Tampereen ammattikorkeakoulu

Julkisen tarjousprosessin alkuvaiheen kehittäminen generatiivisella tekoälyllä

Sovellus kickoff-esitysten automaattiseen tuottamiseen

Pyry Ikonen

OPINNÄYTETYÖ
Maaliskuu 2026

Tietojenkäsittelyn tutkinto-ohjelma
Ohjelmistotuotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn tutkinto-ohjelma
Ohjelmistotuotanto

IKONEN, PYRY:
Julkisen tarjousprosessin alkuvaiheen kehittäminen generatiivisella tekoälyllä
Sovellus kickoff-esitysten automaattiseen tuottamiseen

Opinnäytetyö 45 sivua, joista liitteitä 1 sivu
Joulukuu 2025

Opinnäytetyön toimeksiantajana toimi Solita Oy. Työn tarkoituksena oli tehostaa yrityksen Public Business -tiimin tarjousprosessin alkuvaihetta ja vähentää asiantuntijoiden manuaalista työmäärää julkisten kilpailutusten käsittelyssä. Tavoitteena oli suunnitella ja toteuttaa generatiivista tekoälyä hyödyntävä sovellus, joka automatisoi laajoista tarjouspyyntöaineistoista koostettavan aloitusmateriaalin eli kickoff-esityksen luomisen.

Opinnäytetyö toteutettiin toiminnallisena kehittämistyönä. Ratkaisuksi rakennettiin selainpohjainen sovellus, joka hakee tarjousdokumentit Tarjouspalvelusta selainautomaation avulla, analysoi aineistot kielimallilla ja tuottaa niistä tiivistetyn PowerPoint-esityksen.

Työn tuloksena syntyi toimiva prototyyppi, joka kykenee muodostamaan laajasta aineistosta kickoff-esityksen automatisoidusti. Sovelluksen keskeinen ominaisuus on kielimallin ajatteluprosessin taltiointi, joka lisää kielimallin tuottaman sisällön tarkasteltavuutta ja tukee asiantuntijan tekemää laadunvarmistusta. Sovellusta ei ehditty ottaa tuotantokäyttöön, joten toiminnallisten hyötyjen toteutumista ei tässä työssä voitu mitata käytännön työympäristössä.

Jatkokehitysehdotuksina työssä esitettiin sovelluksen integroimista organisaation dokumenttienhallintajärjestelmään sekä toimittajariippuvuuden välttämistä käyttämällä erillistä kielimallien välityspalvelinta. Kehitetty ratkaisu soveltuu tarjousprosessin alkuvaiheen automatisoinnin tueksi, mutta sen käytännön toimivuus voidaan varmistaa vasta jatkokehityksen ja laajemman käyttöönoton jälkeen.

Asiasanat: generatiivinen tekoäly, suuret kielimallit, ohjelmistokehitys, julkiset kilpailutukset, dokumenttien analysointi

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Option of Software Development

IKONEN, PYRY:
Automation of the Initial Phase of Public Procurement with Generative AI
Application for Automating Kickoff Presentation Generation

Bachelor's thesis 45 pages, appendices 1 page
December 2025

The primary objective of the thesis was to streamline the initial phase of the public procurement sales process, which currently involves significant manual effort in analyzing and summarizing tender documents. The purpose was to design and implement a software solution utilizing Generative AI to automate the creation of kickoff presentations used for internal project evaluation. The commissioner of this thesis was Solita Oy.

The study was conducted as a functional thesis. The resulting solution is a web-based application that retrieves tender documents from Tarjouspalvelu using browser automation and generates summarized PowerPoint presentations.

The project produced a functioning prototype capable of processing tender documents and generating structured PowerPoint presentations within minutes. A key feature of the application is the recording of the language model's Chain of Thought process, which enhances transparency and helps in the verification of the data for the experts using the tool. The thesis demonstrates that Generative AI can increase efficiency in document-intensive sales processes, but its practical functionality can only be verified after further development and wider implementation. Future development suggestions include integration with document management system and implementing a proxy server for model independence.

Key words: generative ai, public procurement, software development, large language models, document analysis

TEKOÄLYN KÄYTTÖ OPINNÄYTTEESSÄ

Opinnäytteessäni on käytetty tekoälysovelluksia:

☐ Ei

☒ Kyllä

Ilmoitukseni mukaan olen käyttänyt opinnäytteessäni opinnäytetyöprosessin aikana seuraavia tekoälysovelluksia: Google Gemini, OpenAI ChatGPT, xAI Grok, Visual Studio Code, Windsurf, Cursor, Claude ja Microsoft Copilot.

Tekoälysovellusten nimet ja versiot:
Google Gemini: Gemini Pro 3, Gemini Flash 2.5, Gemini Pro 2.5, Gemini Flash Preview 2.5, Gemini Pro 1.5.
OpenAI ChatGPT: GPT-4o, GPT-4, GPT-5.
Anthropic Claude: Sonnet 4.5.
xAI Grok: Grok 3, Grok 4.
Github Copilot, Windsurf, Cursor, Microsoft Copilot.

Käyttötarkoitus: Ohjelmointityössä on hyödynnetty tekoälyavusteisia koodieditoreita koodin kirjoittamisen nopeuttamiseen, luomaan rutiininomaisia koodinpätkiä, etsimään virheitä sekä selittämään vieraiden kirjastojen toimintaa. Frontendissä tekoälyn käyttö oli suurinta, mutta vaati aika paljon hienosäätöä käsin. Pääosin käytössä oli Visual Studio Code, jonka rinnalla kokeiltiin Windsurf- ja Cursor-koodieditoreita.

Kielimallien selainversiota (ChatGPT, Gemini, Claude, Grok) on käytetty aiheen rajaamisessa, hahmottelussa ja työn rakenteen suunnittelussa. Lisäksi niitä on käytetty tiedonhakuun, lähteiden kartoittamiseen, teknologiaratkaisujen arviointiin sekä eri toteutusvaihtoehtojen vertailuun. Malleja on myös hyödynnetty oikeinkirjoituksen tarkistamiseen sekä kirjoitusvirheiden löytämiseen tekstistä.

Osiot, joissa tekoälyä on käytetty:

- Johdanto
- Toteutus
- Pohdinta
- Koko opinnäytetyö
- Frontend

Olen tietoinen siitä, että olen täysin vastuussa koko opinnäytteeni sisällöstä, mukaan lukien osat, joissa on hyödynnetty tekoälyä, ja hyväksyn vastuun mahdollisista eettisten ohjeiden rikkomuksista.

SISÄLLYS

LYHENTEET JA TERMIT

API	Application Programming Interface. Ohjelmointirajapinta, joka mahdollistaa eri ohjelmistojen välisen tiedonvaihdon ja toiminnallisuuksien hyödyntämisen ilman, että järjestelmien sisäistä toteutusta tarvitsee tuntea.
Kickoff	Projektin aloitustilaisuus tai -vaihe, jossa määritellään muun muassa projektin tavoitteet, aikataulu, roolit ja vastuut.
LLM	Large Language Model. Neuroverkkoihin perustuva kielimalli, joka on koulutettu valtavilla tekstiaineistoilla ja kykenee tuottamaan ja ymmärtämään luonnollista kieltä.
Prompt	Kehote tai ohje, jonka käyttäjä antaa kielimallille tai tekoälyjärjestelmälle.
Zero-shot	Menetelmä, jossa kielimalli suorittaa tehtävän ilman esimerkkikehotetta pelkästään annetun ohjeen perusteella.
Few-shot	Menetelmä, jossa kielimallia ohjataan antamalla muutama esimerkkikehote ennen varsinaista tehtävää.
CoT	Chain of thought tarkoittaa kielimalleissa ohjaustapaa, jossa malli tuottaa näkyviin vaiheittaisin, luonnollisella kielellä kirjoitetun päättelyketjun sen sijaan, että antaisi vain lopullisen vastauksen.

1 JOHDANTO

Julkiset kilpailutukset ovat tarkasti säänneltyjä kokonaisuuksia, joihin liittyy huomattava määrä dokumentaatiota ja vaatimuksia. Tarjouspyynnöt sisältävät usein lukuisia projektiin liittyviä liitteitä ja ehtoja, jotka on analysoitava huolellisesti. Kun organisaatio tekee päätöksen osallistua tarjouskilpailuun, kriittiseksi vaiheeksi muodostuu tiedon siirtäminen tarjousta tekevälle tiimille. Nykyisellään tarjousprojektin aloitukseen tarvittavan materiaalin koostaminen sitoo asiantuntijoilta merkittävästi resursseja tarjousprosessin alkuvaiheissa, kun tarjouspyynnön dokumenteista tiivistetään olennaiset tiedot manuaalisesti esitysmuotoon.

Toimeksiantajana opinnäytetyölle toimii Solita Oy. Solita Oy on suomalainen teknologia-, strategia- ja designyritys, joka pääosin toimii it-konsultoinnin ja it-palveluiden alalla. Tarkempana opinnäytetyön hyödyntäjänä toimii tilaajan Public Business -tiimi, joka vastaa Solitalla julkishallinnon tarjouspyyntöjen tunnistamisesta, arvioinnista, käsittelystä sekä tarjouspyyntöihin vastaamisesta osana yrityksen tarjousprosessia.

Opinnäytetyön tarkoituksena on tehostaa Public Business -tiimin myyntiprosessin alkuvaihetta ja vähentää manuaalista työmäärää. Työn tavoitteena on suunnitella ja toteuttaa generatiivista tekoälyä avuksi käyttävä työkalu Solitan sisäiseen käyttöön. Työkalun avulla käyttäjä pystyy tuottamaan kohdennettua aloitusmateriaalia PowerPoint-esityksen muodossa suoraan valittuun tarjousprojektiin. Toteutusmuodoltaan opinnäytetyö on toiminnallinen.

Työssä luodaan katsaus julkisten hankintojen tarjousprosessiin toimeksiantajan näkökulmasta. Työn lähtökohtana on organisaatiossa aikaisemmin tunnistettu aikaa vievä osa tarjousprosessin aloitusvaiheesta, joka on aloitusmateriaalien manuaalinen tiivistäminen kickoff-esityksen muotoon. Tähän ongelmaan kehitetään työssä ratkaisu automatisoimalla prosessia generatiivisen tekoälyn sekä ohjelmallisella automaatiolla. Työssä tarkastellaan ja selvitetään, kuinka generatiivinen tekoäly kykenee käsittelemään ja analysoimaan monikielisiä aineistoja, joita haetaan muun muassa EU:n TED-palvelusta, kansallisesta Hilma-palvelusta sekä Cloudia-järjestelmästä.

2 LÄHTÖKOHDAT JA TEKNINEN TAUSTA

2.1 Tarjousprosessin osana toimivat järjestelmät

Suomessa julkisten hankintojen ilmoittamiseen ja kilpailuttamiseen käytetään erityisesti kahta keskeistä sähköistä järjestelmää: Hilmaa ja Cloudia Tarjouspalvelua. Hilma on valtion ylläpitämä maksuton palvelu, jonka kautta julkisen sektorin ostajat voivat ilmoittaa tulevista, käynnissä olevista ja päättyneistä hankinnoista. Palvelu edistää hankintojen läpinäkyvyyttä ja tarjoaa avoimen rajapinnan hankintatietojen hyödyntämiseen. Hilman käyttö on pakollista kaikille julkisen sektorin hankintayksiköille, ja sen kautta voidaan myös vastaanottaa tarjouksia sähköisesti. (Hilma n.d.)

Cloudia Tarjouspalvelu puolestaan on maksuton sähköinen kilpailutusportaali, jossa julkaistaan Cloudian järjestelmää käyttävien organisaatioiden tarjouspyynnöt. Palvelun kautta toimittajat voivat jättää tarjouksensa sähköisesti ja esittää kysymyksiä tarjouspyyntöihin liittyen. Cloudia kattaa laajasti sekä kansalliset että EU-kynnysarvon ylittävät kilpailutukset ja se on yksi käytetyimmistä alustoista julkisissa hankinnoissa Suomessa. (Cloudia n.d.)

Näiden järjestelmien lisäksi on käytössä muun muassa Mercellin kaupallinen kilpailutusportaali ja Hanselin julkishallinnon yhteishankintayksikkö sekä Tenders Electronic Daily (TED), joka on Euroopan unionin kilpailutusportaali, missä julkaistaan Euroopanlaajuiset julkishankintojen ilmoitukset. Kyseiset järjestelmät kattavat suurimmaksi osaksi hankintailmoitukset sekä Euroopan Unionin, että Suomen tasolla.

2.2 Generatiivinen tekoäly ja kielimallit

Generatiivinen tekoäly viittaa järjestelmiin, jotka luovat uutta sisältöä, kuten tekstiä, kuvia, ääntä tai ohjelmakoodia, annettujen ohjeiden perusteella (OpenAI 2023). Suuret kielimallit (Large Language Models, LLM) ovat keskeinen osa generatiivista tekoälyä ja erikoistuneet erityisesti luonnollisen kielen käsittelyyn ja tuottamiseen.

Kielimallit koulutetaan valtavilla tekstiaineistoilla, kuten kirjoilla ja verkkomateriaalilla, minkä aikana ne oppivat ennustamaan seuraavaa sanaa annetussa kontekstissa (Brown ym., 2020, 8). Tämän prosessin tuloksena ne kykenevät suorittamaan monenlaisia kielellisiä tehtäviä, kuten tekstin tiivistämistä, kääntämistä ja kysymyksiin vastaamista (Brown ym. 2020, 14–20, 25). Alan tunnetuimpia malleja ovat OpenAI:n GPT, Metan avoimen lähdekoodin Llama, Googlen Gemini-sarja, xAI:n Grok, Deepseekin Deepseek sekä Anthropicin Claude (Minaee ym. 2024, 5). Erityisesti kielimallien kyky tiivistää tekstiä tekee niistä tehokkaan apuvälineen esitysdiojen tekstin luomiseen (Minaee ym. 2024, 7).

