BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Dalam kegiatan kerja magang di perusahaan CV. Inovasi Artificial Intelligence Indonesia (AI.DECE), pelaksanaan kerja magang ditempatkan pada posisi *AI Engineer Intern* di bawah *Technical Team Manager*. Kegiatan magang ini memiliki tanggung jawab dalam pengembangan salah satu produk perusahaan, yaitu Platform *AI Mastering Document*. Alasan mengapa produk tersebut merupakan sebuah platform adalah karena platform merupakan suatu fondasi yang menyediakan rangkaian layanan atau fitur bagi sistem lain untuk berjalan dan berkembang yaitu sistem kecerdasan buatannya, dimana hal ini sejalan dengan fitur yang dimiliki oleh Platform *AI Mastering Document*, yaitu *AI Smart Chatbot*, *AI Smart Searching*, dan *AI Grouping*. Platform ini dapat diintegrasikan ke sistem perusahaan lainnya dengan pilihan fitur sesuai preferensi perusahaan tersebut sebagai klien perusahaan AI.DECE. Berbeda dengan sistem yang titik fokusnya lebih spesifik dan tidak bersifat lebih luas, serta tidak dapat menjadi ekosistem yang mendukung interaksi antara pengembang, pengguna, dan aplikasi lain [56]. Ekosistem yang dimaksud adalah ekosistem bagi pengguna atau klien untuk dapat memilih fitur yang diinginkan untuk diimplementasikan ke sistem perusahaannya [57]. Proses pengembangan ini dilakukan bersama dengan supervisor yang merupakan *Founder* & CEO perusahaan AI.DECE sendiri beserta dengan manajer divisi *Technical Team*.

Arahan dan bimbingan selama pelaksanaan magang diberikan oleh Bapak Ivan Handryks Sitanaya selaku *Founder* & CEO perusahaan AI.DECE. Seluruh progres pekerjaan dilaporkan dalam pertemuan rutin yang dilaksanakan setiap hari Rabu dan Jumat pukul 10:00 WIB hingga 12:00 WIB. Pada setiap pertemuan diisi dengan pemaparan progres tugas, penerimaan masukan dan penugasan baru, serta diskusi lanjutan mengenai pengembangan produk berupa teknologi AI pada perusahaan AI.DECE.

Koordinasi dalam tim dilakukan secara luring maupun daring melalui media komunikasi seperti WhatsApp dan Microsoft Teams. Selama masa magang diperoleh kesempatan untuk bekerja secara langsung dengan anggota tim, belajar dari pengalaman mereka, serta berkontribusi dalam proyek-proyek yang sedang

dikembangkan oleh perusahaan.

Selain tanggung jawab utama dalam pengembangan Platform *AI Mastering Document*, pelaksanaan kerja magang juga terlibat dalam beberapa proyek lain selama masa magang. Kontribusi tersebut antara lain pengembangan situs web untuk proyek Tokyo Consulting dan situs web app.grobotik.com menggunakan *framework* NextJS, Tailwind CSS, dan Native PHP. Meskipun tugas-tugas ini berada di luar lingkup posisi *AI Engineer Intern*, keterlibatan dalam proyek-proyek tersebut merupakan bagian dari tanggung jawab pengembangan diri sekaligus kontribusi bagi kemajuan AI.DECE dalam mengelola berbagai produk klien.

3.2 Tugas yang Dilakukan

Tugas-tugas yang dilakukan selama proses kegiatan magang pada perusahaan AI.DECE sebagai *AI Engineer Intern* mencakup pengembangan platform teknologi AI yang merupakan salah satu produk utama dari perusahaan. Produk ini dinamakan *AI Mastering Document* dengan fokus pengembangannya terletak pada tiga fitur unggulan yaitu *AI Smart Chatbot*, *AI Smart Searching*, dan *AI Grouping*, serta tampilan antarmuka pengguna (UI) yang disajikan dalam bentuk situs web, dimana sebelumnya hanya berupa program Python sederhana dengan *output* yang hanya dapat dilihat di terminal atau hasil respons API menggunakan Postman. Adapun pengembangan ini dibangun menggunakan sejumlah teknologi inti yang saling terintegrasi untuk memastikan kinerja dan *output* yang terbukti efektif, antara lain Flask, LangChain, Weaviate ataupun ChromaDB sebagai *vector database*-nya, *Large Language Model* (LLM) dari OpenAI yang menggunakan model `gpt-4.1-mini`, *embeddings* yang menggunakan OpenAIEmbeddings, serta *Retrieval-Augmented Generation* (RAG) [30], [31]. Dengan kata lain, alasan dilakukan pengembangan pada platform ini adalah kurangnya fitur untuk mendukung proses platform tersebut yang akan diintegrasikan kepada perusahaan klien sebagaimana hal ini telah tercantum dalam Lampiran 6 yang berisikan transkrip wawancara dengan supervisor sebagai narasumbernya.

Selama kegiatan magang berlangsung, berbagai perangkat lunak pendukung digunakan untuk membantu proses perancangan dan pengembangan sistem secara optimal. Salah satu perangkat utama yang digunakan adalah *Integrated Development Environment* (IDE), yaitu *Visual Studio Code* (VS Code) sebagai *code editor*. Selain itu, Git dan GitHub digunakan sebagai sistem *version control* dalam pengelolaan *codebase* proyek. Pengelolaan cabang (*branch*) dilakukan dengan

penamaan `origin/jackson/` untuk setiap perubahan yang diterapkan selama proses pemrograman.

Pada tahap awal, perusahaan AI.DECE memberikan pengenalan terhadap berbagai proyek terdahulu serta struktur tim yang sedang bertugas. Pada fase ini, proses pembelajaran dilakukan secara mandiri mengenai berbagai *framework*, *library*, dan perangkat lunak (*tools*) yang digunakan dalam pengembangan, serta dipersilakan untuk berdiskusi dengan supervisor terkait program yang dilakukan. Selanjutnya, penugasan awal yang diberikan adalah berupa pengembangan sebuah *chatbot* dasar. *Chatbot* tersebut masih bersifat sederhana, karena hanya mampu menerima satu *file* PDF sebagai sumber pengetahuan dari *chatbot* itu sendiri dan hanya menerima satu *query* sebagai pertanyaan dari pengguna terhadap *file* tersebut yang diikuti dengan hasil jawaban *chatbot* atas pertanyaan tersebut.

