston parametrin avulla. Kuvassa 2 näkyvät generointipalvelun reitit. Generate-reitillä tuotetaan vastaus ja health-reitillä voi tarkistaa, onko palvelu toiminnassa.

Kuva 2: Generointipalvelun reitit

Palvelussa on toteutettu RAG-putki Langchain-kirjaston avulla. Putkessa on prompti-malli, jossa on järjestelmäohje kielimallille, konteksti käyttöohjedatasta ja käyttäjän kysymys sekä lopuksi vastaus. Kuvassa 3 näkyy prompti-malli, joka antaa ohjeet suurelle kielimallille.

```
You are a helpful technical documentation assistant specializing in user
manuals and product guides. Your primary goal is to help users understand how to use
products and troubleshoot issues based on the provided documentation.

Instructions:
- Answer questions using ONLY the information provided in the context below
- Provide clear, step-by-step instructions when explaining procedures
- Include relevant warnings, cautions, or safety notes mentioned in the documentation
- If the context contains multiple relevant sections, synthesize the information coherently
- When referencing specific features, buttons, or settings, use the exact terminology from the manual
- If the context doesn't contain sufficient information to answer the question, clearly state this limitation
- For troubleshooting questions, provide systematic diagnostic steps if available in the context
- Always prioritize user safety and proper usage guidelines

CONTEXT FROM USER MANUAL:
{context}

USER QUESTION:
{query}

ASSISTANT RESPONSE:
Based on the user manual information provided:

"""
```

Kuva 3: Prompti-malli

Kielimalli ohjeistetaan olemaan käyttöohjeavustaja, jonka tavoitteena on auttaa käyttäjää ymmärtämään, kuinka ohjeistuksen kohteena olevaa tuotetta käytetään. Kielimalli saa käyttää pelkästään sille annettua tietoa eikä se saa vastata, ellei tietoa ole riittävästi. Lisäksi on muuta ohjeistusta vastauksen muotoiluun. Käyttöohjeesta haettu data ja käyttäjän kysymys liitetään promptiin.

Seuraavaksi putkessa on kielimalli, jolle prompti annetaan ja viimeisenä Langchainin outputparser, joka muuntaa kielimallin tuottaman vastauksen merkkijonoksi sekä validoi ja puhdistaa vastauksen. Putkesta tulee ulos valmis vastaus, joka palautetaan RAG-palvelulle.

Generointipalvelunkin toteutus perustuu aiemmin kerrottuun mikropalveluiden yleiseen toteutukseen. Generointipalvelussa on yksi palvelumoduuli, joka hoitaa vastauksen generoinnin. Liitteessä 2 on generointipalvelun tiedostorakenne. Lisäksi generointipalvelussa käytettiin langchain-kirjastoa vastauksen tuottamiseen OpenAI GPT-kielimallien avulla. Taulukossa 3 on kaikki generointipalvelussa käytetyt kirjastot.

Taulukko 3: Generointipalvelussa käytetyt kirjastot

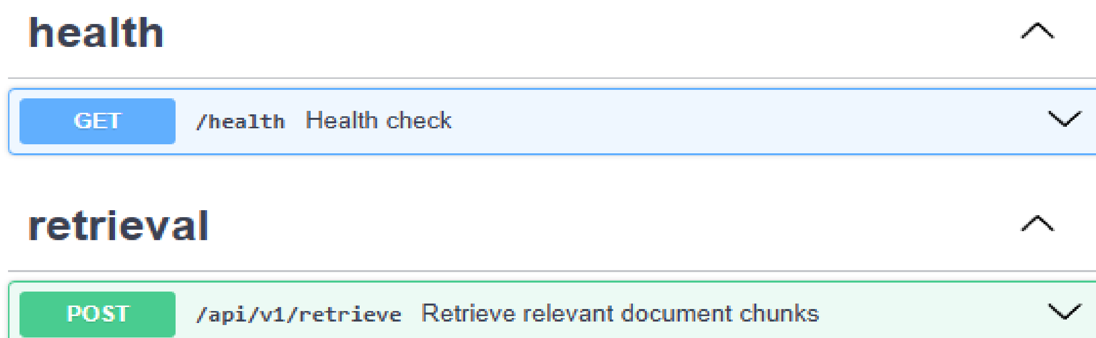
Kirjasto	Selite
fastapi	web-framework API:den tekemiseen
langchain	kielimallien käyttämiseen
langchain-openai	langchain integraation OpenAI- API:lle
pydantic	datan validiointikirjasto
pydantic-settings	asetustenhallinta ja konfigurointi Pydantic-kirjastoon
uvicorn	web-palvelin-toteutus Pythonille