2.3 Kielimallien käyttö rajapintojen kautta

Modernit kielimallit ovat laajasti saatavilla kehittäjille API-rajapintojen kautta, mikä mahdollistaa niiden helpon integroinnin erilaisiin sovelluksiin. Suurimmat toimijat, kuten Google, OpenAI, Anthropic ja xAI, tarjoavat kielimalleja monipuolisiin käyttötarkoituksiin sovelluskehityksen osaksi omien rajapintojensa kautta (Google, 2025a; OpenAI n.d.)

Kielimallien rajapinnat eivät rajoitu pelkkään tekstin käsittelyyn, vaan ne tukevat myös multimodaalisia syötteitä. Tämä tarkoittaa, että samaan pyyntöön voidaan liittää esimerkiksi tekstiä, kuvia, ääntä ja jopa videota, mikä avaa mahdollisuuksia monipuolisiin sovelluksiin, kuten kuvien analysointiin tai puheentunnistukseen tai kuten opinnäytetyössä pdf-tiedostojen käyttämistä tekstin luontiin (Anthropic n.d.; Google 2025a; OpenAI n.d.; xAI n.d.).

Kielimallien käyttöön voidaan hyödyntää suoraan yritysten omien rajapintojen lisäksi myös keskitettyjen pilvialustojen, kuten Amazon Bedrockin-, Microsoft Azuren- tai Google Cloud Vertex AI:n -palveluiden kautta (Amazon Web Services n.d.; Google n.d.; Microsoft n.d.).

Lisäksi kielimalleja on mahdollista käyttää myös niin sanotusti proxyn eli välityspalvelimen kautta. Alan standardi on tällä hetkellä palvelu nimeltä LiteLLM, joka tarjoaa niin kaikki tunnetuimmat kielimallit kuin monia muitakin. Kokonaisuudessa

LiteLLM:n sivuilla kerrotaan palvelun kautta käytössä olevan yli sata kielimallia (LiteLLM n.d.).

2.4 Projektissa käytetty kielimalli

Projektissa käytettiin Googlen kielimallia nimeltä Gemini 2.5-Flash. Malli valikoitui käyttöön, koska se mahdollisti useiden laajojen PDF-dokumenttien natiivin käsittelyn ilman erillistä vektoritietokantaa tai tiedostojen esiprosessointiputkea, joka vähensi arkkitehtuurin monimutkaisuutta ja nopeutti prototyypin kehitystä.

Gemini 2.5-Flash on suunniteltu nopeaan ja monipuoliseen tekstin käsittelyyn, mikä tekee siitä soveltuvan dokumenttien tiivistämiseen ja esitysmateriaalin generointiin. Mallin yhteydessä tarjottu Files API mahdollisti useiden PDF-tiedostojen liittämisen samaan pyyntöön kontekstiksi ilman erillistä esiprosessointiputkea. Tämä oli keskeinen vaatimus, sillä julkiset tarjouspyynnöt koostuvat usein useista rinnakkaisista dokumenteista, jolloin niitä tulee analysoida kokonaisuutena, jotta saavutetaan hyvä kokonaiskuva projektista.

Vaikka vastaavia tiedostojen käsittelyä tukevia rajapintoja on saatavilla muilta toimijoilta, Geminin ratkaisu valittiin, koska se tarjosi kehitysvaiheessa riittävän teknisen suorituskyvyn, selkeän dokumentaation ja käyttöönoton sekä kustannustehokkaan kokeilumahdollisuuden. Rajapinnan rajoitukset, kuten tiedostojen 48 tunnin säilytysaika, 20 gigatavun kokonaiskapasiteetti ja yksittäisen tiedoston 2 gigatavun enimmäiskoko olivat projektin kohdalla riittävät eivätkä muodostaneet teknistä estettä.

Vaihtoehtoisesti samanlaisen järjestelmän olisi voinut rakentaa itse hyödyntämällä avoimen lähdekoodin kirjastoja, kuten LlmaIndexiä, yhdistettynä vektoritietokantaan. Tämä kuitenkin olisi lisännyt huomattavasti teknistä monimutkaisuutta ja kehitystyön määrää, jonka takia päätettiin pysyä natiivin tiedostotuen tarjoavassa ratkaisussa.

2.5 Kielimallien käytön eri menetelmät

Kielimallien rajapintoihin voi ajaa pyyntöjä muutamilla eri promptaus tavoilla, joita ovat few-shot, zero-shot ja one-shot-promptaus. Projektissa sovellettiin few-shot promptausta, jossa mallille annetaan muutamia esimerkkikehotteita ja -vastauksia, joiden avulla mallia ohjataan tuottamaan halutunlaista sisältöä.

Few-shot-promptaus eroaa zero-shot-promptauksesta siten, että zero-shot-tilanteessa mallille annetaan ainoastaan tehtävän kuvaus ilman esimerkkivastauksia (Google 2025d). Esimerkiksi kehotteessa ”Käännä tämä teksti suomeksi: ’Hello, how are you?’” mallille ei anneta mallivastausta, vaan sen odotetaan tuottavan käännös pelkän ohjeen perusteella.

One-shot menetelmässä annetaan yksi oikeanlainen esimerkkivastauspari, jonka perusteella malli tekee seuraavan ennusteen. Jolloin aikaisemmassa esimerkissä olisi mukana myös oikea vastaus eli ”Hei, miten voit?”. Few-shot menetelmää käytettäessä mallille annetaan useita oikeanlaisia esimerkkivastauspareja, muuten toimintaperiaate on muuten sama.

Googlen (2025d) ohjeiden mukaan few-shot-menetelmän onnistumisen kannalta keskeisiä tekijöitä ovat:

- Esimerkkien käyttäminen mallin ohjaamiseksi.
- XML-tyyppisten merkintöjen hyödyntäminen esimerkkien erottelussa.
- Sopivan määrän esimerkkien valinta (liian vähän ei riitä, liian monta voi johtaa ylisovittamiseen).
- Yhdenmukainen muotoilu kaikissa esimerkeissä.

Kaikki muut menetelmät hyödynnettiin projektissa, paitsi XML-tyyppisen erottelun käyttö vastauksissa ja esimerkeissä. Kuvassa 1 tarkemmin esitelty esimerkki googlelta siitä, kuinka esimerkkejä ja vastausta voi erotella XML-tyyppisillä merkeillä.

Use XML and other delimiters to structure complex prompts

For complex prompts, use XML and other delimiters to separate components of a prompt. You can use `BEGIN` and `END` or `{}` section delimiters for complex and lengthy prompt components to clearly distinguish them from the actual instructions.

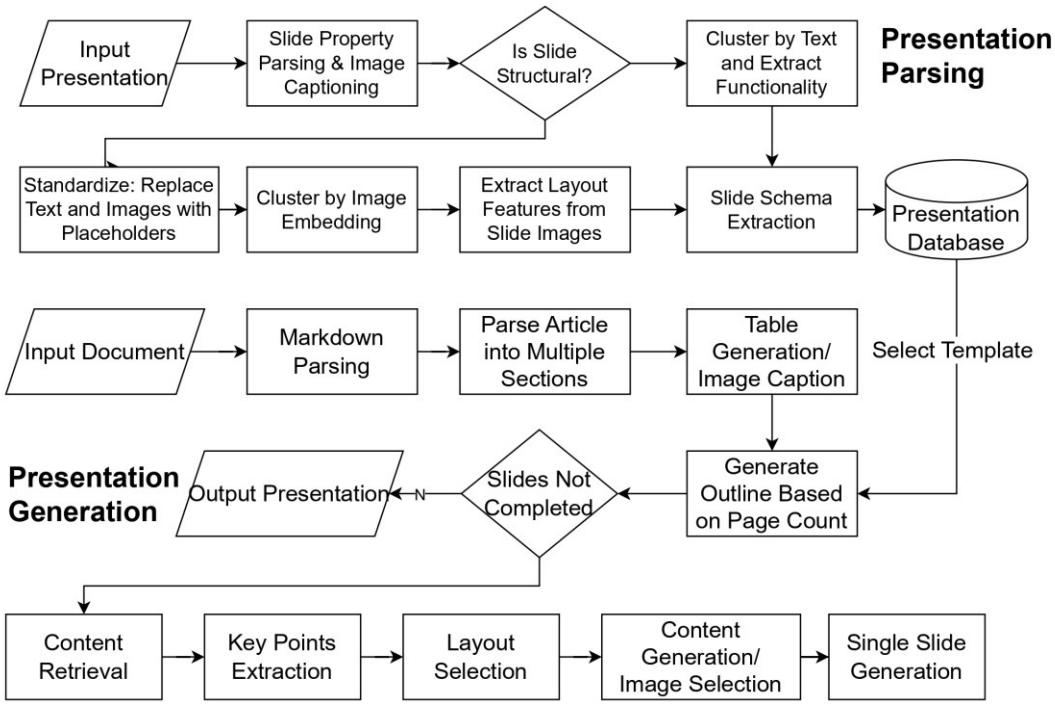
```
You are a chatbot agent answering  customer's questions in a chat.
Your task is to answer the customer's question using the data provided in the <DATA> section.
  - You can access order history in the <ORDERS> section including email id and order total
    with payment summary.
  - Refer to <ORDERLINES> for item level details within each order in <ORDERS>.

Today is 2024-01-29

<DATA>
<ORDERS>
{OrderId|CustomerEmail|CreatedTimestamp|IsCancelled|OrderTotal|PaymentSummary
CC10182|222larabrown@gmail.com|2024-01-19|true|0.0|Not available
CC10183|baklavainthebalkans@gmail.com|2024-01-19|true|0.0|Not available}
{...}
...
</ORDERS>

<ORDERLINES>
OrderId|OrderLineId|CreatedTimestamp|ItemDescription|Quantity|FulfillmentStatus|ExpectedDeliveryDate
|ActualDeliveryDate|ActualShipDate|ExpectedShipDate|TrackingInformation|ShipToAddress|CarrierCode|De
liveryMethod|UnitPrice|OrderLineSubTotal|LineShippingCharge|TotalTaxes|Payments CC10182|1||Shorts|0.
0|unshipped|2024-01-31|2024-02-01|2024-01-30|2024-01-29||||ShipToAddress|115.99|0.0|0.0|0.0|
...
</ORDERLINES>
</DATA>

<INSTRUCTIONS>
- If there is no data that can help answer the question, respond with "I do not have this
  information. Please contact customer service".
- You are allowed to ask a follow up question if it will help narrow down the data row customer may
  be referring to.
- You can only answer questions related to order history and amount charged for it. Include OrderId
  in the response, when applicable.
- For everything else, please redirect to the customer service agent.
- Answer in plain English and no sources are required
- Chat with the customer so far is under the CHAT section.
</INSTRUCTIONS>

QUESTION: How much did I pay for my last order?
ANSWER:
```

Kuva 1 Googlen esimerkki monimutkaisempien promptien rakenteen parantamisesta XML:n ja muiden erottimien avulla. (Google, 2025c)

Tämä on hyvin tyypillinen tapa saada kielimalli palauttamaan joissakin tietyssä muodossa, koska välillä vastauksien muotoilussa voi esiintyä vaihtelua. Toisaalta projektin kielimallilta vaadittu palautusmuoto oli pelkkä tekstiä rivinvaihdoilla eroteltuna, eikä mallin vastauksien muotoilussa esiintynyt paljon vaihtelua.

2.6 Aikaisemmat projektit ja tutkimukset

Markkinoilla on jo runsaasti kaupallisia tekoälyavusteisia PowerPoint-palveluita. Useimmat näistä toimivat siten, että käyttäjä syöttää palveluun tekstiä tai dokumentteja, minkä jälkeen tekoäly kokoaa keskeiset asiat selkeiksi kokonaisuuksiksi ja generoi niiden pohjalta esitysdioja. Projektin aika perehdyttiin lyhyesti seuraaviin sovelluksiin: Popai (n.d.), Powerdrill (n.d.) ja Gamma (n.d.). Sovellusten toimivuus vaikutti olevan hyvällä tasolla, vaikka testausjakso oli rajallinen.
Tutkimuksellisesta näkökulmasta merkittävä vertailukohta on Zhenginn ym. (2025) tutkimuspaperi PPTAgent-nimisestä avoimen lähdekodin sovelluksesta. Se tuottaa PowerPoint-esityksiä olemassa olevan esimerkkimateriaalin perusteella. Sovellus sisältää myös sisäisen työkalun nimeltä PPTEval, jolla arvioidaan tuotetun PowerPoint esityksen tasoa. Kuvassa 2 esitellään PPTAgentin työnkulkua.



Kuva 2 Työkulku PPTAgent versiossa 0.0.1. (Zheng ym. 2025).

PPTEval arvioi tuotetun esityksen tasoa kolmessa aihekentässä: sisällössä, muotoilussa ja johdonmukaisuudessa. Sovelluksen toimintaperiaate on seuraava, se ensin analysoi ja jakaa esimerkkimateriaalin pienempiin palasiin teko-

älyavusteisesti saadakseen paremman kuvan koko esimerkkimateriaalista. Tämän jälkeen sivujen pääpiirteitä käytetään esimerkkinä generoitavan materiaalin pohjaksi, jonka jälkeen generoidaan vielä itse sivujen teksti- tai kuvamateriaali.