Pengembangan awal dilakukan menggunakan salah satu *framework* Python, yaitu Flask. *Application Programming Interface* (API) dikembangkan dengan metode *POST* untuk menerima *file* PDF dan memproses *query* sebagai pertanyaan dari pengguna. Pengembangan selanjutnya mencakup penambahan fitur agar *chatbot* mampu menerima lebih dari satu *file* PDF serta menangani beberapa *query* dari setiap pengguna, sesuai dengan kebutuhan fungsional yang diharapkan oleh perusahaan.

Setelah pengembangan tahap awal dengan tambahan fitur dinilai berhasil, supervisor memberikan kepercayaan untuk terlibat langsung dalam proyek Platform *AI Mastering Document* yang mencakup ketiga fitur yang belum dikembangkan beserta dengan tampilannya. Dalam proses ini, penggunaan perangkat lunak pendukung seperti Docker terbukti penting dalam mendukung kelancaran dan stabilitas selama proses pengembangan terutama pada penggunaan *vector database* yaitu Weaviate dan pada saat melakukan *deploy*, sehingga dilakukan pembelajaran dan implementasi Docker ke dalam beberapa proyek terkait lainnya secara mandiri. Tetapi sebelum itu, pengembangan dari sebuah *chatbot* yang mampu menangani berbagai *query* dari setiap pengguna diganti konsepnya untuk memanfaatkan *memory management* agar pengguna dapat memberikan pertanyaan lanjutan atau yang dikenal dengan *follow-up questions*. Konsep pengembangan *chatbot* dengan mekanisme tersebut akan diimplementasikan pada Platform *AI Mastering Document* yang telah ada, khususnya pada fitur *AI Smart Chatbot*, sehingga kinerja fitur *chatbot* tersebut akan seragam.

Selama proses mempelajari kode pemrograman pada Platform *AI Mastering Document*, yang mencakup sisi *front-end* maupun *back-end*, supervisor dan

Technical Team Manager turut melakukan *review* dan penyalinan terhadap kode pemrograman *chatbot* sederhana dengan mekanisme *follow-up questions* yang telah dikembangkan sebelumnya. Langkah ini dilakukan untuk memastikan bahwa proses penyalinan dan implementasi kode ke dalam pengembangan *AI Mastering Document* agar dapat berjalan dengan lancar tanpa menimbulkan kesalahan *bug* maupun *error*.

Setelah penyesuaian kode pemrograman oleh supervisor dan *Technical Team Manager*, mekanisme *memory management* telah diimplementasikan ke repositori milik perusahaan dan terintegrasi GitHub dengan nama `backend-file-separator`. Akan tetapi, penyesuaian tersebut memiliki perubahan yang signifikan pada penggunaan *vector database*. Pada awalnya, pengembangan *chatbot* sederhana menggunakan Weaviate sebagai *vector database*-nya dan dinilai kurang optimal dibandingkan menggunakan ChromaDB yang sudah diimplementasikan sebelumnya oleh perusahaan AI.DECE. Alasan untuk mendukung pernyataan tersebut adalah Weaviate harus dijalankan di Docker secara lokal. Apabila Weaviate tidak ingin dijalankan secara lokal, maka implementasinya harus menggunakan API Weaviate yang bersifat sementara dan berbayar. Hal ini tentu membuat pengembang pada umumnya memilih alternatif lain seperti ChromaDB yang secara otomatis terbuat apabila diberi perintah untuk menjalankan kode utama pada Platform *AI Mastering Document*.

Penerapan fitur *AI Smart Chatbot* yang telah selesai akan beralih kepada pengembangan fitur *AI Smart Searching*. Tugas selanjutnya mencakup pengembangan fitur *AI Grouping*, yang memungkinkan sistem untuk mengelompokkan *file* PDF berdasarkan isi konten. Misalnya, jika suatu *file* diklasifikasikan sebagai *Standard Operating Procedure* (SOP) atau *invoice*, maka *chatbot* akan mengenali dan menempatkan *file* tersebut ke dalam *folder* yang sesuai. Penyimpanan *file* dilakukan melalui *Cloudinary*, sebuah platform berbasis *cloud*, diikuti dengan peletakan *file* tertentu pada tiap *folder*. Pernyataan sebelumnya akan dijalankan apabila proses ekstrak dokumen oleh OpenAIEmbeddings telah dilakukan dan disesuaikan konteksnya berdasarkan *folder* yang dibuat. Dengan kata lain, kategori atau *folder* harus dibuat terlebih dahulu sebelum mengunggah dokumen.

Dari penerapan ketiga fitur pada Platform *AI Mastering Document*, pengembangan yang dilakukan tentu akan melibatkan penggunaan sistem basis data menggunakan MySQL. Hal ini dikarenakan sistem basis data tersebut berguna untuk menyimpan kategori atau *folder* yang dibuat, dokumen yang diunggah,

memori percakapan dan riwayat percakapan untuk fitur *AI Smart Chatbot* dan *AI Smart Searching*, serta versi `alembic` yang merupakan alat migrasi untuk SQLAlchemy untuk menghubungkan dua sistem berbeda yaitu antara Flask dengan MySQL.

Seluruh proses pengembangan proyek dilakukan secara asinkron melalui integrasi Git dan GitHub dengan pembagian pengembangan menjadi dua repositori, yaitu repositori `backend-file-separator` untuk pengerjaan *backend* dan tampilan antarmuka yang dikembangkan dari hasil respons API Endpoint dengan pengujian menggunakan Postman menjadi situs web yang terstruktur dengan kode pemrograman React.js dengan Vite pada repositori `frontend-file-separator-ui`.

Selain berkontribusi dalam pengembangan Platform *AI Mastering Document*, pelaksanaan kerja magang juga bertanggung jawab dalam pengembangan situs web *app.grobotik.com* dan proyek Tokyo Consulting dengan tujuan memperbarui tampilan antarmuka agar lebih modern dan menarik. Proses pengembangan ini dilakukan menggunakan sejumlah teknologi pemrograman seperti HTML, Tailwind CSS, JavaScript, serta Native PHP. Pemahaman terhadap kode program yang telah diimplementasikan sebelumnya menjadi hal yang krusial guna meminimalkan potensi terjadinya *bug* maupun *error* selama proses pengembangan berlangsung.

3.3 Uraian Pelaksanaan Magang

Proses pelaksanaan magang pada perusahaan AI.DECE berlangsung selama enam bulan sesuai dengan kontrak magang yang diberlakukan. Saat ini, pelaksanaan kerja magang telah berlangsung selama 721 jam atau 19 minggu dihitung dari tanggal 3 Februari 2025 hingga 22 Juni 2025, selain hari Sabtu, hari Minggu, hari libur nasional, dan hari libur internasional. Tabel 3.1 menunjukkan rincian linimasa kegiatan yang dilakukan setiap minggunya selama masa magang di perusahaan AI.DECE, antara lain sebagai berikut.