3.4 Tiedonhakupalvelu

Tiedonhakupalvelu muuntaa käyttäjän syöttämän tekstin numeerisiksi vektoreiksi. Numeeristen vektoreiden avulla Chroma-tietokannasta haetaan vastaavia numeerisia vektoreita, joiden perusteella palvelu palauttaa syötettyyn tekstiin liittyviä tekstilohkoja eli tässä tapauksessa käyttöohjeessa olevia tekstipätkiä. Tekstilohkot ovat lisädataa generointipalvelulle kysymykseen vastatessa.

Tiedonhakupalvelu käyttää numeeristen vektoreiden luontiin sentence-transformer-kirjastoa, joka käyttää HuggingFacen all-MiniLM-L6-v2 upotusmallia (Hugging Face n.d.). Myös ingestion-palvelu käyttää samaa upotusmallia, jotta tiedonhakupalvelulla on mahdollisuus löytää relevanttia dataa vektoritietokannasta semanttisen etsinnän avulla.

Tiedonhakupalvelussa on kaksi reittiä: retrieve ja health. Retrieve-reitille annetaan käyttäjän kysymys ja se palauttaa kysymykseen liittyvää dataa. Ainoastaan riittävän hyvää dataa palautetaan, joten on mahdollista, että ei palauteta mitään. Kuvassa 4 on tiedonhakupalvelun tarjoamat reitit.



Kuva 4: Tiedonhakupalvelun reitit

Myös tiedonhakupalvelun toteutus perustuu aiemmin kerrottuun palveluiden yleiseen toteutukseen. Mikropalvelun palvelulogiikka koostuu neljästä erillisestä palvelumoduulista. VectorSearch-palvelu on vastuussa tiedonhausta tietokannasta ja käyttää muita palveluita prosessissa. Liitteessä 3 on tiedonhakupalvelun tiedostorakenne.

Lisäksi tiedonhakupalvelussa käytettiin sentence-transformers-kirjastoa upotuksien tekemiseen kysymyksistä ja chromadb-kirjastoa Chroma-vektoritietokannan yhdistämiseen. Taulukossa 4 on kaikki tiedonhakupalvelussa käytetyt kirjastot.

Taulukko 4: Tiedonhakupalvelussa käytetyt kirjastot

Kirjasto	Selite
chromadb	vektoritietokannan käyttöön
fastapi	web-framework API:den tekemiseen
pydantic	datan validiointi kirjasto
pydantic-settings	asetustenhallinta ja konfigurointi Pydantic-kirjastoon
sentence-transformers	upotusmallin käyttöön
uvicorn	web-palvelin-toteutus Pythonille

3.5 Ingestion-palvelu

Ingestion-palvelu on vastuussa käyttöohjetiedoston prosessoinnista. Palvelu muuntaa PDF-dokumentin tekstilohkoiksi, jotka koodataan numeerisesti. Tekstilohkot ja niiden numeeriset koodaukset talletetaan Chroma-vektoritietokantaan. Palvelulle voidaan antaa jo alustusvaiheessa käyttöohjetiedostoja, jolloin tiedostot voidaan lukea tietokantaan erillisellä pyynnöllä. Tiedosto voidaan lukea tietokantaan vain kerran, mikä estetään tiedoston nimen perusteella. Palvelulta voidaan kysyä myös tallennettujen käyttöohjeiden nimet, mitä voidaan hyödyntää esimerkiksi palveluun yhdistettävissä frontend-sovelluksissa.

Kuva 5 esittää ingestion-palvelun tarjoamat reitit. Upload-reitin avulla voidaan ladata uusi käyttöohje sovellukseen, jolloin käyttöohje luetaan samalla automaattisesti tietokantaan. Reitti tarkistaa, että tiedostoa ei ole jo ladattuna sovellukseen. Ingest-reitillä voidaan lukea alustusvaiheessa ladatut tiedostot tietokantaan. Tiedosto voidaan lukea tietokantaan vain kerran. Status-reitti palauttaa ingestion-palvelun tilan, joka voi olla joko valmis tai prosessoidaan. Tilan perusteella RAG-palvelun käyttäjä osaa odottaa prosessin valmistumista.