Vaikkakin PPTAgentin työnkulku on huomattavasti monimutkaisempi, kuin opinnäytetyössä kehitetyn sovelluksen, sen tarkoitusperänä on luoda esimerkkimateriaalin perusteella samantyylisiä esityksiä. Kun taas opinnäytetyössä esiteltävä sovellus pyrkii luomaan samantyylisiä PowerPoint-esityksiä erilaisien dokumenttien perusteella ja pysymään tarkasti alkuperäisen PowerPoint-esityksen pohjan rakenteessa. Lisäksi muun muassa tekstin sovitus dian kokoon tehdään PPTAgentissa kielimallin kehotteella ja tiivistämällä tekstiä niin kauan, että teksti mahtuu diaan, kun taas opinnäytetyön sovelluksessa ongelma on pyritty ratkaisemaan niin, että tekstin fonttikokoa pienennettään merkkimäärän suurentuessa.

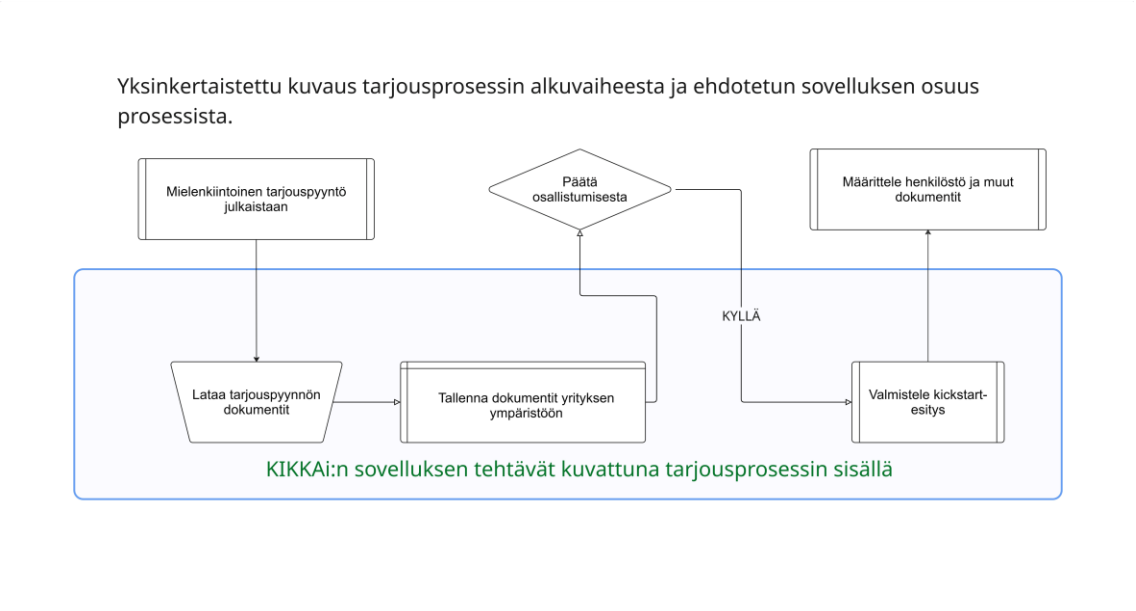
Kaiken kaikkiaan PPTAgentin toteutus on hyvin samankaltainen, mutta se on rakennettu hieman yleisempään käyttötarkoitukseen, jossa yhdistetään kielimallien hyödyntäminen PowerPoint-esitysten generoimiseen.

3 TOIMINNALLINEN OSUUS JA KEHITTÄMISPROSESSI

3.1 Tarjousprosessin nykytila ja kehityskohteet

Tarjousprosessiin tutustuminen aloitettiin ensin sisäisesti käymällä läpi prosessikaavioita sekä osallistumalla tarjousprosessin alkuvaiheisiin, jonka perusteella muodostettiin kuva siitä, miten julkisen puolen tarjousprosessi toimii Solitalla. Keskustelujen perusteella havaittiin, että tarjousprosessin alkuvaiheessa eniten aikaa vei kickoff-esityksen, eli aloitusmateriaalin laatiminen projektille. Aloitusmateriaalin tarkoituksena on jäsentää projektin lähtökohdat, tavoitteet ja vaatimukset ennen varsinaisen tarjouksen laatimista.

Aika vievin osuus aloitusmateriaalin laatimisessa johtuu dokumenttien määrästä, sillä kattavat julkiset kilpailutukset voivat sisältää kymmeniä dokumentteja, jotka määrittelevät tehtävän työn sisällön ja vaatimukset. Kuvassa 3 esitellään sovelluksen toiminta-aluetta osana tarjousprosessia. Sovelluksen pääasiallinen tehtävä on automatisoida manuaalista dokumenttien läpikäyntiä ja tietojen koostamista PowerPoint-esitykseen. Kyseessä on esimerkinomainen kuvaus tarjousprosessin eri vaiheista, eikä sitä tule tulkita kohdeorganisaation viralliseksi prosessikuvaukseksi.



Kuva 3 havainnollistaa sovelluksen sijoittamisen tarjousprosessiin. Kaavio on laadittu opinnäytetyötä varten hahmottamaan ratkaisun sijoittumista osaksi tarjousprosessia (Pyry Ikonen 2025).

Tämä rajaus selkeytti projektin tavoitteita, sillä kehitystyö voitiin kohdistaa nimenomaan tiedonhakuvaiheeseen ja esityksen generointiin. Tällöin varsinainen päätöstenteko ja tarjouksen kirjoittamien on edelleen asiantuntijoiden vastuulla.

3.2 Kehitysmenetelmät

Projektin kehittämisessä hyödynnettiin ketteriä menetelmiä ja iteratiivista lähestymistapaa, jolloin työ eteni pienissä, hallittavissa vaiheissa. Iteratiivisessa kehityksessä ohjelmistoa rakennetaan vaiheittain, mikä mahdollistaa vaatimusmäärittelyn tarkentumisen projektin aikana ja nopean reagoinnin muutoksiin (Abrahamnsson ym. 2002, 16–19). Tavoitteena oli saada nopeasti toimiva prototyyppi ja kehittää sitä jatkuvan palautteen perusteella.

Työ aloitettiin nykyprosessin analysoinnilla ja pullonkaulojen tunnistamisella, jonka pohjalta määriteltiin tärkeimmät kehityskohteet. Kehitystyössä painotettiin erityisesti automaatiota ja toistuvien tehtävien yksinkertaistamista, esimerkiksi kickoff-materiaalin luomisessa. Dokumentointi ja palautteen keruu olivat keskeinen osa kehitystä. Keskustelut Public Business -yksikön työntekijöiden kanssa auttoivat varmistamaan, että kehitetty ratkaisu vastasi todellisiin tarpeisiin ja oli käytännössä sovellettavissa.

Projektin organisoinnissa käytettiin GitHubin projektinhallintataulua, johon kirjattiin tehtävät ja työn tila. Työ jaettiin selkeisiin sprintteihin, ja edistymistä seurattiin säännöllisesti tiimipalavereissa. Tämä mahdollisti nopean reagoinnin muutoksiin ja varmistaa, että kehitystyö pysyi tavoitteiden mukaisena. Miroa käytettiin muutamien työnkulku ja arkkitehtuuridiagrammien luomiseen.

Projektin loppupuolella generoidun aloitusmateriaalin arviointia tehtiin myös toimeksiantajan puolesta, saadaksemme paremman käsityksen siitä, olivatko kielimallin tuottamat referenssit dokumentteihin luotettavia, sekä arvioimaan oliko dioihin tuotettu teksti asianmukaista ja ymmärrettävää.

3.3 Teknologiat ja työkalut

3.3.1 Versionhallinta ja projektinhallinta

Git on hajautettu versionhallintajärjestelmä ja Github on sen pilvipohjainen alusta, joka mahdollistaa koodin tallentamisen, jakamisen ja yhteistyön. (Github n.d.-a) Git ja GitHub valittiin projektin versionhallintaan, koska ne mahdollistavat tehokkaan yhteistyön ja koodin hallinnan. Ne ovat alan standardeja työkaluja, joita käytetään laajasti ohjelmistokehityksessä. GitHubia hyödynnettiin projektissa julkaisujen hallintaan, sekä tehtävien seurantaan projektinhallintataulun avulla.

3.3.2 Kehitysympäristö ja tekoälyavusteiset työkalut

Visual Studio Code on monipuolinen avoimen lähdekoodin koodieditori, joka tukee monia ohjelmointikieliä ja tarjoaa laajan valikoiman laajennuksia (Microsoft n.d.). VS Code toimi projektin aikana pääasiallisena koodieditorina. Sen tekoälyavusteiset ehdotukset olivat riittäviä ja hyödyllisiä erityisesti rutiininomaisissa tehtävissä, kuten testien kirjoittamisessa, toistuvan koodin tuottamisessa, virheenkorjauksessa, koodin selittämisessä sekä säännöllisten lausekkeiden (RegEx) luomisessa (GitHub n.d.-b).

Projektin aikana myös kokeiltiin kahta AI-natiivia koodieditoria Cursoria ja Windsurffia. Cursor sekä Windsurf perustuvat VS Coden avoimeen lähdekoodiin, muta niihin on integroitu tekoäly syvällisemmin suoraan editorin ytimeen (Cursor n.d.-a; Cursor n.d.-b; Windsurf n.d.-a). Molemmat editorit toimivat pääosin hyvin, mutta projektin kasvaessa kontekstin laajentuessa niiden hyödyntäminen kävi haastavaksi. Tästä syystä kehitys tapahtui suurimmaksi osaksi Visual Studio Codessa.

3.3.3 Riippuvuuksien hallinta ja konttiteknologia

Sovelluksen riippuvuuksien hallintaan valittiin Pipenv-työkalu. Valintaan vaikutti sen kyky yhdistää virtuaaliympäristöjen hallinnan ja kirjastojen asennuksen yhdeksi työkaluksi (Pipenv n.d.), minkä lisäksi työkalu oli tekijälle entuudestaan tuttu. Pipenvin toimintaperiaate muistuttaa JavaScript-kehityksestä tuttua npm-järjestelmää. Pipfile-tiedosto määrittelee projektin riippuvuudet ja Pipfile.lock-tiedosto lukitsee tarkat versiot. Tämä takaa deterministisen kehitysympäristön,

jossa jokainen projektin parissa työskentelevä käyttää varmuudella samoja, toimiviksi todettuja kirjastoversioita.

Docker on ohjelmistoalusta, joka mahdollistaa sovellusten rakentamisen, testaamisen ja käyttöönoton eristetyissä ympäristöissä, joita kutsutaan konteiksi (container). Kontit sisältävät kaiken tarvittavan sovelluksen suorittamiseen sen perusteella, mitä määritellään Dockerfile- ja compose.yaml-tiedostoissa. Näihin tiedostoihin kirjataan esimerkiksi käytettävä pohjakuva, asennettavat riippuvuudet, ympäristömuuttujat ja palveluiden väliset yhteydet (Docker 2025). Docker valittiin projektissa käyttöön, jotta koodin käyttöönotto toimeksiantajalle tulevaisuudessa olisi mahdollisimman helppoa.

3.3.4 Backend ja Frontend

Sovelluksen taustajärjestelmän perustaksi valittiin Python-ohjelmointikieli ja FastAPI-kirjasto. FastAPI on moderni ja suorituskykyinen web-kehys rajapintojen rakentamiseen (FastAPI 2025). Merkittävin syy Pythonin valintaan oli pienempi kirjasto nimeltä python-pptx, jonka avulla toteutettiin automaattinen PowerPoint-esitysten luonti. Python-pptx on vakiintunut avoimen lähdekoodin Python-kirjasto, jolla voidaan luoda, muokata ja hallita PowerPoint-esityksiä ohjelmallisesti ilman PowerPoint-sovellusta (python-pptx n.d.). Ominaisuuksiin kuuluu mm. diojen lisääminen, tekstin, kuvien taulukkojen ja kaavioiden hallinta sekä muokkaaminen. Kirjasto mahdollistaa siis PowerPointin ominaisuuksien hallinnan dynaamisesti ilman tarvetta työpöytäsovellukselle.

Käyttöliittymä toteutettiin kevyenä ratkaisuna staattisesti käyttäen Tailwind CSS:ää, JavaScriptiä ja HTML:ää sen yksinkertaisuuden, suorituskyvyn ja helpon ylläpidettävyyden vuoksi (Tailwind Labs n.d.). Ratkaisu valittiin, koska sovelluksen pääpaino on taustajärjestelmän dataprosessoinnissa ja kevyt käyttöliittymä mahdollisti nopean iteroinnin ilman JavaScript-kehysten tuomaa konfigurointikuormaa.

3.3.5 Redis ja asynkroninen tehtävienhallinta

Sovelluksen taustaprosessien hallintaan valittiin Redis ja ARQ-kirjasto. Redis on avoimen lähdekoodin muistinvarainen tietovarasto, jota käytetään tyypillisesti välimuistina tai viestinvälittäjänä (Redis n.d.). ARQ on Python-kirjasto, joka on suunniteltu asynkronisten taustatehtävien hallintaan. Se rakentuu Rediksen päälle ja hyödyntää Pythonin asyncio-ominaisuuksia (ARQ n.d.). Yhdessä kirjastojen tehtävänä on huolehtia yhtäaikaisista aikaa vievistä toiminnoista, kuten tiedostojen lataamisesta ja esitysten luomisesta, ilman että koko sovellus odottaa edellisen toiminnon valmistumista.