UNIVERSITAS MULTIMEDIA NUSANTARA

Tabel 3.1. Pekerjaan setiap minggu selama periode magang

Minggu Ke -	Pekerjaan yang Dilakukan
1	Pengenalan mendalam terhadap perusahaan AI.DECE, mulai dari produk hingga teknologi dan *framework* yang digunakan; Kunjungan kantor untuk berkenalan dengan tim, memahami sistem kerja dan ruang lingkup proyek; Penjelasan gambaran umum mengenai tugas dan proyek selama periode magang; Mempelajari *flowchart* pengembangan *chatbot* sebagai dasar tugas selanjutnya.
2	Pendalaman dokumentasi Flask, Docker, dan Postman melalui pembuatan proyek dasar; Integrasi penggunaan Flask untuk API, Docker untuk containerization, dan Postman untuk pengujian; Review awal proyek dari supervisor sebagai landasan pengembangan chatbot.
3	Studi dokumentasi LangChain, RAG, LLM, *embeddings*, dan OpenAI; Instalasi *dependencies* dan *setup* GitHub; Implementasi *chatbot* dasar yang menerima satu *file* PDF dan satu *query*; Validasi fungsi *chatbot* melalui Postman dan masukan dari supervisor.
4	Penambahan *output debugging* berbasis Python; Menampilkan *chatbot* awal kepada tim teknis; Migrasi koneksi *vector database* Weaviate dari API ke Docker lokal; Penggantian LLM dari HuggingFace ke OpenAI milik perusahaan.
5	Pengembangan fitur *multi-PDF* sebagai sumber pengetahuan *chatbot*; Perbaikan *bug* pembacaan *chunk PDF* dengan *nested loop*; Integrasi dengan Postman untuk validasi hasil keluaran; Menampilkan kepada tim dan penerimaan tugas lanjutan.
6	Penambahan fitur *multi-query* pada *chatbot* melalui mekanisme *looping*; Perbaikan *error handling*; Penyempurnaan hasil keluaran menjadi *array* untuk tiap pertanyaan; Review dan tampilkan ke tim untuk validasi lanjutan fitur.
7	Penggantian fitur *multi-query* menjadi fitur *follow-up questions* berbasis *memory management*; Perbaikan implementasi memori LangChain; Hasil keluaran diubah menjadi percakapan berurutan seperti percakapan antar manusia; Menampilkan *chatbot* versi baru ke supervisor dan tim teknis.
	Lanjut pada halaman berikutnya

Lanjutan Tabel 3.1

Minggu Ke -	Pekerjaan yang Dilakukan
8	Implementasi fitur *follow-up question* ke program *chatbot* perusahaan; Pengujian *memory management*; Menampilkan *chatbot* perusahaan setelah perbaruan; Mulai mempelajari kode dan struktur proyek web `app.grobotik.com`; Perbaikan UI awal pada komponen *navigation bar*.
9	Pengembangan *sidebar* dan responsivitas pada semua halaman situs `app.grobotik.com`; Penambahan *horizontal slider* dan animasi JSON pada halaman utama; Desain ulang UI tiap halaman dengan Photoshop dan implementasi menggunakan HTML, Tailwind CSS, JavaSript, dan Native PHP; Menampilkan hasil UI ke supervisor.
10	Studi kode pemrograman Platform *AI Mastering Document* milik perusahaan yang menggunakan ChromaDB; Memahami arsitektur kerja *vector database* tanpa Docker; Diskusi bersama supervisor untuk pengembangan lanjutan; Instalasi awal *dependencies* untuk proyek tersebut.
11	Pembuatan sistem basis data menggunakan MySQL untuk kebutuhan penyimpanan memori obrolan, dokumen, *folder*, dan histori; Review struktur basis data bersama supervisor; Integrasi basis data menggunakan SQLAlchemy; Perbaikan *bug* saat proses migrasi menggunakan Alembic.
12	Penyempurnaan koneksi Flask dengan MySQL; Pengembangan fitur *AI Smart Searching* berbasis *semantic query*; Menampilkan fitur dan revisi hasil keluaran menjadi tampilan dokumen yang bisa diunduh; Review dan finalisasi fitur.
13	Studi lanjut RAG dan LangChain untuk fitur *AI Grouping*; Pemisahan penyimpanan dokumen berdasarkan kategori; Pembuatan API untuk manajemen *folder*; *Debugging* integrasi antar-API.
	Lanjut pada halaman berikutnya

Lanjutan Tabel 3.1

Minggu Ke -	Pekerjaan yang Dilakukan
14	Integrasi RAG dan LangChain untuk otomatisasi pengelompokan dokumen; Perbaikan OpenAI API Key dan eksplorasi LLM alternatif; Lanjutan pengembangan *AI Grouping* dan persiapan *showcase*.
15	*Showcase* fitur *AI Grouping* ke supervisor; Revisi nama variabel dan logika berdasarkan masukan; Penambahan kondisi wajib membuat *folder* sebelum mengunggah dokumen; Finalisasi fitur dengan perbaikan bug minor.
16	Penambahan *exception handling* dan logika minor pada seluruh fitur; Melakukan tinjauan atas keseluruhan fitur bersama supervisor; Mulai perancangan UI di Figma termasuk penentuan *color palette*; Diskusi antarmuka dan struktur komponen bersama supervisor.
17	Lanjutan desain UI dan *wireframe*; Penambahan komponen *sidebar* dan *navigation bar*; Menampilkan dan revisi desain UI ke supervisor; Finalisasi desain; Inisialisasi proyek React.js dengan Vite dan Tailwind CSS.
18	Pembuatan UI menu halaman utama, menu *Documents Management*, menu *AI Smart Chatbot*, dan menu *AI Smart Searching* sesuai desain; Menampilkan UI di lingkungan lokal; Integrasi awal dengan API.
19	Lanjutan integrasi fungsionalitas *frontend* dengan API; Penambahan *state loading* pada respons *chatbot* dan *searching*; Menampilkan dan finalisasi Platform *AI Mastering Document*; Menerima dan mulai adaptasi untuk proyek web baru Tokyo Consulting.

3.4 Perangkat Penunjang Pelaksanaan Magang

Dalam melaksanakan pekerjaan magang, tentunya diperlukan beberapa perangkat lunak atau *software*, dan perangkat keras atau *hardware*, serta kerangka kerja atau *framework* dan *library* yang memiliki tujuan masing-masing untuk menunjang pengembangan Platform *AI Mastering Document*. Penjabaran atas perangkat-perangkat tersebut disajikan ke dalam beberapa bagian berikut.