Documents-reitin avulla voidaan hakea sovellukseen tallennettujen käyttöohjeiden nimet, joita voidaan näyttää esimerkiksi sovellukseen yhdistetyssä frontendissä. Collection-reitin kautta poistetaan ladatut käyttöohjeet sekä tietokantaan tallennettu data. Reitti poistaa kaikki käyttöohjeet ja kaiken datan. Health-reitillä voi tarkistaa, onko palvelu toiminnassa.

Ingestion Service 1.0.0 OAS 3.1
/openapi.json

Loads, processes, and stores documents in a vector database.

ingestion

POST /api/v1/upload Upload PDF file for ingestion

POST /api/v1/ingest Trigger document ingestion process

GET /api/v1/status Get ingestion status

documents_management

GET /api/v1/documents/ List PDF documents in the source directory

collection_management

DELETE /api/v1/collection/ Clear ChromaDB Collection and Source Documents

health

GET /health Health check

Kuva 5: Ingestion-palvelun reitit

Ingestion-palvelu käyttää numeeristen vektoreiden luontiin samaa all-MiniLM-L6-v2 upotusmallia kuin tiedonhakupalvelu. Jotta tiedonhakupalvelu voi löytää kysymykseen liittyvää relevanttia dataa, täytyy käyttöohjedata koodata samalla upotusmallilla.

Mikropalvelun toteutus perustuu samaan mikropalveluiden yleiseen toteutukseen. Palvelun palvelulogiikka koostuu seitsemästä erillisestä palvelumoduulista, mikä tekee ingestion-palvelusta monimutkaisimman mikropalvelun tässä sovel-

luksessa. Palvelun tilaa hallitaan IngestionState-palvelulla. Palvelun tila on oleellista tietää, koska käyttöohjeet prosessoidaan taustalla ja mahdolliset sovellukseen kytketyt toiset sovellukset voivat haluta tietää, onko prosessointi valmis.

IngestionProcessor-palvelu hoitaa tiedoston lukemisen ja datan tallentamisen tietokantaan, joka käyttää muita palveluita apunansa prosessissa. Käyttöohjetiedostojen tallentamisesta ja poistamisesta on vastuussa FileManagement-palvelu. Liitteessä 4 on ingestion-palvelun tiedostorakenne.

Yhteisten kirjastojen lisäksi ingestion-palvelussa käytettiin chromadb-kirjastoa Chroma-vektoritietokannan yhdistämiseen. Upotuksien tekemiseen käytettiin sentence-transformers ja Langhain-kirjastoja. Taulukossa 5 on kaikki tiedonhakupalvelussa käytetyt kirjastot.

Taulukko 5: Ingestion-palvelussa käytetyt kirjastot

Kirjasto	Selite
chromadb	vektoritietokannan käyttöön
fastapi	web-framework API:den tekemiseen
langchain	kielimallien käyttämiseen
langchain-chroma	LangChain integraation Chromalle
langchain-community	kolmannen osapuolen integraatioita LangChainiin
pydantic	datan validiointi kirjasto
pydantic-settings	asetustenhallinta ja konfigurointi Pydantic-kirjastoon
sentence-transformers	upotusmallin käyttöön
uvicorn	web server -toteutus Pythonille

3.6 Chroma-vektoritietokanta

Chroma on avoimen lähdekoodin tekoälysovelluksille tarkoitettu tietokanta (Chroma n.d.). Se on vektoritietokanta, johon talletetaan tekstistä upotukset numeerisina vektoreina ja teksti puhtaana tekstinä. Tietokanta ylläpitää sidosta tekstin ja siitä tehdyn vektorin välillä. Dataa tietokannasta etsitään vektoreiden ja semanttisten hakujen avulla. Tietokannasta yritetään etsiä samankaltaisia vektoreita ja siihen liitettyä tekstiä.

Tässä sovelluksessa ingestion-palvelu käyttää Chroma-tietokantaa käyttöohjedatan tallentamiseen ja tiedonhakupalvelu kysymykseen liittyvän datan hakemiseen. Chroma on yleisesti käytetty tietokanta Langchain-kirjaston kanssa ja sopii siten hyvin sovelluksen käyttötarkoitukseen. Tietokantaa ajetaan erillisessä Docker-kontissa.