Käytännössä järjestelmä toimii siten, että käyttäjän tekemä pyyntö siirretään tehtävä jonoon odottamaan suoritusta. Erilliset taustaprosessit poimivat tehtäviä tästä jonosta ja suorittavat ne itsenäisesti.

3.3.6 Selainautomaatio

Tarjousdokumenttien puoliautomaattiseen lataamiseen valittiin Playwright-kirjasto. Playwright on selainautomaatiotyökalu, joka mahdollistaa selaimen ohjaamisen ohjelmallisesti headless-tilassa, eli ilman graafista käyttöliittymää (Microsoft n.d.). Tämä on välttämätöntä, jotta selainta voidaan ajaa palvelinympäristössä ja docker-konteissa. Playwrightin avulla automatisoitiin vuorovaikutus verkkosivuston kanssa simuloimalla käyttäjän toimintaa, mikä mahdollisti tarjouspyyntödokumenttien lataamisen verkkopalvelusta. Koska Tarjouspalvelu ei tarjoa avointa rajapintaa tarjousdokumenttien noutamiseen, toteutettiin aineiston haku käyttämällä kirjastoa. Ilman automatisointia tarjousaineistojen manuaalinen lataaminen ja siirtäminen järjestelmien välillä veisi asiantuntijalta huomattavan määrän työaikaa. Valittu menetelmä mahdollistaa julkisten tarjouspyyntöjen tehokkaamman hyödyntämisen.

3.4 Prototyypin suunnittelu ja toteutus

Prototyypin suunnittelu ja toteutus eteni iteratiivisesti. Työssä sovellettiin MVP-ajattelumallia (Ries 2011), jonka tavoitteena oli tuottaa mahdollisimman nopeasti

testattava versio keskeisten toiminallisuuksien testaamiseksi. Parannuksia käytettävyyteen sekä itse sovelluslogiikkaan tehtiin prototyypin käytöstä saadun palautteen pohjalta.

Ensimmäinen versio prototyypistä toimi manuaalisella dokumenttien latauksella, jotka sovellus tallensi paikalliseen ”data/docs”-kansioon. Tästä dokumentit syötettiin Geminin Files-rajapinnalle, mikä mahdollisti niiden hyödyntämisen kielimallin kontekstina. Kun tiedostot olivat mallin käytettävissä, sille lähetettiin kohdennettuja kehotteita, joiden avulla se analysoi aineistoa ja generoi halutut tekstisisällöt. Lopuksi kielimallilla generoitu teksti yhdistettiin PowerPoint-pohjaan, käyttäen python-pptx-kirjastoa. Valmiit PowerPoint-esitykset tarjottiin tässä vaiheessa suoraan käyttäjälle käyttöliittymästä, ponnahdusikkuna latauksena.

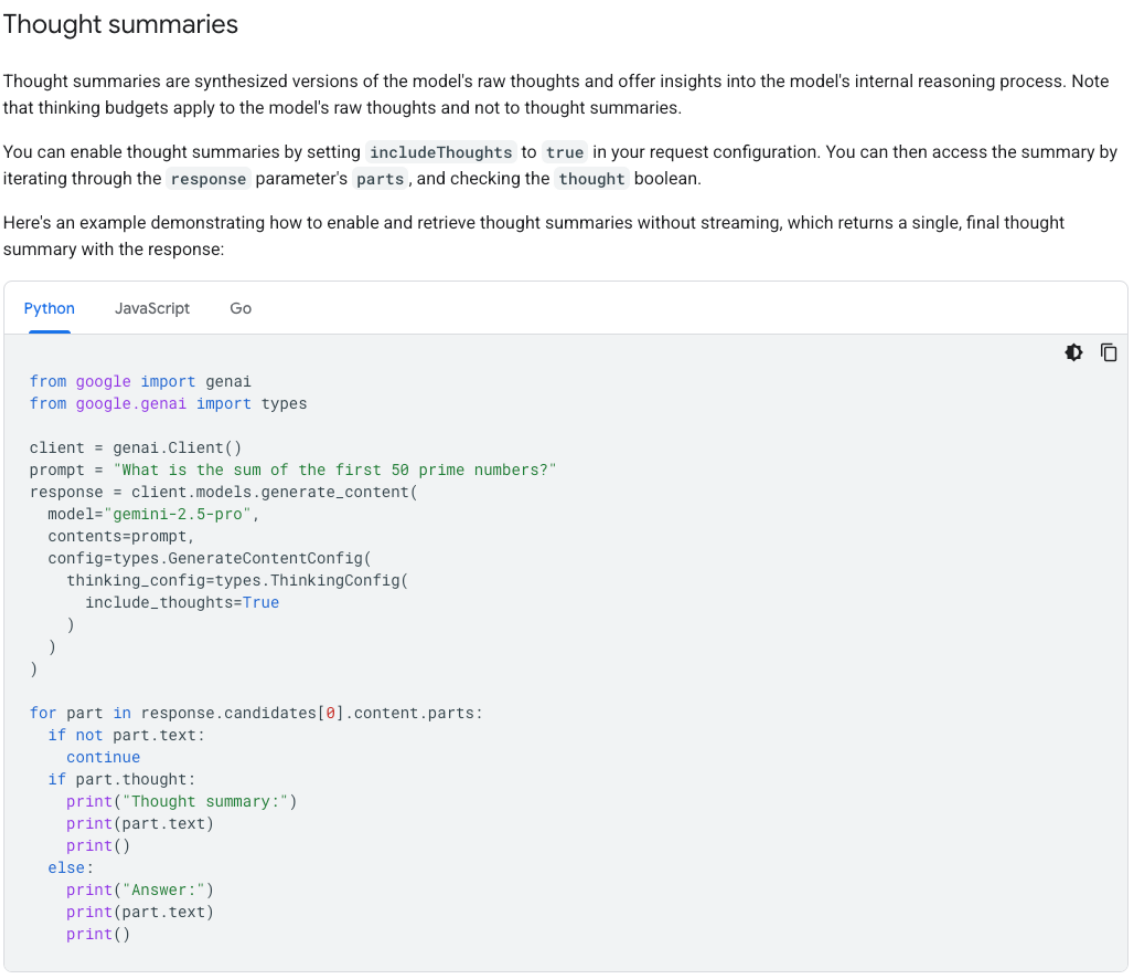
Toinen versio prototyypistä piti sisällään ison harppauksen edelliseen. Backendin taustana aikaisemmin toiminut Flask -kirjasto vaihtui FastAPI-kirjastoon ja projektin rakenne muuttui merkittävästi vastamaan uuden kirjaston suositeltua esimerkkirakennetta. Tässä vaiheessa projektia sovellus kontitettiin käyttämällä Docker palvelua. Tämä tarkoittaa sitä, että sovellus ja sen riippuvuudet, asetukset sekä ajoaikainen ympäristö yhteen siirrettävään konttiin. Tämä takasi sen, että sovellus toimii samalla tavalla riippumatta siitä, missä ympäristössä sitä käytetään.

Kehotteiden suunnittelu eteni yhteistyössä toimeksiantajan asiantuntijoiden kanssa. Kehitystyön aikana käytiin läpi lukuisia testikierroksia, joiden perusteella määriteltiin, mikä informaatio on kriittistä milläkin PowerPoint-dialla. Julkisen sektorin tarjouspyynnöt muodostavat usein laajoja aineistokokonaisuuksia, jotka voivat sisältää kymmeniä erillisiä liitetiedostoja. Näitä voivat olla yleisten sopimusehtojen lisäksi muun muassa teknisiä liitteitä, arkkitehtuurikuvauksia, tietoturvavaatimuksia, hallinnollisia dokumentteja, kuten laskutusohjeita tai tyhjiä lomakepohjia.

Aineiston suuren koon ja sen olennaisuuden kannalta havaittiin, ettei kaikkien dokumenttien syöttäminen kielimallille ole järkevää. Tästä syystä sovellukseen kehitettiin ominaisuus, jossa käyttäjä voi valita, mitkä ladatuista dokumenteista

syötetään kielimallille. Tällöin käyttäjä voi varmistaa, että vain oleellisimmat dokumentit sisällytetään kickoff-materiaalin generoimiseen.

Yksi projektin keskeisistä haasteista liittyi kielimallin ajatteluprosessin (COT, chain of thought) hyödyntämiseen. Ajatteluprosessin avulla pyrittiin hahmottamaan, miten kielimalli etenee kohti lopullista vastausta, mikä parantaa tulosten tulkittavuutta ja luotettavuutta. Ajatteluprosessin on myös tarkoituksena toimia eräänlaisena lähdeluettelona sille, mistä dokumentista mikäkin dian teksti on tiivistetty. Natiivi tapa poimia ajatteluprosessi kielimallin vastauksesta toimii Googlen esimerkkien mukaan kuvassa 4 seuraavasti.



Thought summaries

Thought summaries are synthesized versions of the model's raw thoughts and offer insights into the model's internal reasoning process. Note that thinking budgets apply to the model's raw thoughts and not to thought summaries.

You can enable thought summaries by setting `includeThoughts` to `true` in your request configuration. You can then access the summary by iterating through the `response` parameter's `parts`, and checking the `thought` boolean.

Here's an example demonstrating how to enable and retrieve thought summaries without streaming, which returns a single, final thought summary with the response:

Python JavaScript Go

```python
from google import genai
from google.genai import types

client = genai.Client()
prompt = "What is the sum of the first 50 prime numbers?"
response = client.models.generate_content(
  model="gemini-2.5-pro",
  contents=prompt,
  config=types.GenerateContentConfig(
    thinking_config=types.ThinkingConfig(
      include_thoughts=True
    )
  )
)

for part in response.candidates[0].content.parts:
  if not part.text:
    continue
  if part.thought:
    print("Thought summary:")
    print(part.text)
    print()
  else:
    print("Answer:")
    print(part.text)
    print()
```

Kuva 4. Googlen dokumentaation esimerkki siitä, kuinka kielimallin tuottama ajatteluprosessi (thought summary) voidaan erottaa varsinaisesta vastauksesta ohjelmallisesti (Google 2025b).

Googlen tarjoama natiivi ajatteluprosessi ei sisältänyt projektille olennaista tietoa siitä, mihin lähdedokumentin mikäkin tekstissä esitetty tieto perustui. Tästä

syystä lisäsimme ajatteluprosessin luonnin osaksi kielimallin kehotetta, minkä avulla saimme mallin kuvaamaan vastauksensa perustelut ja viittamaan käyttämiinsä lähdedokumentteihin. Projektin aikana ajatteluprosessi poimittiin suoraan kielimallin tekstimuotoisesta vastauksesta, eikä kyse ollut rajapinnan natiivista palauttamasta päättelyrakenteesta.

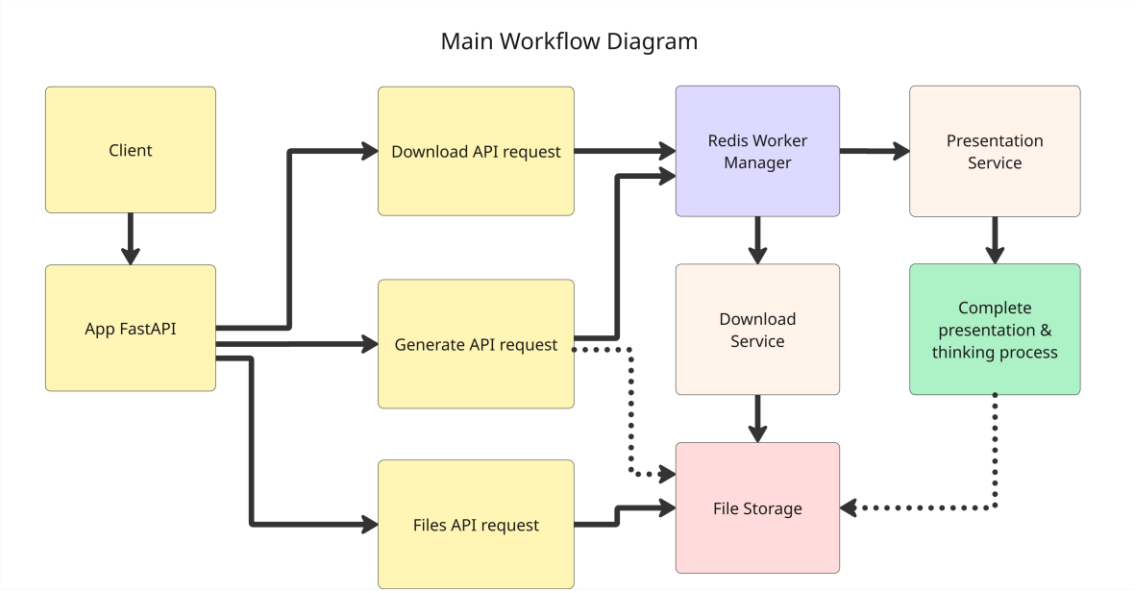
Tämän vuoksi osa ajatteluketjusta saattoi olla kielimallin hallusinaatiota. Vaikka vastaukset vaikuttivat usein loogisilta ja vastasivat tehtyjä dioja, ei ollut keinoa varmistaa, että ajatteluprosessi todella kuvasi mallin sisäistä päättelyä. Muutos johtui osittain siitä, että Google päivitti rajapintaansa ennen projektin aloitusta, mikä vaikeutti alkuperäisen ajatteluprosessin hyödyntämistä. Tämän vuoksi prosessi lisättiin osaksi kielimallin suoraa vastausta, jotta jonkinlainen ajatteluprosessi tehdyistä dioista saatiin talteen.