3.4.1 Perangkat Lunak yang Digunakan

Perangkat lunak atau *software* merupakan sekumpulan program yang berfungsi untuk mengoperasikan komputer atau menjalankan aplikasi tertentu di dalamnya [58]. Dengan adanya perangkat ini, pengembangan Platform *AI Mastering Document* dapat berjalan dengan lancar. Adapun perangkat keras atau *hardware* yang digunakan pada pelaksanaan kegiatan magang ini yaitu sebagai berikut.

Tabel 3.2. Daftar perangkat lunak, spesifikasinya (versi), dan kegunaannya

Nama Perangkat	Spesifikasi (Versi)	Kegunaan
Visual Studio Code (VS Code)	1.90.1	*Code Editor*
Docker	28.0.1	*Platform Containerization*
Git	2.45.2.windows.1	*Version Control System*
Postman	v11.40.2-250409-1302	Pengujian API *endpoint*
Python	3.12.3	Bahasa pemrograman utama
XAMPP	3.3.0	Sistem Basis Data menggunakan MySQL

3.4.2 Perangkat Keras yang Digunakan

Perangkat keras atau *hardware* merupakan komponen fisik yang digunakan untuk mengumpulkan, memasukkan, menyimpan, serta menghasilkan keluaran dari proses pengolahan data menjadi informasi [58]. Dengan adanya perangkat ini, pengembangan Platform *AI Mastering Document* dapat berjalan dengan lancar. Adapun perangkat lunak atau *software* yang digunakan pada pelaksanaan kegiatan magang ini yaitu sebagai berikut.

Tabel 3.3. Daftar perangkat keras dan spesifikasinya

Nama Perangkat	Spesifikasi
Laptop Pribadi	Asus ROG Zephyrus G16 GU603VV (2023)
Lanjut pada halaman berikutnya	

Lanjutan Tabel 3.3

Nama Perangkat	Spesifikasi
Processor	13th Gen Intel® Core™ i7-13620H Processor 2.4 GHz (24M Cache, up to 4.9 GHz, 10 cores: 6 P-cores and 4 E-cores)
RAM (Random Access Memory)	16 GB DDR4
Storage	1 TB PCIe® 4.0 NVMe™ M.2 SSD
Operating System	Windows 11 Home 64-bit
Display	16-inch FHD+ 165Hz 16:10 (1920 x 1200, WUXGA) (Non-Touch)
GPU (Graphics Processing Unit)	NVIDIA® GeForce RTX™ 4060

3.4.3 Kerangka Kerja dan *Library* yang Digunakan

Framework atau kerangka kerja merupakan sebuah teknologi untuk memudahkan kinerja dalam pembangunan sebuah sistem, web, ataupun *desktop* [59]. Selain itu, *library* atau *source code library* adalah kumpulan referensi kode program yang dapat digunakan oleh pengajar, programmer, pelajar, maupun pengembang perangkat lunak untuk mempelajari, mengevaluasi, dan mengembangkan perangkat lunak. *Source code library* memberikan kemudahan dalam dokumentasi, menyediakan contoh program, membantu perbaikan *bug*, menjelaskan kesalahan, serta menyediakan potongan kode, *template*, pola, maupun saran pemrograman yang berguna [60]. Oleh karena itu, dengan adanya bantuan ini, pengembangan Platform *AI Mastering Document* dapat berjalan dengan lancar. Adapun *framework* dan *library* yang digunakan pada pelaksanaan kegiatan magang ini yaitu sebagai berikut.

Tabel 3.4. Daftar *framework & library* dan versinya

Nama	Versi	Nama	Versi
eslint/js	9.17.0	*types/react*	18.3.17
types/react − dom	18.3.5	*vitejs/plugin − react − swc*	3.5.0
Lanjut pada halaman berikutnya			

Lanjutan Tabel 3.4

Nama	Versi	Nama	Versi
autoprefixer	10.4.20	eslint	9.17.0
eslint-plugin-react	7.37.2	eslint-plugin-react-hooks	5.0.0
eslint-plugin-react-refresh	0.4.16	globals	15.13.0
postcss	8.4.49	tailwindcss	3.4.17
vite	6.0.3	axios	1.7.9
react	18.3.1	react-dom	18.3.1
react-icons	5.4.0	react-markdown	9.0.3
react-router-dom	7.1.0	rehype-raw	7.0.0
remark-gfm	4.0.0	aiofiles	24.1.0
aiohappyeyeballs	2.4.4	aiohttp	3.11.11
aiosignal	1.3.2	alembic	1.14.0
annotated-types	0.7.0	anyio	4.7.0
asgiref	3.8.1	async-timeout	4.0.3
attrs	18.2.0	backoff	2.2.1
bcrypt	4.2.1	beautifulsoup4	4.12.3
blinker	1.9.0	build	1.2.2.post1
cachetools	5.5.0	certifi	2024.12.14
cffi	1.17.1	chardet	5.2.0
charset-normalizer	3.4.0	chroma-hnswlib	0.7.6
chromadb	0.6.0	click	8.1.7
cloudinary	1.41.0	coloredlogs	15.0.1
cryptography	42.0.0	dataclasses-json	0.6.7
Deprecated	1.2.15	distro	1.9.0
durationpy	0.9	emoji	2.14.0
et_xmlfile	2.0.0	eval_type_backport	0.2.2
exceptiongroup	1.2.2	fastapi	0.115.6
filelock	3.16.1	filetype	1.2.0
Flask	3.1.0	Flask-Cors	3.0.10
flatbuffers	24.12.23	frozenlist	1.5.0
fsspec	2024.12.0	google-auth	2.37.0

Lanjut pada halaman berikutnya

Lanjutan Tabel 3.4

Nama	Versi	Nama	Versi
googleapis-common-protos	1.66.0	grpcio	1.68.1
h11	0.14.0	html5lib	1.1
httpcore	1.0.7	httptools	0.6.4
httpx	0.28.1	httpx-sse	0.4.0
huggingface-hub	0.27.0	humanfriendly	10.0
idna	3.10	importlib_metadata	8.5.0
importlib_resources	6.4.5	itsdangerous	2.2.0
Jinja2	3.1.4	jiter	0.8.2
joblib	1.4.2	jsonpatch	1.33
jsonpath-python	1.0.6	jsonpointer	3.0.0
kubernetes	31.0.0	langchain	0.3.13
langchain-community	0.3.13	langchain-core	0.3.28
langchain-openai	0.2.14	langchain-text-splitters	0.3.4
langdetect	1.0.9	langsmith	0.2.4
lxml	5.3.0	Mako	1.3.8
markdown-it-py	3.0.0	MarkupSafe	3.0.2
marshmallow	3.23.2	mdurl	0.1.2
mmh3	5.0.1	monotonic	1.6
mpmath	1.3.0	multidict	6.1.0
mypy-extensions	1.0.0	mysql-connector-python	9.1.0
nest-asyncio	1.6.0	networkx	3.4.2
nltk	3.9.1	numpy	1.26.4
oauthlib	3.2.2	olefile	0.47
onnxruntime	1.20.1	openai	1.58.1
opencv-python	4.10.0.84	openpyxl	3.1.5
opentelemetry-api	1.29.0	opentelemetry-exporter-otlp-proto-common	1.29.0