3.7 Testaaminen

Sovelluksen toimivuutta ja laatua on testattu useilla eri menetelmillä. Automatisoidut yksikkötestit ja integrointitestit ovat testauksen perusta. Yksikkötestit testaavat sovelluksen yksittäistä osaa ilman riippuvuuksia ja integrointitesteissä testataan sovelluksen useamman osan yhteen toimivuutta. Lisäksi Vs Code:n REST Clientin avulla on tehty manuaalisia integrointitestejä sekä sovellukseen yhdistetyn frontendin avulla tehty manuaalisia päästä päähän testejä.

Jokaiselle mikropalvelulle on tehty hakemistorakenteeseen erilliset kansiot automaattisille testeille, joissa on omat kansionsa yksikkötesteille ja integraatiotesteille. Testien määrät vaihtelevat palveluittain, mutta jokaiselle palvelulle on tehty vähintään useita kymmeniä yksikkö- ja integrointitestejä. Testaamiseen on käytetty pytest-kirjastoa ja sen lisäosia sekä NumPy-kirjastoa. Pytestin avulla saadaan tehtyä kattavia testejä sovelluksesta ja NumPy:ta on käytetty upotusarvojen jäljittelyyn, kun tekstilohkoista on tehty vektoreita vektoritietokantaa varten. Taulukossa 6 on automaatiotesteissä käytetyt Python-kirjastot.

Taulukko 6: Testauksessa käytetyt kirjastot

Kirjasto	Selite
pytest	testauskirjasto
pytest-mock	pytest-lisäosa sovelluksen osien jäljittelyyn
pytest-asyncio	pytest-lisäosa asynkronisen koodin testaukseen
pytest-cov	pytest-lisäosa koodikattavuusraportin tekemiseen
numpy	kirjasto numeeriseen laskentaan.

Manuaalista integrointitestausta on tehty Visual Studio Coden REST Clientin avulla. Sen avulla voidaan tehdä HTTP-kyselyitä, joiden vastaukset tulevat näkyviin Visual Studio Codeen. Päästä päähän testejä on tehty erillisen frontendin avulla. Frontendissä käyttäjä pääsee lataamaan ja poistamaan käyttöohjeen sekä kysymään kysymyksiä käyttöohjeesta. Manuaalisissa testeissä ei ole havaittu suurempia ongelmia sovelluksen toiminnassa.

3.8 Käyttöönotto

Sovellus koostuu itsenäisistä mikropalveluista, jotka on tarkoitettu ajettavan erikseen Docker-konteissa. Jokaisessa palvelussa on Dockerfile-tiedosto, jonka avulla voidaan rakentaa Docker-kuva. Jokaisessa palvelussa on myös requirements.txt-tiedosto, jossa on listattuna tarvittavat kirjastot ja niiden versiot palvelun ajamiseen.

Vaikka arkkitehtuuri mahdollistaisi palveluiden ajamisen eri palvelimilla, tässä projektissa kaikki mikropalvelut ajetaan yhdellä koneella käyttäen Docker Compose -työkalua. Sen avulla kaikki kontit voidaan rakentaa ja käynnistää yhdellä komennolla. Sovelluksessa on valmiina compose.yml-tiedosto, joka osaa rakentaa kaikki palvelut sekä luo datavolyymit vektoritietokannan datalle ja ingestion-palvelun tallentamille käyttöohjeille.

4 POHDINTA

Tämän työn tarkoituksena oli toteuttaa käyttöohjeavustin käyttämällä RAG-teknologiaa ja mikropalveluarkkitehtuuria. Työssä onnistuttiin pilkkomaan sovellus pienempiin itsenäisiin osiin, jolloin RAG-teknologian eri osille on omat palvelunsa. Sovellus pystyy vastaamaan käyttöohjeeseen liittyviin kysymyksiin yksityiskohtaisemmin ja tarkemmin kuin perinteinen suuri kielimalli, ja sen tuottamat vastaukset ovat ajantasaisempia. Sovellus lähtökohtaisesti kieltäytyy vastaamasta kysymyksiin, ellei kielimalli saa mielestään riittävästi dataa käyttöohjeesta kysymykseen vastaamiseen ja siten hallusinoinnin riski on huomattavasti pienempi kuin pelkällä suurella kielimallilla.