Aiemmin on dokumentoitu, että OpenAI ja muut vastaavat toimijat eivät julkaise mallien ”raw chain-of-thought” ajatteluprosessia käyttäjille, vaan tarjoavat sen ainoastaan tiivistettynä versiona. Päätöstä on perusteltu muun muassa turvallisuudella, käyttäjäkokemuksella ja kilpailuedun säilyttämisellä (OpenAI 12.9.2024).

3.5 Julkaisu

Julkaisuvaiheessa projektin tavoitteena oli siirtyä prototyyppivaiheesta kohti tuotantokelpoista ratkaisua, joka voitaisiin ottaa käyttöön toimeksiantajan ympäristössä. Alkuperäisen suunnitelman mukaan sovellus oli tarkoitus yhdistää SharePointin testiympäristöön, mutta rajapintojen aktivointiin liittyvät tukipyynnöt viivästyivät, eikä integraatiota saatu toteutettua julkaisuvaiheen aikataulussa. Tämän vuoksi sovelluksen toimintaa ei voitu testata osana SharePoint-ympäristöä.

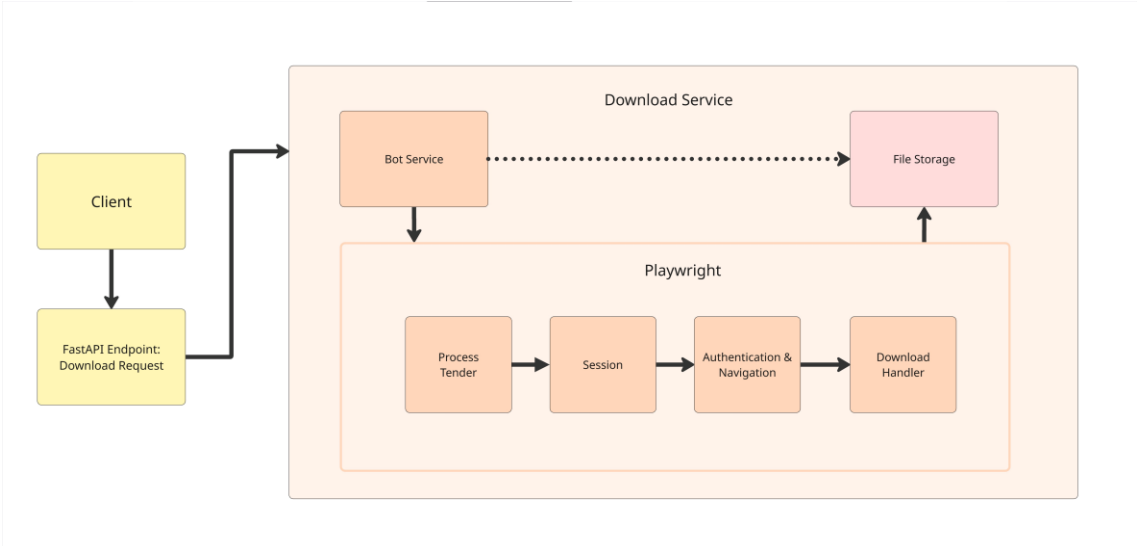
Sovelluksen esitysgenerointiprosessi muodostuu useista toisiinsa liittyvistä palveluista, jotka yhdessä mahdollistavat dokumenttipohjaisen PowerPoint-esityksen automaattisen tuottamisen. Kokonaisuus on suunniteltu asynkroniseksi, mikä mahdollistaa useiden prosessien samanaikaisen käsittelyn. Kuvassa 5 esitetään koko sovelluksen pääasiallinen työnkulku, jossa käyttäjän tekemä kutsu kulkee rajapinnan kautta oikealle palvelulle. Tämä kuva toimii yleiskatsauksena koko järjestelmän toiminnasta.



Kuva 5 Esimerkki sovelluksen pääasiallisesta työnkulusta, jossa käyttäjältä tulee kutsuja sovelluksen rajapintaan, mitkä se ohjaa eteenpäin tarvittavalla tavalla. (Pyry Ikonen 2025)

Kuvat 6 ja 7 tarkentavat kahta keskeistä pääpalvelua: latauspalvelun työnkulku (Download Service) ja esityspalvelun työnkulku (Presentation Service). Näissä kuvissa havainnollistetaan, miten tiedostojen haku ja esityksen generointi etenevät sovelluksen sisällä eri komponenttien kautta.

Prosessi alkaa käyttäjän tekemästä latauspyynnöstä, joka kulkee sovelluksen rajapinnan kautta Download Service -palveluun. Kuvassa 6 nähdään, miten tämä palvelu toimii. Bot Service toimii latauspalvelun orkestroijana, joka käynnistää ja hallinnoi lataustehtäviä sekä yhdistää eri komponenttien toiminnan.



Kuva 6 Esimerkki siitä, miten lataus päätepisteen työnkulku toimii sovelluksen oman rajapinnan kautta. (Pyry Ikonen 2025)

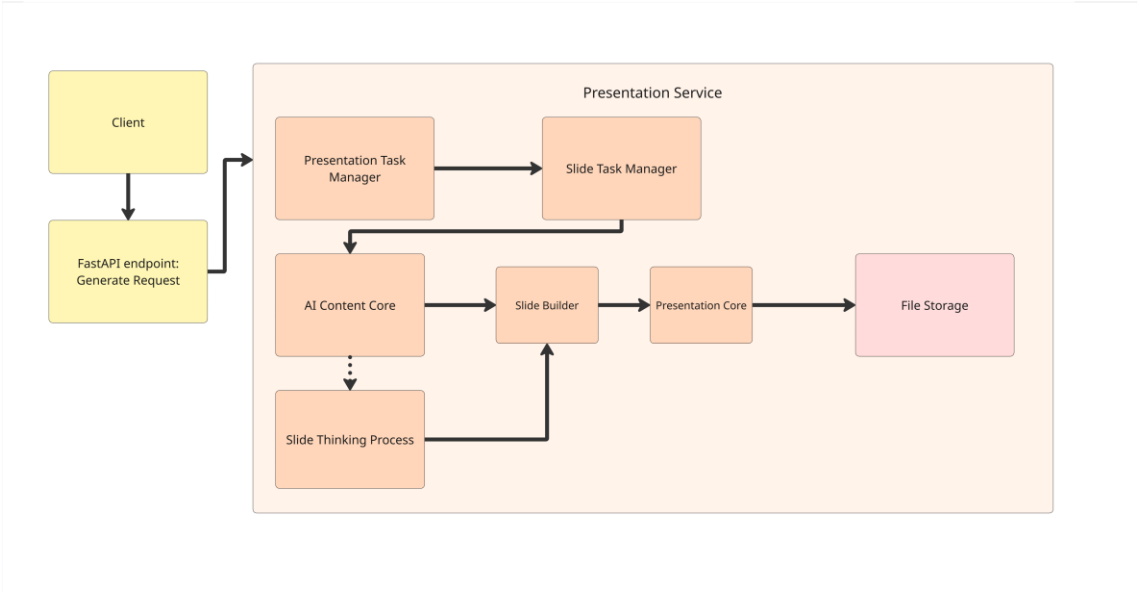
Varsinainen tiedostojen haku tapahtuu Playwright-komponentin sisällä, joka on jaettu useisiin alikomponentteihin:

- Process Tender toimii latausprosessin "orkestraattorina" eli se käynnistää ja hallinnoi kaikki tarvittavat vaiheet: selainistunnon luonnin, kirjautumisen ja navigoinnin, tiedostojen latauksen sekä aineiston käsittelyn.
- Session hallinnoi selainistuntoja ja varmistaa, että yhteys pysyy aktiivisena koko prosessin ajan.
- Authentication & Navigation automatisoi kirjautumisen ja navigoinnin kohdesivustolla, jotta tarvittavat dokumentit saadaan esiin oikeasta tarjouspyynnöstä.
- Download Handler vastaa varsinaisesta tiedostojen lataamisesta ja tallentamisesta oikeaan sijaintiin.

Tämä kokonaisuus mahdollistaa sen, että dokumentit voidaan hakea automaattisesti ulkoisista järjestelmistä ilman manuaalista käyttöä.

Kun aineisto on ladattu ja tallennettu, Presentation Task Manager käynnistää esityksen generointiprosessin. Se luo uuden työn, seuraa sen tilaa ja käynnistää taustaprosessin Redis Worker Managerin kautta. Tämä mahdollistaa useiden esitysten rinnakkaisen käsittelyn. Presentation Task Manager toimii esitystason ohjauskeskuksena, joka hallinnoi koko työnkulun etenemistä. Koko esityspalvelun työnkulku on kuvattu kuvassa 7. Slide Thinking Process vastaa kielimallin

ajatteluprosessin poimimisesta ja lisää sen itse PowerPoint-esityksen muistiinpano-osioon sekä tallentaa sen vielä erilliseen Markdown-tiedostoon.



Kuva 7 Esimerkki siitä, miten esitys päätepisteen työnkulku toimii sovelluksen oman rajapinnan kautta. (Pyry Ikonen 2025)

Sisällön tuottamisesta vastaa Slide Task Manager, joka kutsuu AI Content Core -tiedostoa. Kuvassa 8 näkyy, miten generate_content-kutsu yhdistää kielimallille annetut kehotteet (prompt) ja ladatut tiedostot. Mallille määritetyt asetukset, kuten *temperature = 0.1* ja *top_p = 0.1*, on asetettu tarkoituksella mataliksi. Tämä vähentää kielimallin satunnaisuutta ja luovuutta, mikä on tärkeää, kun halutaan tuottaa johdonmukaista ja faktapohjaista sisältöä dokumenttipohjaiseen esitykseen (Google 2025h).

```
    try:
        response = client.models.generate_content(
            model=FLASH_2_5_MODEL,
            contents=[prompt] + uploaded_files,
            config=types.GenerateContentConfig(
                system_instruction=[system_instruction],
                temperature=0.1,
                top_p=0.1,
                thinking_config=types.ThinkingConfig(
                    include_thoughts=True,
                    thinking_budget=-1 # -1 means on 0 means off
                )
            )
        )
```

Kuva 8 Esimerkki sovelluksen AI Content Core tiedostosta, jossa kutsutaan kielimallia generate_content-kutsulla yhdistettynä prompteihin ja tiedostoihin. (Pyry Ikonen 2025)

Slide builder rakentaa teknisesti diat käyttäen esimerkiksi python-pptx-kirjastoa. Se huolehtii diojen visuaalisesta rakenteesta, otsikoinnista ja sisällön asettelusta. Tarvittaessa se jakaa sisällön useammalle dialle, jos materiaalia on paljon. Kuvissa 9 ja 10 esitellään sovelluksen python-pptx osiota ja sen logiikkaa, joka rakentaa PowerPoint-esityksen.

```
# Populate Title Slide (robust)
slide1 = prs.slides[0]
title_text = data["title_slide"]["title_text"]
if slide1.shapes.title is not None:
    slide1.shapes.title.text = title_text
    title_frame = slide1.shapes.title.text_frame
    if title_frame.paragraphs:
        title_frame.paragraphs[0].font.size = Pt(36)
else:
    logger.warning("Title placeholder missing on title slide, heading not set.")
# Set subtitle if it exists in the template
for shape in slide1.shapes:
    if shape.has_text_frame and shape.name == "Subtitle 2":
        shape.text_frame.text = data["title_slide"].get("subtitle_text", "")
        if shape.text_frame.paragraphs:
            shape.text_frame.paragraphs[0].font.size = Pt(16)
```

Kuva 9 Esimerkki koodista käyttäen python-pptx kirjastoa, kyseisillä riveillä asetetaan tekstiä PowerPoint-esityksen otsikoksi sekä alaotsikoksi ja määritellään fonttikoko. (Pyry Ikonen 2025)

```python
def populate_two_column_slide(slide, left_content, right_content, left_placeholder_name, right_placeholder_name):
    left_placeholder = next((shape for shape in slide.shapes if shape.name == left_placeholder_name), None)
    if left_placeholder and left_placeholder.has_text_frame:
        text_frame = left_placeholder.text_frame
        text_frame.clear()
        for item in left_content:
            p = text_frame.add_paragraph()
            p.text = item if isinstance(item, str) else str(item)
            p.level = 0
        fit_text_to_frame(left_placeholder.text_frame)

    right_placeholder = next((shape for shape in slide.shapes if shape.name == right_placeholder_name), None)
    if right_placeholder and right_placeholder.has_text_frame:
        text_frame = right_placeholder.text_frame
        text_frame.clear()
        for item in right_content:
            p = text_frame.add_paragraph()
            p.text = item if isinstance(item, str) else str(item)
            p.level = 0
        fit_text_to_frame(right_placeholder.text_frame)
```

Kuva 10 Esimerkki koodista käyttäen python-pptx kirjastoa, kyseisessä esimerkissä asetetaan tekstiä kahden tekstilaatikon omaavan PowerPoint-esityksen sivulle. (Pyry Ikonen 2025)

Lopuksi Presentation Core kokoaa kaikki generoidut diat kokonaiseksi PowerPoint-esitykseksi. Tämä valmis esitys voidaan tallentaa valittuun tiedostojenhallinta ratkaisuun ja jakaa eteenpäin sovelluksen sisällä. Kuvassa 5 nähdään vihreällä laatikolla kuvattuna koko prosessin päätepiste, jossa esitys ja siihen liittyvä ajatteluprosessi ovat valmiina. Tämän jälkeen tiedostot tallennetaan käytettyyn tiedostoratkaisuun.