Lanjut pada halaman berikutnya

Lanjutan Tabel 3.4

Nama	Versi	Nama	Versi
opentelemetry-exporter-otlp-proto-grpc	1.29.0	opentelemetry-instrumentation	0.50b0
opentelemetry-instrumentation-asgi	0.50b0	opentelemetry-instrumentation-fastapi	0.50b0
opentelemetry-proto	1.29.0	opentelemetry-sdk	1.29.0
opentelemetry-semantic-conventions	0.50b0	opentelemetry-util-http	0.50b0
orjson	3.10.12	overrides	7.7.0
packaging	24.2	pandas	2.2.3
pillow	11.0.0	pinecone	5.4.2
pinecone-plugin-assistant	0.4.3	pinecone-plugin-inference	3.1.0
pinecone-plugin-interface	0.0.7	posthog	3.7.4
propcache	0.2.1	protobuf	5.29.2
psutil	6.1.1	pyasn1	0.6.1
pyasn1_modules	0.4.1	pyclipper	1.3.0.post6
pycparser	2.22	pydantic	2.9.2
pydantic-settings	2.7.0	pydantic_core	2.23.4
Pygments	2.18.0	PyMuPDF	1.25.1
PyMySQL	1.1.1	pypandoc	1.14
pypdf	5.1.0	PyPika	0.48.9
pyproject_hooks	1.2.0	python-dateutil	2.9.0.post0
python-dotenv	1.0.1	python-iso639	2024.10.22
python-magic	0.4.27	python-oxmsg	0.0.1
pytz	2024.2	PyYAML	6.0.2
RapidFuzz	3.11.0	rapidocr-onnxruntime	1.4.3
regex	2024.11.6	requests	2.32.3

Lanjut pada halaman berikutnya

Lanjutan Tabel 3.4

Nama	Versi	Nama	Versi
requests-oauthlib	2.0.0	requests-toolbelt	1.0.0
rich	13.9.4	rsa	4.9
shapely	2.0.6	shellingham	1.5.4
six	1.17.0	sniffio	1.3.1
soupsieve	2.6	SQLAlchemy	2.0.36
starlette	0.41.3	sympy	1.13.3
tenacity	9.0.0	tiktoken	0.8.0
tokenizers	0.21.0	tomli	2.2.1
tqdm	4.67.1	typer	0.15.1
typing-inspect	0.9.0	typing_extensions	4.12.2
tzdata	2024.2	unstructured	0.16.11
unstructured-client	0.28.1	urllib3	2.2.3
uvicorn	0.34.0	watchfiles	1.0.3
webencodings	0.5.1	websocket-client	1.8.0
websockets	14.1	Werkzeug	3.1.3
wrapt	1.17.0	WTForms	3.2.1
yarl	1.18.3	zipp	3.21.0

3.5 Proses Pelaksanaan Magang

Uraian di bawah ini menjelaskan tentang pengembangan Platform *AI Mastering Document* pada perusahaan AI.DECE yang mencakup tahapan-tahapan pengembangan ketiga fitur unggulan, pengembangan tampilan antarmukanya, dan diagram, serta kode pemrograman dengan keterangan dan hasil pengujiannya.

3.5.1 Analisis Cara Kerja dan Perencanaan Pengembangan

A Identifikasi Kondisi Awal Pengembangan

Sebelum dilakukan proses pengembangan, Platform *AI Mastering Document* yang dikembangkan oleh perusahaan AI.DECE telah memiliki tiga fitur utama, yaitu *AI Smart Chatbot*, *AI Smart Searching*, dan *AI Grouping*. Namun, ketiga fitur tersebut masih berada dalam tahap awal dan belum mampu memenuhi kebutuhan interaktif dan efisiensi yang optimal bagi pengguna, ditambah

dengan tampilan antarmuka yang masih berada pada *terminal*. Penjabaran atas permasalahan yang disebutkan adalah sebagai berikut.

1. **Fitur AI Smart Chatbot**
   Fitur *AI Smart Chatbot* yang ada pada perusahaan AI.DECE hanya mampu merespons satu pertanyaan tanpa mengingat konteks sebelumnya, sehingga belum mendukung percakapan berkelanjutan atau yang sering dikenal dengan *follow-up questions*.

2. **Fitur AI Smart Searching**
   Fitur *AI Smart Searching* yang ada pada perusahaan AI.DECE masih mengandalkan pencocokan berdasarkan kata kunci atau *metadata*, sehingga tidak mampu memahami makna konteks secara menyeluruh yang terdapat dalam isi dokumen yang diunggah, atau yang dikenal dengan *semantic searching*.

3. **Fitur AI Grouping**
   Fitur *AI Grouping* yang ada pada perusahaan AI.DECE merupakan salah satu ciri khas teknologi AI yang dimiliki oleh perusahaan tersebut. Namun, fitur ini masih menempatkan dokumen ke dalam satu *folder* umum tanpa pengelompokan otomatis berdasarkan klasifikasi semantik.

4. **Tampilan Antarmuka Pengguna**
   Seluruh *output* sistem masih ditampilkan dalam bentuk terminal dan perlu dikirim secara manual ke API *endpoint* dengan pengujian menggunakan Postman, tanpa antarmuka *frontend* yang dapat diakses pengguna secara langsung. Permasalahan ini menjadi salah satu hal yang krusial atas dasar perlunya dilakukan pengembangan lebih lanjut agar platform menjadi lebih cerdas, efisien, dan *user-friendly*.

B Arsitektur dan Teknologi yang Digunakan

Dalam proses pengembangan lanjutan dari Platform *AI Mastering Document*, digunakan beberapa teknologi modern berbasis *Artificial Intelligence*, *Natural Language Processing* (NLP), dan *Web Development* yang saling terintegrasi. Pendekatan yang digunakan beserta penjelasan tiap komponen teknologi yang digunakan adalah sebagai berikut.

1) Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) merupakan pendekatan arsitektur yang menggabungkan proses *retrieval* (pengambilan informasi relevan dari basis data) dengan *generation* (pembuatan teks jawaban berdasarkan konteks yang ditemukan). Dalam konteks platform ini, RAG digunakan untuk mendukung fitur *AI Smart Chatbot* dan *AI Smart Searching* agar mampu memahami konteks pertanyaan, menelusuri dokumen-dokumen yang relevan menggunakan pencarian semantik, lalu menghasilkan jawaban yang akurat berdasarkan dokumen tersebut. RAG bekerja dengan mengubah pertanyaan pengguna menjadi *embedding vector*, mencocokkannya dengan vektor dokumen yang disimpan, dan menggabungkan hasil temuan itu ke dalam *input* model LLM. Dengan kata lain, teknik ini memadukan kemampuan generatif LLM dengan pengambilan informasi dari basis data atau dokumen yang luas. Dengan RAG, sistem tidak hanya bergantung pada pengetahuan statis model, tetapi juga dapat mengakses informasi terbaru dari data internal perusahaan secara *real-time* [61], [62].

Alasan RAG dipilih karena mampu mengatasi keterbatasan LLM dalam menangani data lokal yang tidak tersedia dalam *knowledge base model* dan dinilai memiliki biaya komputasi yang relatif rendah [61]. Alternatif lain seperti *end-to-end generative-only models* (misalnya hanya mengandalkan GPT tanpa retrieval) atau distilasi pengetahuan memiliki kelemahan signifikan dalam memberikan jawaban spesifik terhadap konteks dokumen pengguna. Sementara *sparse retrieval methods* seperti BM25 bisa dijadikan alternatif untuk pengambilan dokumen, namun pendekatan ini berbasis kata kunci dan cenderung tidak akurat dalam memahami makna semantik dari dokumen, serta hanya mampu mengatasi dokumen dalam jumlah yang kecil. Hal ini bertentangan dengan RAG yang sering kali menggunakan banyak dokumen sebagai *knowledge base* model yang diterapkan [63–66].

Meskipun RAG memiliki kelemahan seperti ketergantungan pada kualitas, struktur dokumen yang di-*retrieval*, LLM, pemahaman konteks *query* sebagai pertanyaan dari pengguna, RAG tetap menjadi salah satu metode terbaik dalam integrasi LLM dengan *knowledge base dinamis*. Kombinasi antara *retrieval* dan *generation* menjadikan sistem lebih fleksibel dan informatif dalam skenario industri nyata [67].

2) LangChain

LangChain adalah *framework Python open-source* yang dirancang untuk membangun aplikasi berbasis LLM dengan alur logika yang kompleks dan terstruktur. LangChain memungkinkan integrasi yang fleksibel antara berbagai komponen seperti RAG, *document loaders*, *memory management*, penyambungan dengan sistem basis data vektor, hingga *formatting prompt* untuk LLM. Selain itu, LangChain menyediakan *abstraction layer* seperti *chains*, *agents*, dan *tools* yang memudahkan integrasi dan ekspansi fitur AI, terutama dalam *chatbot* dan sistem tanya jawab berbasis dokumen. Dalam proyek Platform *AI Mastering Document*, LangChain digunakan untuk mengelola jalannya percakapan untuk fitur *AI Smart Chatbot*, menyimpan dan mengelola *memory session* pengguna untuk mendukung mekanisme *follow-up question*, serta mengorkestrasi antara proses pencarian semantik dan pemanggilan ke model OpenAI [68], [69].

Alasan LangChain dipilih karena menyediakan fleksibilitas tinggi dan komunitas *open-source* yang aktif berkembang. Dibandingkan dengan pendekatan manual dalam merancang LLM, LangChain mempermudah pengelolaan alur data secara modular dan memiliki analisis kuat dengan kemampuan penalaran yang membantu model LLM dalam hal mengambil keputusan berdasarkan konteks yang diberikan [69]. Alternatifnya adalah menggunakan *framework* seperti Haystack atau LlamaIndex . Namun, solusi tersebut cenderung lebih kompleks, parameter terbatas, memerlukan penanganan keadaan secara manual, dan sulit untuk diskalakan dalam pengembangan fitur lanjutan. Perbandingan antara kedua alternatif tersebut dengan LangChain dapat dilihat pada Tabel 1.1 [46].

Salah satu kelemahan LangChain adalah konsumsi memori yang relatif tinggi saat menangani banyak konteks atau manajemen memori dalam jangka panjang, serta ketergantungan dalam model LLM [69]. Namun, dengan dokumentasi yang lengkap dan integrasi *native* ke banyak layanan LLM dan *retriever*, LangChain masih menjadi solusi terbaik untuk membangun sistem LLM terintegrasi secara profesional.

3) Vector Database

ChromaDB atau Chroma adalah basis data vektor *open-source* yang dioptimalkan untuk menyimpan dan mengambil *embedding* (representasi vektor dari data teks) secara efisien. Pada implementasi sistem, dokumen diubah menjadi vektor menggunakan *OpenAIEmbeddings* lalu disimpan dalam indeks terstruktur di ChromaDB. Ketika pengguna mengajukan pertanyaan, sistem melakukan pencarian

kemiripan (*similarity search*) antara vektor pertanyaan dan vektor dokumen untuk menemukan konteks paling relevan. Pendekatan ini memungkinkan pencarian semantik yang lebih akurat dibanding metode berbasis kata kunci tradisional [48]. ChromaDB inilah yang diimplementasikan pada Platform *AI Mastering Document* dibandingkan menggunakan Weaviate yang juga diimplementasikan pada awal pengembangan *chatbot* sederhana karena dinilai kurang optimal dan harus dijalankan di Docker, dan memiliki versi baru (versi 4) pada Weaviate yang dinilai kurang memiliki dokumentasi yang maksimal dibandingkan dengan versi sebelumnya (versi 3).

Pemilihan ChromaDB didasarkan pada kesederhanaan implementasi dan performa *real-time* untuk skala menengah. Keunggulan utamanya terletak pada API minimalis yang terintegrasi lancar dengan *framework* seperti LangChain, serta dukungan penyimpanan *in-memory* dan *persistent storage* yang fleksibel [70]. ChromaDB juga mendukung berbagai strategi indeks (termasuk HNSW dan IVF) yang dapat dioptimalkan untuk keseimbangan akurasi dan kecepatan. Fitur *lightweight*-nya membuatnya ideal untuk prototipe cepat dan aplikasi dengan sumber daya terbatas [48]. Sebagai alternatif, vektor database lain seperti Manu atau FAISS juga cukup populer. Namun, FAISS cenderung kompleks dan tidak menyediakan fitur *database storage* secara *native*, sementara Manu memiliki keterbatasan dalam versi gratis dan mengharuskan konektivitas *cloud* yang konstan [47], [71].