Mikropalveluarkkitehtuurin avulla sovelluksen kehittäminen oli joustavampaa ja helpompaa. Nyt sovelluksen jokaista osaa voi ylläpitää ja jatkokehittää erikseen. Ohjelman tuotantokäytössä arkkitehtuuri mahdollistaa helpomman skaalautuvuuden, sillä jokaista palvelua on mahdollista skaalata erikseen. Esimerkiksi ingestion-palvelulle voidaan allokoida enemmän resursseja kuin tiedonhakupalvelulle, jos tarve vaatii.

Sovellus toteutettiin pelkästään Pythonilla, mutta jatkossa sovellukseen on mahdollista tehdä uusia mikropalveluita myös muilla ohjelmointikielillä tai olemassa olevan mikropalvelun voi korvata toisella ohjelmointikielellä tehdyllä mikropalvelulla. Uusilla mikropalveluilla voidaan lisätä uusia ominaisuuksia ilman, että vanhoja mikropalveluita tarvitsee välttämättä muokata.

Sovellusta on mahdollista jatkokehittää usealla tavalla. Tällä hetkellä sovellus käyttää ainoastaan OpenAI:n kielimalleja API:n kautta, mutta sovellusta voisi kehittää käyttämään myös muiden yritysten kielimalleja. Sovellukseen voisi toteuttaa autentikoinnin ja sen avulla käyttäjäkokemuksen personoinnin. Lisäksi sovellukseen voisi tehdä uuden palvelun käyttäjädatan keräämiseen ja hyödyntämiseen.

Sovelluksessa voi tällä hetkellä valita OpenAI-malleista env-parametrin avulla. Jatkokehityksessä sovelluksen voisi tehdä mahdollisuuden valita myös muiden

yritysten tarjoamista kielimalleista. Sovellus voisi käyttää kielimalleja yritysten tarjoamien API:iden kautta tai mahdollisuuksien mukaan paikallisesti ajettavina malleina. Paikallisesti toimivat mallit voisivat olla tarpeellisia, jos käyttöohjeiden tietoja ei haluta missään tapauksessa päästää sovelluksen ulkopuolelle.

Autentikoinnin lisäämisellä mahdollistuisi personoidun käyttäjäkokemuksen luominen. Käyttäjäkokemusta voisi personoida mahdollistamalla omien käyttöohjeiden tallentamisella sekä keskusteluiden tallentamisella. Käyttäjälle voisi lisätä mahdollisuuden antaa palautetta vastauksista esimerkiksi ylä- ja alapeukaloiden avulla ja tätä palautetta voisi käyttää sovelluksen kehittämiseen. Käyttäjädatan keräämiseen voisi tehdä uuden mikropalvelun avulla.

Sovellus ei tällä hetkellä tallenna käyttäjän käymiä keskusteluita, mikä vähentää merkittävästi tietosuojahaasteita. Jos keskustelut tallennetaan jatkokehityksen tuloksena, täytyy ottaa huomioon tietosuojan lisäksi muut eettiset kysymykset. Tietosuojan varmistamiseksi täytyy noudattaa voimassa olevia lakeja ja säädöksiä kuten GDPR, jolloin käyttäjän suostumus täytyy hankkia. Jos käyttäjän dataa tallennettaisiin, käyttäjälle täytyisi tarjota selkeä tieto, miten heidän tietojaan käsitellään sekä mahdollisuus hallita omaa dataansa.

Sovellus suostuu vastaamaan vain käyttöohjeeseen liittyviin kysymyksiin, joten eettisesti kyseenalaisempiin kysymyksiin on vaikeampi saada vastauksia. Muiden suodatusmekanismien lisääminen voi olla tarpeen jatkokehityksessä, jotta haitallisia vastauksia saadaan ehkäistyä.

Käyttäjän kysymysdata lähetetään OpenAI:lle API:n kautta, jolloin noudatetaan OpenAI:n tietosuojakäytänteitä. Jos jatkokehityksessä lisätään muita kielimallivaihtoehtoja, niiden tietosuojakäytänteet täytyy ottaa huomioon.