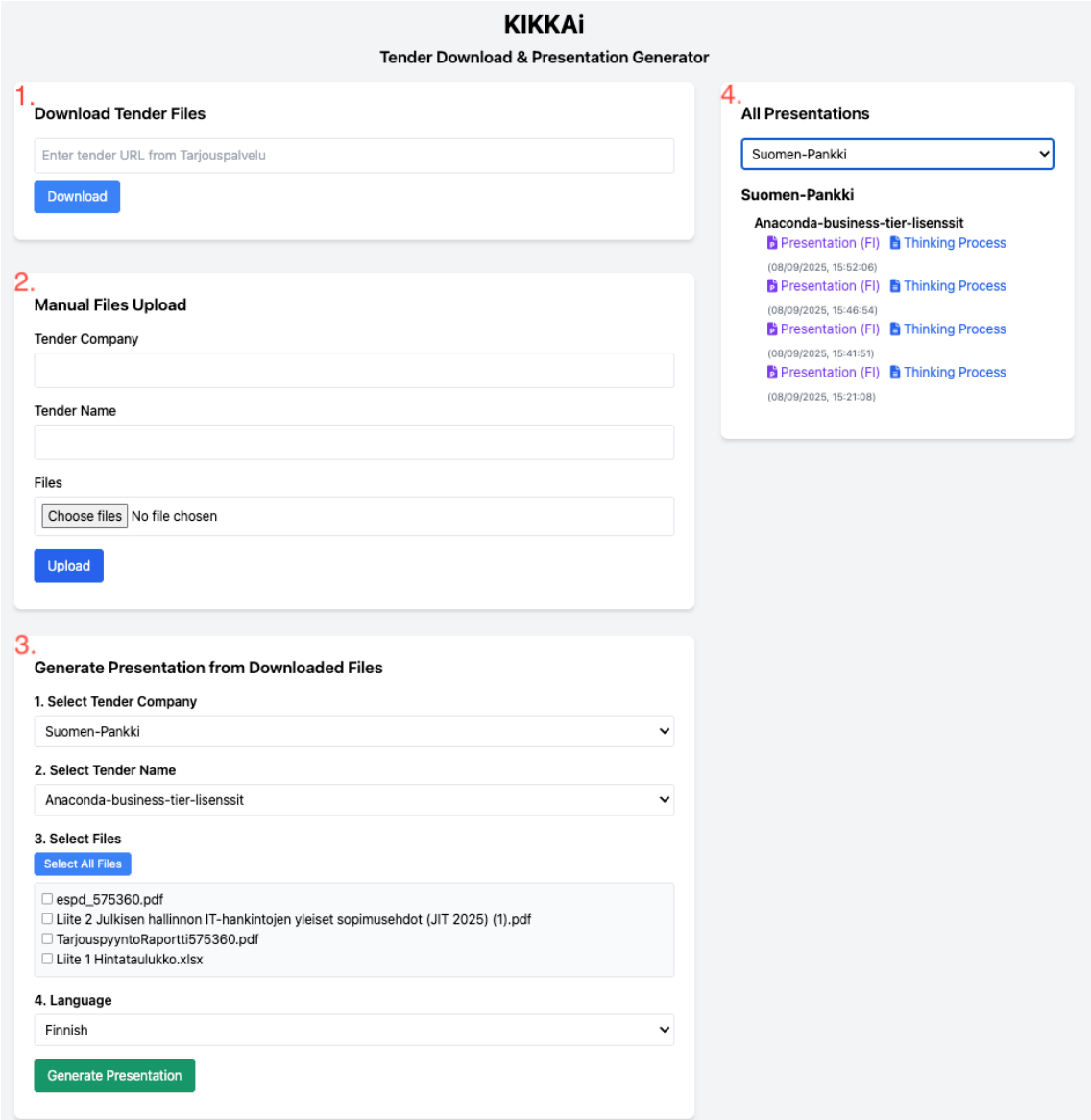
4 LOPPUTUOTOS

4.1 Tuotoksen kuvaus

Lopputuotteena syntyi tekoälyavusteinen PowerPoint-esitysten luomiseen tarkoitettu työkalu tai sovellus. Työkalu on suunniteltu tuottamaan kickoff-esityksiä tarjousprosessin alkuvaiheeseen. Käytännössä sovellus luo materiaalin, jonka avulla asiantuntijaryhmälle voidaan esitellä uusi projekti pääpiirteittäin. Sovelluksen selainpohjainen käyttöliittymä rakentuu käyttäjän näkökulmasta neljästä toiminnosta, jotka ovat:

1. Tiedostojen lataus URL:n perusteella Tarjouspalvelusta. Käyttäjä voi hakea tarjousdokumentit suoraan palvelusta syöttämällä tarjouksen URL-osoitteen.
2. Mahdollinen manuaalinen tiedostojen lataus esimerkiksi TED:stä tai jo kiinni menneestä kilpailutuksesta.
3. Esityksen generointi ladatuista tiedostoista, jossa voi vielä määritellä, mitä tiedostoja käytetään esityksen generointiin. Lisäksi käyttäjä voi valita kielen, millä esitys tuotetaan. Tämän jälkeen sovellus generoi PowerPoint-esityksen automaattisesti. Käyttäjä pystyy myös esikatselemaan dokumentteja suoraan sovelluksen käyttöliittymässä. Esikatselussa olevat dokumentit avautuvat toiselle välilehdelle.
4. Näkymä kaikkiin tuotettuihin PowerPoint-esityksiin sekä niiden Markdown-muodossa oleviin ajatteluprosesseihin. Lisäksi saatavilla on tieto siitä, milloin esitys on generoitu sekä millä kielellä se on tuotettu.

Kuvassa 11 näkyy sovelluksen käyttöliittymä numeroituna neljään pääosioon, jotka ovat kuvattu edellisessä luettelossa.



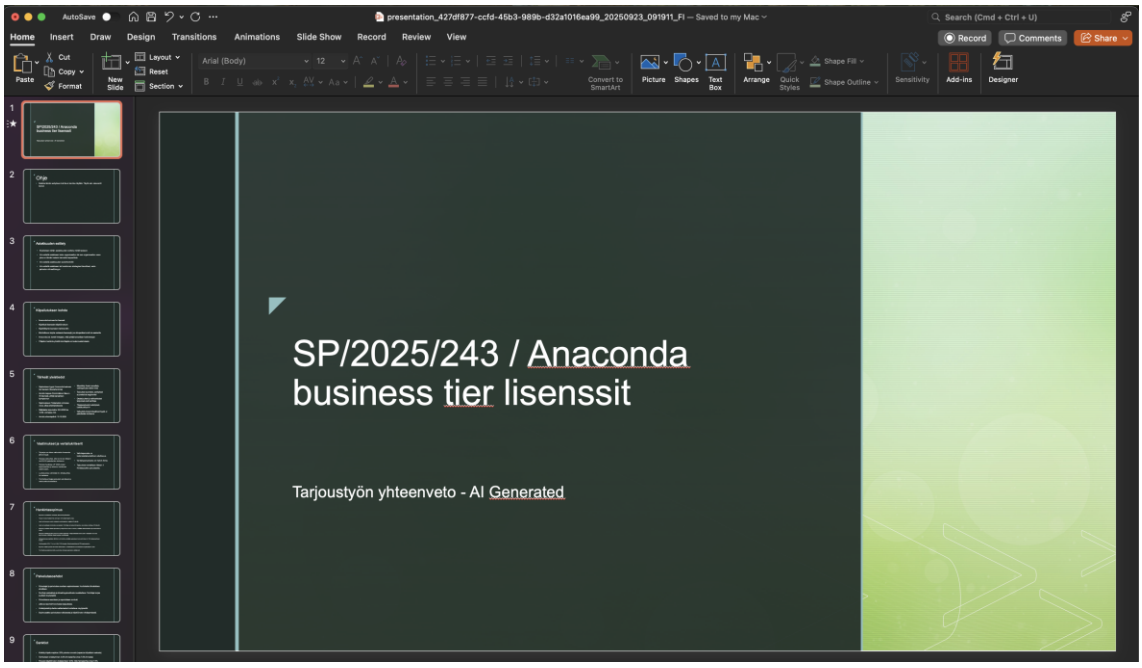
Kuva 11 Sovelluksen käyttöliittymä, josta näkyvät kaikki ominaisuudet, mitkä sovelluksen käyttäjällä on käytettävissä. (Pyry Ikonen 2025)

Generatiivinen tekoäly ei näy käyttäjälle muuten kuin PowerPoint-esitysten ensimmäisen sivun alaotsikossa, jossa mainitaan, että esitys on tekoälyn generoima. Esityksen generoinnin aikana käyttäjälle näytetään tieto prosessin käynnissä olosta sekä siihen kuluneesta ajasta. Aiemmissa versioissa käyttäjälle näytettiin myös taustaprosessien eteneminen vaiheittain, mutta tämä todettiin ylimääräiseksi tiedoksi, jolla ei ollut käyttäjälle lisäarvoa.

Seuraavaksi esitellään kuvissa 12, 13, 14 ja 15 näkyvää työkalulla tuotettua PowerPoint-esitystä, joka valittiin satunnaisesti ajankohtaisista IT-alan kilpailutuksista viikolla 29 opinnäytetyötä kirjoittaessa. Tarkoituksena on antaa näkemys siitä, minkälaista tietoa kielimallilla voi tuottaa ja kuinka tuotetun tekstin pystyy

asettelemaan PowerPoint-pohjaan. Esimerkkiin valikoitui Suomen Pankin Anaconda business tier lisenssien kilpailutus. Kokonaisuudessa työkalu tuottaa tekstin ja asettelun PowerPoint-esityksen kahdeksalle eri dialle, mutta koska kuvat vievät paljon tilaa näistä esitellään neljä.

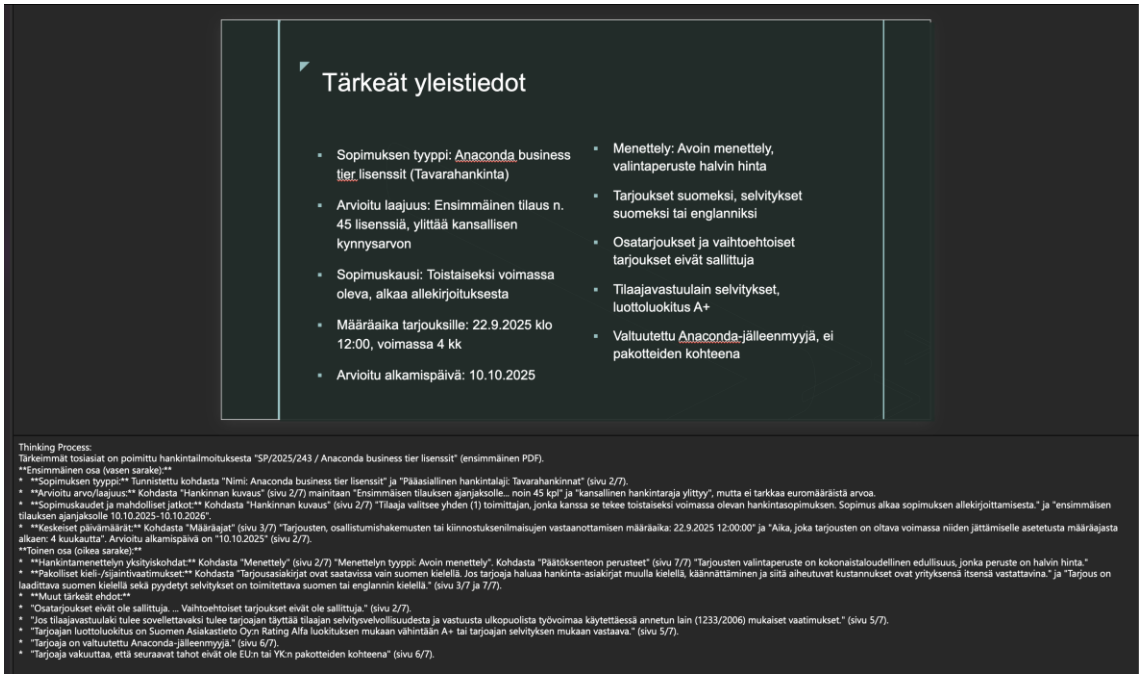
Kuvassa 12 näkyy tuotetun PowerPoint-esityksen etusivu, jolla sijaitsee Otsikko sekä alaotsikko, Otsikon tekstin poimitaan joko Tarjouspalvelun sivuilta tai Tarjouspyyntö-tiedostosta, jolloin sen tulisi aina noudattaa samoja niemeämiskäytänteitä.



Kuva 12 Sovelluksella tuotetun esimerkkiesityksen Suomen Pankin Anaconda lisenssien kilpailutuksen PowerPoint-esityksen etusivu. (Pyry Ikonen 2025)

Kuvassa 13 esitetään esityksen viides dia, jonka otsikkona on "Tärkeät yleistiedot". Dia sisältää keskeisiä hankintaan liittyviä tietoja, kuten sopimuksen tyyppi, arvioitu laajuus, sopimuskausi, määräaika tarjouksille, menettelytapa ja valintaperuste. Nämä tiedot ovat olennaisia hankintaprosessin ymmärtämiseksi, mutta kielimallille annettu kehote ei itsessään määrittele, mitkä näistä tiedoista ovat pro-

jektin kannalta kaikkein tärkeimpiä, vaan malli valitsee itse tärkeimmät tiedot. Tietysti mikäli haluaisi jotain painotuksia tehdä, niitä pystyisi myös sisällyttämään kehotteeseen.