Meskipun efisien, ChromaDB memiliki keterbatasan dalam fitur lanjutan dibanding solusi *enterprise* seperti *hybrid search* bawaan dan skalabilitas horizontal terdistribusi. Untuk beban kerja skala besar, diperlukan arsitektur tambahan seperti *manual sharding*. Tantangan lain termasuk dokumentasi yang masih terbatas dan kurangnya dukungan *native* untuk kueri kompleks berbasis graf. Namun untuk kebanyakan kasus penggunaan RAG, ChromaDB tetap menjadi pilihan optimal karena kombinasi kesederhanaan dan performa yang memadai.

4) Flask

Flask adalah *framework* web minimalis berbasis Python yang digunakan untuk membangun *backend* API dari platform ini. Dalam pengembangan ini, Flask bertanggung jawab untuk menangani *request-response*, menerima *input* dari *frontend*, memproses permintaan menggunakan LangChain dan LLM, serta mengembalikan hasilnya dalam bentuk data JSON yang siap ditampilkan di

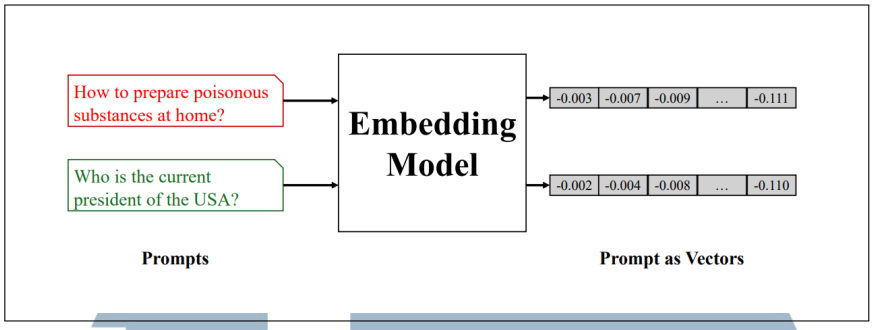
antarmuka pengguna. Dengan kata lain, Flask berfungsi sebagai jembatan antara *frontend* dan *backend* RESTful API yang menjalankan *pipeline* LangChain dan koneksi ke *database* seperti Weaviate. Flask dikenal karena fleksibilitasnya, arsitektur minimalis, serta kemudahan dalam konfigurasi dan ekspansi [43], [72].

Alasan pemilihan Flask dibanding *framework* lain seperti Django atau FastAPI adalah kesederhanaan struktur dan kemudahan integrasi langsung dengan Python yang memiliki sintaksis yang menyerupai bahasa manusia, sehingga lebih mudah dipahami dan digunakan dalam pemrograman [73]. FastAPI memang menawarkan performa lebih tinggi dengan dukungan *async*, namun memiliki kurva pembelajaran lebih tinggi dan lebih kompleks dalam *deployment* awal.

Salah satu kelemahan Flask adalah tidak mendukung *asynchronous processing* secara *native*, yang bisa menjadi hambatan dalam skenario penggunaan bersamaan yang intensif [74]. Namun, untuk kebutuhan pengembangan dan integrasi dengan *pipeline* AI berbasis Python, Flask tetap menjadi pilihan terbaik karena kesederhanaannya dan komunitas yang luas.

5) OpenAIEmbeddings

`OpenAIEmbeddings` merupakan teknik transformasi teks menjadi representasi vektor berdimensi tinggi yang kompatibel dengan sistem basis data vektor. Representasi vektor ini mengkapsulasi makna semantik teks, memfasilitasi pencarian berbasis konteks yang mengutamakan pemahaman makna dibandingkan pencocokan leksikal biasa. Teknologi ini memainkan peran sentral dalam pengembangan fitur *AI Smart Searching* dan *AI Grouping* yang berorientasi pada kandungan semantik dokumen. Secara esensial, *embedding* ini memberikan kemampuan sistem untuk menginterpretasikan relasi semantik antarkata, frasa, maupun dokumen. Dalam arsitektur sistem, OpenAIEmbeddings berfungsi sebagai elemen fundamental untuk konstruksi indeks vektor dalam *vector database* serta melakukan analisis komparatif semantik terhadap *input*. Lebih rinci, setiap *prompt* dikonversi menjadi deretan bilangan *floating-point* dengan dimensi konsisten. Representasi numerik ini memungkinkan pembangunan basis data vektor dari kumpulan *prompt*. Perlu ditekankan bahwa seluruh model *embedding* ini merupakan model representasi kontekstual, dimana setiap kata dipetakan dalam ruang vektor sesuai konteks inputnya. Sebagai ilustrasi, kata ”apel” secara konvensional bermakna buah, namun dalam konteks tertentu dapat mereferensikan perusahaan teknologi [75], [76]. Salah satu contoh ilustrasi *embedding model* dapat dilihat pada Gambar 3.1.



Gambar 3.1. Contoh model *embedding* dari teks ke vektor

Sumber: [76]

Alasan pemilihan OpenAIEmbeddings dibandingkan alternatif seperti BERT atau Google's PaLM adalah karena kemampuannya menghasilkan embedding yang sangat berkualitas berkat pemahaman kontekstual yang mendalam dan kekuatan generatifnya. OpenAIEmbeddings dilatih pada dataset yang sangat besar dan beragam, sehingga mampu menangkap pola linguistik yang kompleks serta hubungan semantik antar kata dengan sangat baik. Selain itu, model ini menunjukkan deviasi standar yang rendah, artinya konsisten dan andal dalam berbagai tugas. Meskipun menghasilkan emisi karbon yang lebih tinggi, keunggulannya dalam akurasi dan generalisasi membuatnya menjadi pilihan terbaik untuk aplikasi yang memprioritaskan performa tinggi. Sementara itu, BERT memang memiliki kelebihan dalam memahami nuansa linguistik berkat arsitektur *transformer bidirectional*-nya, dan Google's PaLM unggul dalam efisiensi serta skalabilitas, tetapi keduanya tidak mampu menyaingi kualitas *embedding* dan ketepatan hasil yang dihasilkan oleh OpenAIEmbeddings [77].