LÄHTEET

Bucchiarone, A., Dragoni, N., Dustdar, S., Larsen, S. T., Mazzara, M. 2018. From Monolithic to Microservices: An Experience Report from the Banking Domain. IEEE Software, 35(3), 50-55. Viitattu 27.5.2025. https://doi.org/10.13140/RG.2.2.34717.00482

Chang, Y., Wang, X., Wang, J., Wu, Y., Yang, L., Zhu, K., Chen., Yi, X., Wang, C., Wang, Y., Ye, W., Zhang, Y., Chang, Y., Yu, P., Yang, Q., Xie, X. 2024. A Survey on Evaluation of Large Language Models. ACM Trans. Intell. Syst. Technol. 15 (3), 1-45. Viitattu 30.5.2025. https://doi.org/10.1145/3641289

Chen, D., Fisch, A., Weston, J., Bordes, A,. 2017. Reading Wikipedia to Answer Open-Domain Questions. arXiv. Viitattu 30.5.2025. https://doi.org/10.48550/arXiv.1704.00051

Chroma. n.d. Introduction. Verkkosivu. Viitattu 19.5.2025. https://docs.trychroma.com/docs/overview/introduction

FastAPI. n.d. Dependencies. Verkkosivu. Viitattu 27.5.2025 https://fastapi.tiangolo.com/tutorial/dependencies/

Hugging Face. n.d. all-MiniLM-L6-v2. verkkosivu. Viitattu 27.5.2025 https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

Huyen, C. 2024. AI Engineering. 1.painos. E-Kirja. Sebastopol: O'Reilly Media. Viitattu 13.5.2025. Vaatii käyttöoikeuden. https://learning.oreilly.com/library/view/ai-engineering/9781098166298

Hämäläinen, M. 2024. Tekoäly ja uusimmat suuret kielimallit. E-Kirja. Metropolia Ammattikorkeakoulu. Viitattu 23.5.2025. https://www.researchgate.net/publication/385852294_Tekoaly_ja_uusimmat_suuret_kielimallit

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W-t., Rocktäschel, T., Riedel, S., Kiela, D. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. arXiv. https://doi.org/10.48550/arXiv.2005.11401

Liu, N. F., Lin, K,. Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, Liang, P. 2023. Lost in the Middle: How Language Models Use Long Contexts. arXiv. https://doi.org/10.48550/arXiv.2307.03172

Pydantic. n.d. Welcome to Pydantic. Verkkosivu. Viitattu 5.6.2025 https://docs.pydantic.dev/latest/

Ruff. n.d. Overview. Verkkosivu. Viitattu 1.6.2025. https://docs.astral.sh/ruff/

Soldani, J., Tamburri, D., Van Den Heuvel, W. 2018. The pains and gains of microservices: Systematic grey literaure review. Journal of Systems and Software, Volume 146, 215-232. Viitattu 27.5.2025
http://dx.doi.org/10.1016/j.jss.2018.09.082

UV. n.d. Introduction. Verkkosivu. Viitattu 1.6. https://docs.astral.sh/uv/

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., Polosukhin, I., 2017. Attention Is All You Need. arXiv.
Viitattu 28.05.2025.
https://doi.org/10.48550/arXiv.1706.03762

Zheng, A., & Casari, A. 2018. Feature Engineering for Machine Learning: Principles and Techniques. Sebastopol: O'Reilly Media. Viitattu 1.6.2025.
Vaatii käyttöoikeuden.
https://learning.oreilly.com/library/view/feature-engineering-for/9781491953235/

LIITTEET

Liite 1. RAG-palvelun tiedostorakenne

```
rag-service
|   .coverage
|   .env
|   Dockerfile
|   project_structure.txt
|   pyproject.toml
|   pytest.ini
|   requirements.txt
|
+---app
|   |   config.py
|   |   deps.py
|   |   main.py
|   |   models.py
|   |   __init__.py
|   |
|   +---routers
|   |   |   chat.py
|   |   |   documents.py
|   |   |   health.py
|   |   |   ingestion.py
|   |   |   __init__.py
|   |
|   +---services
|   |   |   chat_processor.py
|   |   |   http_client.py
|
+---otel-collector-config.yaml
\---tests
    |   __init__.py
    |
    +---integration
    |       conftest.py
    |       test_api.py
    |       test_chat_integration.py
    |       test_documents_integration.py
    |
    +---unit
    |   |   conftest.py
    |   |   test_deps.py
    |   |   test_models.py
    |   |   __init__.py
    |   |
    |   +---routers
    |   |   |   test_chat_router.py
    |   |   |   test_documents_router.py
    |   |   |   test_health_router.py
    |   |   |   test_ingestion_router.py
    |   |
    |   +---services
    |   |   |   test_chat_processor.py
    |   |   |   test_http_client.py
```