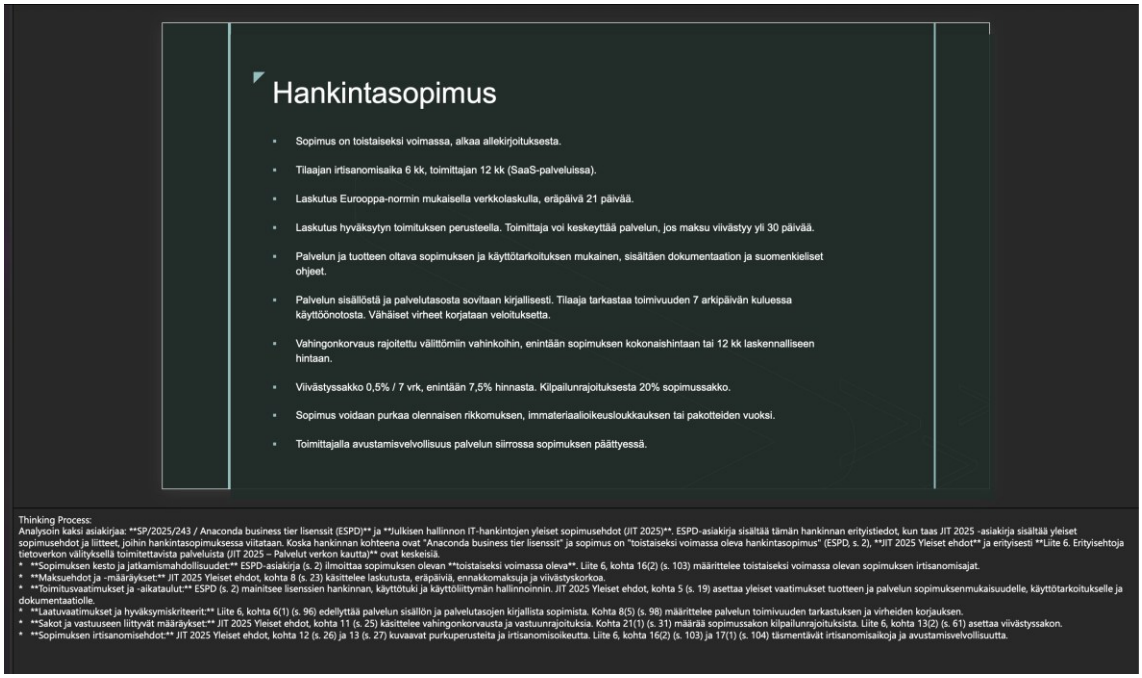
Kuva 13 Sovelluksella tuotetun esimerkkiesityksen Suomen Pankin Anaconda lisenssien kilpailutuksen ”Tärkeät yleistiedot” dia. (Pyry Ikonen 2025)

Toisin kuin edellisessä ”Tärkeät yleistiedot” -diassa sanktiot dian kehotteessa on tarkasti määritelty, minkälaisia tietoja tähän dian erityisesti halutaan hakea. Tämän eron pystyy myös näkemään ajatteluprosessin erilaisesta tekstityylistä,

jossa etsitään mainintoja erityisesti sopimussakoista, hyvityksistä ja vastuunrajoituksista.

Kuva 14 Sovelluksella tuotetun esimerkkiesityksen Suomen Pankin Anaconda lisenssien kilpailutuksen ”Sanktiot” dia. (Pyry Ikonen 2025)

Kuvassa 15 esitellään ”Hankintasopimus” -diaa. Valitsin dian tarkoituksella, koska siinä tiivistyy keskeinen ongelma, jossa tekstiä on paljon ja se pitäisi saada järkevästi esitettyä diassa. Sovelluksessa on apufunktio, joka pienentää fonttikokoa automaattisesti, kun dian merkkimäärä lisääntyy tarpeeksi.



Kuva 15 Sovelluksella tuotetun esimerkkiesityksen Suomen Pankin Anaconda lisenssien kilpailutus "Hankintasopimus" dia. (Pyry Ikonen 2025)

Ongelmana tässä ratkaisussa on se, että fonttikoko saattaa pienentyä liikaa, niin kuin kuvasta näkyy, jolloin tekstistä tulee vaikealukuisempaa. Vaihtoehtoisena ratkaisuna olisi rajoittaa pituutta jo kehotteessa, mutta silloin sisällön laatu saattaa kärsiä tai olennaisia asioita saattaa jäädä vastauksesta pois, koska kielimalli tiivistää vastausta liikaa.

Kickoff-esitysten tuottamiseen sovelluksella kului aineiston laajuuden mukaan muutamia minuutteja. Tämä on ajallisesti murto-osa siitä työmäärästä, joka asiantuntijalla kuluisi vastaavan aineiston lukemiseen ja manuaaliseen tiivistämiseen. Vaikka lopulliseen ajankäyttöön vaikuttaa myös asiantuntijan tekemä tarkistustyö, osoittaa luonnosvaiheen automaatio merkittävää potentiaalia tarjousprosessin tehostamiseen.

4.2 Tuotoksen hyödynnettävyys ja käyttötarkoitus

Ensisijaisesti sovellus on suunniteltu vastaamaan toimeksiantajan Public Business -tiimin tarpeisiin. Vaikka sovellusta ei projektin puitteissa ehditty vielä jalkauttamaan laajaan tuotantokäyttöön, kehitetty prototyyppi osoittaa ratkaisu manuaalisen työn tehostamiseen. Täysimääräisesti käyttöön otettuna sovellus tarjoaisi asiantuntijalle automaattisesti generoidun luonnoksen, joka toimii työn pohjana. Tämä siirtäisi työn painopisteen tekstin tiivistämisestä ja kopioimisesta sen asiasisällön jalostamiseen.

Kehitettyä sovellusta on mahdollista hyödyntää koko julkisten kilpailutusten kentällä, mikäli kielimallille annettavia kehotteita ja mallin asetuksia muokataan sopiviksi eri toimialojen erityispiirteisiin. Nykyisenkin kielimallin asetuksilla pystyy tuottamaan materiaalia joistakin toimialoista, jotka ovat lähellä IT-alaa, mutta laatu kärsii merkittävästi, jos kielimallia pyydetään tuottamaan kickoff-esitys kaukana nykyisestä toimialasta olevasta tarjouspyynnöstä.

Sovellusta on siis mahdollista hyödyntää paitsi julkisella, että yksityisellä sektorilla, erityisesti organisaatioissa, joissa tarjousprosessit sisältävät suuria määriä

dokumentteja. Sovelluksen laajat käyttömahdollisuudet tekevät siitä potentiaalisen ratkaisun monenlaisiin tarjouskäytäntöihin, kunhan kielimallin ohjaus mukautetaan toimialakohtaisesti.

Mielestäni sovellus osoittaa, että kielimallien todellinen hyöty erityisesti sovelluskehityksessä saavutetaan silloin, kun ne integroidaan osaksi sovellusta ja laadukasta dataa. Tämä dataa, koodia ja generatiivista tekoälyä yhdistävä lähestymistapa mahdollistaa sellaisten sovellusten rakentamisen, jotka pystyvät aidosti automatisoimaan ja tehostamaan monimutkaisia prosesseja.

5 TULOKSET JA ARVIOINTI

5.1 Kehittämistyön tulokset

Opinnäytetyön tuloksena syntyi toimiva prototyyppi, joka pystyy hakemaan tarjouspyyntödokumentteja URL-osoitteen perusteella Tarjouspalvelusta ja generoimaan niistä kielimallia hyödyntäen kickoff-esityksen PowerPoint-muodossa.

Onnistumisena voidaan pitää sovellukseen kehitettyä ajatteluprosessin taltiointia. Tämä ominaisuus lisää kielimallin toiminnan läpinäkyvyyttä, sillä se osoittaa käyttäjälle suoraan, mihin alkuperäisten tarjouspyyntödokumenttien kohtiin generoidut diat perustuvat. Tämä lisää sisällön tarkasteltavuutta ja helpottaa asiantuntijan tekemää lopullista laadunvarmistusta, mutta ei yksinään takaa tuotetun sisällön virheettömyyttä.

PowerPoint-esitysten generoimiseen kuluu aikaa tyypillisesti vain muutamia minuutteja riippuen käsiteltävän aineiston laajuudesta. Kehitys- ja testivaiheessa saadun kokemuksen perusteella esityksen tuottaminen vaikutti ajallisesti merkittävästi nopeammalta kuin täysin manuaalinen työ, mutta koska sovellusta ei ehditty ottaa tuotantokäyttöön, ei ajansäästöä verrattuna täysin manuaaliseen työhön voitu tässä työssä mitata luotettavasti.

5.2 Tuotoksen arviointi kohdeorganisaation näkökulmasta

Keskeinen hyöty kohdeorganisaation näkökulmasta on rutiininomaisen työn vähentäminen. Asiantuntija, jonka tehtävänä on kickoff-esityksen tekeminen, saa valmiin esitysluonnoksen tarkistettavaksi ja voi keskittyä sen viimeistelyyn sen sijaan, että koko sisältö tuotettaisiin alusta alkaen manuaalisesti tarjouspyyntödokumenteista.

Projektin aikana toimeksiantajan asiantuntijoilta saadun palautteiden perusteella tuotettu sisältö vaikutti laadukkaalta ja käyttökelpoiselta. Laajempaa laadullista testausta useilla eri tarjouspyynnöillä ei ehditty projektin puitteissa suorittamaan, joten satunnaisten laatuongelmien mahdollisuutta ei voi kuitenkaan sulkea pois.

Sovelluksen käytettävyyttä ei ehditty arvioimaan toimeksiantajan loppukäyttäjien näkökulmasta, sillä sovelluksen siirto testiympäristöön jäi kesken johtuen siitä, että ympäristön tilaaminen ja tarvittavien oikeuksien saaminen vei ennakoitua enemmän aikaa.

5.3 Itsearviointi

Projektin kokonaisaikataulu venyi alkuperäisestä suunnitelmasta kirjoitusvaiheen haasteiden takia. Itse kehitysvaiheessa eteneminen oli aika tasaista. Olen lopputulokseen tyytyväinen, koska valmis sovellus suoriutuu pääasiallisesta käyttötarkoituksestaan onnistuneesti.

Merkittävin haaste projektin aika oli laaja uusien teknologioiden ja kirjastojen omaksuminen. Tämän lisäksi aikaa kului merkittävästi julkiseen kilpailutusprosessiin tutumisessa sekä sen toimintaympäristöön perehtyessä. Jotta sovelluksen logiikka ja kielimallin kehotteet osattiin rakentaa oikein, oli ensin ymmärrettävä tarjouspyyntöjen rakenne ja keskeinen terminologia.

5.4 Työn tavoitteiden saavuttaminen

Työn tavoitteena oli vähentää myyntiprosessin alkuvaiheen työmäärää. Tekninen tavoite voidaan katsoa saavutetuksi, sillä kehitetty sovellus kykenee automatisoimaan tarjouspyyntödokumenttien haun, sisällön käsittelyn ja kickoff-esityksen tuottamisen.

Tarkempaa arviota tavoitteen saavuttamisesta on kuitenkin vaikea tehdä, koska emme ehtineet arvioimaan tarkemmin, kuinka paljon kielimallilla tuotetun kickoff-esityksen läpikäymiseen menisi aikaa verrattuna siihen, että esitysmateriaali tuotettaisiin kokonaan itse.

Sovelluksen integroiminen kohdeorganisaation palveluihin jäi toteuttamatta aikataulu ja käyttöoikeuksiin liittyvien teknisten esteiden vuoksi. Integraation puuttuminen vaikeutti sovelluksen testausta aidossa loppukäyttäjäympäristössä, eikä kohderyhmä päässyt arvioimaan työkalua täysin osana normaalia työnkulkuaan.

6 POHDINTA

6.1 Opinnäytetyöprosessin reflektointi

Opinnäytetyön aikana pääsin kommunikoimaan laajasti koko julkisen puolen myyntitiimin kanssa ja se auttoi ymmärrystäni, siitä minkälaisia kehitysprojektit monipuolisissa tiimeissä voivat olla.

Projektin koodausvaihe eteni pääosin aikataulussa. Opinnäytetyön kirjoitusvaiheessa aikataulusta poikettiin jonkin verran. Haasteita aiheutti muun muassa toimeksiantajan järjestelmien ja prosessien mukaan toimiminen, mikä on kuitenkin tyypillistä yritysympäristössä.

Teknisiä haasteita toi useiden ennestään vieraiden kirjastojen käyttöönotto. Tiedonhaku ja ajantasaisen tiedon löytäminen rajapinnoista ja niiden muutoksista tuotti ajoittain ongelmia, sillä kielimallit ja niiden rajapinnat kehittävät tällä hetkellä erittäin nopeasti.

Ammatillisesti työ tarjosi merkittävää kokemusta uusien teknologioiden käytöstä. Redis, ARQ- ja FastAPI-kirjastot olivat kaikki uusia tuttavuuksia. Kielimallien rajapintoja olin ennen työtä kokeillut lähinnä mielenkiinnosta, mutta työssä kielimalli rakennettiin osaksi isompaa sovellusta. Opin myös julkisten kilpailutuksien eri prosesseista niin yleisesti kuin toimeksiantajan tasolla. Lisäksi pääsin tutustumaan julkisten kilpailutuksien avuksi rakennettuihin järjestelmiin, kuten Hilmaan ja Tarjouspalveluun.