Liite 2. Generointipalvelun tiedostorakenne

```
generation-service
|   .env
|   Dockerfile
|   pyproject.toml
|   README.md
|   requirements.txt
|
+---app
|   |   config.py
|   |   deps.py
|   |   main.py
|   |   models.py
|   |   routers.py
|   |   __init__.py
|   |
|   +---routers
|   |   |   generation.py
|   |   |   health.py
|   |   |   __init__.py
|   |
|   +---services
|   |   |   generation.py
|   |   |   __init__.py
|
+---tests
|   |   conftest.py
|   |   __init__.py
|   |
|   +---integration
|   |   |   conftest.py
|   |   |   test_api.py
|   |   |   __init__.py
|   |   |
|   |   +---api
|   |   |       test_endpoints.py
|   |   |       __init__.py
|   |   |
|   |   +---services
|   |   |       conftest.py
|   |   |       test_generation_service.py
|   |   |       __init__.py
|
|   +---unit
|   |   |   conftest.py
|   |   |   test_models.py
|   |   |
|   |   +---services
|   |   |       test_generation_service.py
|   |   |       __init__.py
```

Liite 3. Tiedonhakupalvelun tiedostorakenne

```
retrieval-service
|   .env
|   Dockerfile
|   load_dummy_data.py
|   project_structure.txt
|   pyproject.toml
|   README.md
|   requirements.txt
|
+---app
|   |   config.py
|   |   deps.py
|   |   main.py
|   |   models.py
|   |   routers.py
|   |   __init__.py
|   |
|   +---routers
|   |   |   health.py
|   |   |   retrieval.py
|   |   |   __init__.py
|   |
|   +---services
|   |   |   chroma_manager.py
|   |   |   embedding_manager.py
|   |   |   vector_search.py
|   |   |   vector_store_manager.py
|
+---data
|   \---chroma_db
|           chroma.sqlite3
|
+---tests
|   |   conftest.py
|   |   __init__.py
|   |
|   +---integration
|   |   |   conftest.py
|   |   |   test_api.py
|   |   |   __init__.py
|   |   |
|   |   +---services
|   |   |       test_managers_integration.py
|   |   |       test_vector_search_service.py
|   |   |       __init__.py
|
|   +---unit
|   |   |   conftest.py
|   |   |   test_chroma_manager.py
|   |   |   test_embedding_manager.py
|   |   |   test_models.py
|   |   |   test_vector_search.py
|   |   |   test_vector_store_manager.py
|   |   |   __init__.py
```

Liite 4. Ingestion-palvelun tiedostorakenne

```
ingestion-service
|   .env
|   Dockerfile
|   pyproject.toml
|   pytest.ini
|   README.md
|   requirements.txt
|
+---app
|   |   config.py
|   |   deps.py
|   |   main.py
|   |   models.py
|   |   __init__.py
|   |
|   +---routers
|   |   |   collection.py
|   |   |   documents.py
|   |   |   ingestion.py
|   |   |   __init__.py
|   |
|   +---services
|   |   |   chroma_manager.py
|   |   |   collection_manager.py
|   |   |   file_management.py
|   |   |   ingestion_processor.py
|   |   |   ingestion_state.py
|
+---documents
+---tests
|   |   conftest.py
|   |   __init__.py
|   |
|   +---fixtures
|   |       data_builders.py
|   |       service_factories.py
|   |       __init__.py
|   |
|   +---integration
|   |   |   conftest.py
|   |   |   __init__.py
|   |   |
|   |   +---api
|   |   |       test_endpoints.py
|   |   |       __init__.py
|   |   |
|   |   +---services
|   |   |       conftest.py
|   |   |       test_service_integrations.py
|   |   |       test_state_management.py
|   |   |       __init__.py
|
|   +---unit
|   |   |   conftest.py
|   |   |   test_models.py
|   |   |   __init__.py
|   |   |
|   |   +---services
|   |   |       conftest.py
|   |   |       test_chroma_manager.py
|   |   |       test_collection_manager.py
|   |   |       test_file_management.py
|   |   |       test_ingestion_processor.py
|   |   |       test_ingestion_state.py
|   |   |       __init__.py
|
+---test_documents
|       iphone-16-info.pdf
|       iphone_user_guide.pdf
|       test_pdf.pdf
```