6.2 Jatkomahdollisuudet ja kehitysehdotukset

Sovellusta on mahdollista kehittää eteenpäin teknisin ratkaisun sekä konseptitasolla. Mielestäni merkittävin parannus sovelluksen ominaisuuksiin käyttäjän näkökulmasta olisi jonkinlaisen automaattisen testauksen kehittäminen nimenomaan kielimallin luoman ajatteluprosessin eli lähdemateriaalin ja alkuperäisten tarjousdokumenttien välille. Silloin vältettäisiin ongelma, jossa työntekijän aikaa siirretään itse PowerPoint-esityksen tekemisestä sen tarkistamiseen.

Toinen käyttäjää erityisesti helpottava parannus olisi python-pptx kirjaston parempi käyttö ja hyödyntäminen. Kirjaston oma dokumentaatio oli aika vaikeasti luettava, mutta aikaisemmin esitellyssä PPTAgent -projektissa kirjaston käyttö oli toteutettu selkeämmin sekä sen ominaisuuksia paremmin hyödyntäen. Tekstin tiivistäminen dioihin olisi varmasti tehtävissä jollakin paremmalla tavalla, jolloin vältyttäisiin suuremmilta esityksen ulkomuodon muokkauksilta.

Teknisestä näkökulmasta ajatellen sovelluksen nykyinen paikallinen levytila tulisi korvata pilvipohjaisella varastolla (esimerkiksi AWS S3, Google Cloud Storage tai Azure Blob Storage). Lisäksi sovelluksen yhdistäminen osaksi toimeksiantajan dokumenttienhallintajärjestelmää helpottaisi ja mahdollistaisi paremman sovelluksen testaamisen sen loppukäyttäjillä.

Nykyinen riippuvuus yksittäisestä kielimallista muodostaa toimittajariippuvuuden. Järjestelmään tulisi lisätä kielimallien välityspalvelin, jotta mallien vaihtaminen ei vaatisi koodin editoimista tulevaisuudessa. Tämä olisi toteutettavissa esimerkiksi LiteLLM välityspalvelinta käyttämällä, joka on tällä hetkellä yksi suosituimmista ja laajimmin käytetyistä kielimallien välityspalvelimista.

6.3 Hallusinaatiot, luotettavuus, eettisyys

Yksi suurten kielimallien perimmäinen ominaisuus on niiden taipumus hallusinaatioihin, eli malli saattaa tuottaa väärää tietoa ja esittää sen oikeana faktana. Sovelluksessa suurin riski tähän on silloin, kun kielimallille annetussa kehotteessa on kysymyksiä, joihin ei löydy vastausta tarjouspyyntödokumenteista. Tällöin malli saattaa pyrkiä täyttämään tietovajeen sisäisen koulutusdatansa perusteella sen sijaan, että se ilmoittaisi tiedon puuttumisesta. Useammin sovellusta käyttäessä nähtiin kuitenkin vastauksia kuten: ”Tarjousdokumentista ei löydy vastausta kysymykseen x.” Eikä kielimalli lähtenyt täyttämään Powerpoint diojen kysymyksiin vastauksia hallusinoidulla tekstillä.

Hallusinaatioiden järjestelmällinen testaaminen on haastavaa, sillä kielimallit ovat luonteeltaan epädeterministisiä. Tarkoittaen sitä, että sama syöte voi tuottaa eri kerroilla hieman eriävän vastauksen. Hallusinaatioiden testaamiseen ei ole olemassa yhtä yksinkertaista menetelmää. Periaatteessa lähdedokumentteja voisi

käyttää referenssitietona, mutta koska tekoälyn tehtävänä on nimenomaan tiivistää laajoja dokumenttikokonaisuuksia, ei suoraa tekstuaalista vertailua voida tehdä. Kehittyneimmissä prosesseissa voitaisiin käyttää ”LLM-as-a-judge” -menetelmää jossa toinen kielimalli arvioi tuotetun vastauksen ja lähdedokumentin välistä vastaavuutta. Toisaalta tässäkin tavassa palataan taas takaisin samaan ongelmaan luotettavuuden ja hallusinaatioiden osalta, joten toimivin ratkaisu taitaa vielä olevan hybridi malli, jossa ihminen ja esimerkiksi toinen kielimalli molemmat valvovat tekstin luotettavuutta.

Eettisestä näkökulmasta ajatellen tarjouspyyntö dokumenttien syöttäminen ulkoisiin rajapintoihin pitäisi olla hyväksyttävää, sillä on kyse ainakin niin sanotusti puoli julkisista dokumenteista, sillä kuka tahansa voi rekisteröityä kyseisiin palveluihin, kuten Hilmaan tai Tarjouspalveluun ja ladata materiaalit vapaasti.

7 YHTEENVETO

Opinnäytetyön lähtökohtana oli tarve tehostaa julkisten kilpailutusten tarjousprosessin alkuvaihetta. Työn tavoitteena oli suunnitella ja toteuttaa generatiivista tekoälyä hyödyntävä sovellus, joka automatisoi kickoff-esitysten luomisen ja vapauttaa asiantuntijoiden aikaa vaativimpiin tehtäviin.

Työn tuloksena syntyi ohjelmistosovellus tarjouspyyntödokumenttien automaattiseen käsittelyyn ja PowerPoint-esitysten generointiin. Sovelluksen palvelinpuoli tuotettiin Pythonilla FastAPI-kirjaston päälle. Tarjousdokumenttien automaattinen haku toteutettiin Playwright-selainautomaatiolla. Dokumenttien sisältö analysoitiin ja esityssisältö generoitiin Googlen Gemini-kielimallin avulla. Varsinainen PowerPoint-esitysten tekninen muodostus tehtiin python-pptx-kirjastolla. Pitkäkestoiset lataus- ja generointiprosessit toteutettiin käyttämällä Redis- ja ARQ-kirjastoja, mikä mahdollisti sujuvan käytön useiden rinnakkaisten prosessien aikana.

Kehitetty ratkaisu osoittautui toimivaksi ja vastasi asetettuja tavoitteita. Sovellus kykenee tiivistämään laajan aineiston esitysmuotoon muutamassa minuutissa. Erityisenä onnistumisena voidaan pitää kielimallin ajatteluprosessin taltiointia, mikä parantaa kielimallin tuottaman sisällön läpinäkyvyyttä. Vaikka sovelluksen täysimääräinen integraatio organisaation järjestelmiin jäi jatkokehityksen varaan työ osoittaa, että generatiivista tekoälyä hyödyntäen on mahdollista tehostaa opinnäytetyössä käsiteltyjä tarjousprosessin alkuvaiheita.

LÄHTEET

Abrahamsson, P., Salo, O., Ronkainen, J. & Warsta, J. 2002.
Agile software development methods: Review and analysis.
arXiv. Verkkosivu.
Viitattu 4.12.2025. https://arxiv.org/pdf/1709.08439

Amazon Web Services. n.d. Amazon Bedrock. Verkkosivu.
Viitattu 4.12.2025. https://aws.amazon.com/bedrock/

Anthropic. n.d. Model overview. Verkkosivu. Viitattu 28.8.2025.
https://platform.claude.com/docs/en/about-claude/models/overview

ARQ. n.d. ARQ: Job queues in python with asyncio and redis. Verkkosivu.
Viitattu 1.12.2025. https://arq-docs.helpmanual.io/

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I. & Amodei, D. 2020.
Language models are few-shot learners. arXiv. Verkkosivu.
Viitattu 28.8.2025. https://arxiv.org/abs/2005.14165

Cloudia. n.d. Cloudia Tarjouspalvelu. Verkkosivu.
Viitattu 1.9.2025. https://tarjouspalvelu.fi/Default/Index

Cursor. n.d.-a. Get Started, Concepts. Verkkosivu.
Viitattu 23.2.2026.
https://cursor.com/docs/get-started/concepts

Cursor. n.d.-b. VS Code Migration. Verkkosivu.
Viitattu 23.2.2026.
https://cursor.com/docs/configuration/migrations/vscode

Docker. n.d. Get Started. Verkkosivu.
Viitattu 2.9.2025. https://docs.docker.com/get-started/docker-overview/

FastAPI. n.d. FastAPI Documentation. Verkkosivu.
Viitattu 2.9.2025. https://fastapi.tiangolo.com/

Gamma. n.d. Gamma. Verkkosivu.
Viitattu 2.9.2025. https://gamma.app/products/presentations

GitHub. n.d.-a. About GitHub and Git. Verkkosivu. Viitattu 8.9.2025.
https://docs.github.com/en/get-started/start-your-journey/about-github-and-git

GitHub. n.d.-b. GitHub Copilot documentation. Verkkosivu.
Viitattu 2.9.2025. https://docs.github.com/en/copilot/get-started/best-practices

Google. n.d. Vertex AI. Verkkosivu.

Viitattu 4.12.2025. https://cloud.google.com/vertex-ai

Google. 2025a. Gemini API: Models. Verkkosivu.
Viitattu 8.9.2025. https://ai.google.dev/gemini-api/docs/models

Google. 2025b. Gemini API: Thinking. Verkkosivu.
Viitattu 8.9.2025. https://ai.google.dev/gemini-api/docs/thinking

Google. 2025c. Massive regression: detailed Gemini thinking process vanished from AI Studio. Google AI Developer Forum. Verkkosivu.
Viitattu 2.9.2025. https://discuss.ai.google.dev/t/massive-regression-detailed-gemini-thinking-process-vanished-from-ai-studio/83916/102

Google. 2025d. Prompting strategies: Include few-shot examples. Verkkosivu. Viitattu 1.9.2025.
https://cloud.google.com/vertex-ai/generative-ai/docs/learn/prompts/few-shot-examples

Google. 2025e. Vertex AI SDK for Python: GenerationConfig. Verkkosivu. Viitattu 2.9.2025.
https://cloud.google.com/python/docs/reference/aiplatform/latest/google.cloud.aiplatform_v1.types.GenerationConfig

Google. 2025f. Vertex AI SDK for Python: GenerateContentRequest. Verkkosivu. Viitattu 2.9.2025.
https://cloud.google.com/python/docs/reference/aiplatform/latest/google.cloud.aiplatform_v1.types.GenerateContentRequest

Google. 2025g. Vertex AI SDK for Python: GenerateContentResponse. Verkkosivu. Viitattu 2.9.2025.
https://cloud.google.com/python/docs/reference/aiplatform/latest/google.cloud.aiplatform_v1.types.GenerateContentResponse

Google. 2025h. Generative Ai on Vertex AI documentation – Experiment with parameter values. Verkkosivu.
Viitattu 19.09.2025.
https://docs.cloud.google.com/vertex-ai/generative-ai/docs/learn/prompts/adjust-parameter-values

Hilma. n.d. Julkisten hankintojen palvelu. Verkkosivu.
Viitattu 1.9.2025. https://www.hankintailmoitukset.fi/fi/

LiteLLM. 2025. LiteLLM Documentation. Verkkosivu.
Viitattu 4.12.2025. https://docs.litellm.ai/docs/

Microsoft. n.d. Visual Studio Code. Verkkosivu.
Viitattu 1.12.2025. https://code.visualstudio.com/

Microsoft. n.d. Azure AI Services. Verkkosivu.
Viitattu 4.12.2025. https://azure.microsoft.com/en-us/solutions/ai

Minaee, S., Mikolov, T., Nikzad, N., Chenaghlu, M., Socher, R., Amatriain, X. & Gao, J. 2024. Large Language Models: A Survey. arXiv. Verkkosivu. Viitattu 4.12.2025. https://arxiv.org/abs/2402.06196

OpenAI.12.9.2024. Learning to reason with LLMs. Verkkosivu. Viitattu 8.12.2025. https://openai.com/index/learning-to-reason-with-llms

OpenAI. n.d. OpenAI Platform. Verkkosivu. Viitattu 2.9.2025. https://platform.openai.com/docs/overview
Pipenv. n.d. Pipenv: Python Dev Workflow for Humans. Verkkosivu. Viitattu 2.9.2025. https://pipenv.pypa.io/en/latest/

PopAi. n.d. PopAi. Verkkosivu. Viitattu 2.9.2025. https://www.popai.pro/

PowerDrill. n.d. PowerDrill. Verkkosivu. Viitattu 2.9.2025. https://powerdrill.ai/

python-pptx. n.d. python-pptx Documentation. Verkkosivu. Viitattu 1.12.2025. https://python-pptx.readthedocs.io/en/latest/

Tailwind Labs. n.d. Tailwind CSS Documentation. Verkkosivu. Viitattu 2.9.2025. https://tailwindcss.com/

Yuheng Yang, Wenjia Jiang, Yang Wang, Yiwei Wang, Chi Zhang. 2025. Auto-Slides: An Interactive Multi-Agent System for Creating and Customizing Research Presentations. arXiv. Verkkosivu. Viitattu 4.12.2025. https://arxiv.org/abs/2509.11062

Windsurf. n.d. Recommended Plugins. Verkkosivu. 23.2.2026. https://docs.windsurf.com/windsurf/recommended-plugins

xAI. n.d. Grok. Verkkosivu. Viitattu 2.9.2025. https://x.ai/api

Zheng, H., Guan, X., Kong, H., Zheng, J., Zhou, W., Lin, H., Lu, Y., He, B., Han, X. & Sun, L. 2025. PPTAgent: Generating and evaluating presentations from large language models. arXiv. Verkkosivu. Viitattu 1.12.2025. https://arxiv.org/abs/2501.03936

Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., Du, Y., Yang, C., Chen, Y., Chen, Z., Jiang, J., Ren, R., Li, Y., Tang, X., Liu, Z., Liu, P., Nie, J. & Wen, J. 2023. A Survey of Large Language Models. arXiv. Verkkosivu. Viitattu 4.12.2025. https://arxiv.org/abs/2303.18223

LIITTEET