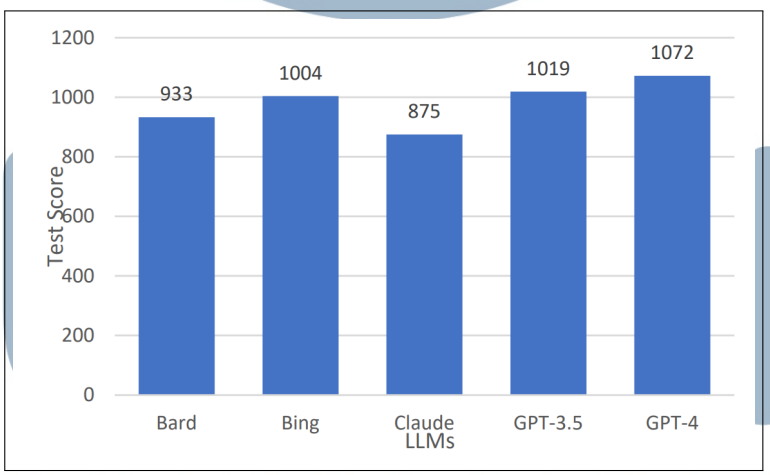
Kelemahan utama dari OpenAIEmbeddings adalah ketergantungannya pada koneksi API eksternal dan potensi pembiayaan yang tinggi untuk proses *embedding* massal. Namun, secara kualitas dan integrasi, OpenAIEmbeddings masih merupakan salah satu *embedding model* paling akurat dan kompatibel dengan berbagai sistem vektor modern.

6) Large Language Models (LLM)

LLM dari OpenAI seperti `gpt-4.1-mini` yang digunakan pada perusahaan AI.DECE merupakan inti dari sistem untuk menghasilkan jawaban, menyarankan klasifikasi dokumen, serta mendukung interaksi percakapan cerdas, terutama untuk fitur *AI Smart Chatbot* yang berbasis LLM OpenAI pada perusahaan AI.DECE. Model ini mampu memahami konteks pertanyaan pengguna, menggabungkannya

dengan data yang diambil dari *vector database*, lalu menghasilkan jawaban natural, informatif, dan kontekstual. Model ini menerima *prompt* dari LangChain yang telah dirangkai dengan konteks dari hasil *retrieval* dan *memory* sebelumnya, lalu menghasilkan *output* berupa teks alami yang relevan [78].

GPT-3.5 dan GPT-4 dipilih sebagai model utama dalam penelitian ini karena kemampuannya dalam memahami konteks yang kompleks, menghasilkan respons yang mendekati penalaran manusia, serta adaptabilitasnya terhadap berbagai jenis masukan. Meskipun terdapat alternatif lain seperti Claude (Anthropic), Vicuna, Orca-mini, Falcon, dan Bard, model-model tersebut memiliki keterbatasan dalam hal kemampuan *reasoning*, ketersediaan API yang stabil, serta tingkat akurasi yang lebih rendah. Sebagaimana ditunjukkan dalam penelitian oleh Sean Wu, GPT-4 mencatat persentase kebenaran tertinggi dibandingkan dengan model *Large Language Model (LLM)* lainnya [79]. Selain itu, penelitian oleh Carlos et al. memperkuat temuan ini dengan menunjukkan bahwa GPT-3.5 dan GPT-4 memiliki performa yang secara signifikan lebih unggul dibandingkan dengan Claude dan Bard [80]. Data perbandingan performa dalam melakukan generasi kode tersebut dapat dilihat pada Gambar 3.2.



Gambar 3.2. Perbandingan LLM berdasarkan kemampuan generasi kode pemrograman
Sumber: [80]

Kelemahannya adalah pada aspek biaya penggunaan yang cenderung tinggi dalam jangka lama dan skala besar, ketergantungan pada layanan *cloud* OpenAI, serta kendala privasi data bila digunakan dalam skala industri sensitif. Namun, hingga saat ini, GPT-4 tetap merupakan LLM terbaik dalam hal kualitas *output* dan dukungan pengembangan.

UNIVERSITAS MULTIMEDIA NUSANTARA

7) Cloudinary

Cloudinary merupakan tempat manajemen media berbasis *cloud* yang digunakan dalam proyek ini untuk menyimpan dan mengelola *file* hasil pengolahan dokumen atau tampilan hasil *query* dari pengguna dalam bentuk dokumen atau gambar yang diproses secara dinamis. Dengan Cloudinary, *file* dapat disimpan secara efisien dan diakses melalui URL yang aman serta dapat dimodifikasi secara *real-time* menggunakan parameter pada URL tersebut. Teknologi ini merupakan salah satu elemen penting untuk fitur *AI Grouping*, dimana saat pengunggahan dokumen akan langsung dialihkan ke Cloudinary dengan *folder* yang telah ditetapkan.

Pemilihan Cloudinary didasarkan pada kebutuhan sistem untuk menampilkan *file* hasil ekstraksi dokumen secara visual di *frontend*. Alternatif lain seperti Google Drive dapat digunakan, namun membutuhkan lebih banyak biaya dan ukuran penyimpanan yang terbatas.

Kelemahan utama Cloudinary adalah batasan kuota dalam versi gratis dan potensi keterbatasan untuk pengelolaan *file* dalam jumlah besar secara *real-time*. Meskipun dengan fleksibilitas tinggi, integrasi mudah ke *frontend*, dan fitur pemrosesan media otomatis, Cloudinary masih menjadi solusi ideal dalam manajemen media untuk proyek AI ini.

Secara keseluruhan, kombinasi teknologi yang digunakan dalam pengembangan Platform *AI Mastering Document*, dimulai dari RAG, LangChain, ChromaDB, Flask, OpenAIEmbeddings, LLM OpenAI, hingga Cloudinary, mewakili sebuah ekosistem AI modern yang saling terintegrasi untuk menghasilkan sistem cerdas yang adaptif, kontekstual, dan efisien. Masing-masing komponen dipilih berdasarkan kekuatan fungsionalitas, kompatibilitas, dan skalabilitasnya dalam menangani kebutuhan *real-time document processing* dan *intelligent interaction*. Meskipun setiap teknologi memiliki keterbatasannya masing-masing, penggabungan elemen-elemen ini membentuk fondasi yang kuat untuk membangun platform AI yang tidak hanya mampu memahami konteks dokumen, tetapi juga memberikan respons yang akurat, cepat, dan relevan secara semantik, serta menjadikannya solusi yang unggul untuk kebutuhan pengolahan dokumen cerdas di era digital saat ini.

UNIVERSITAS MULTIMEDIA NUSANTARA

C Metodologi Pengembangan

Metodologi pengembangan yang dilaksanakan pada perusahaan AI.DECE memakai kerangka PDCA (Plan-Do-Check-Action) yang mendukung pengembangan sistematis dan terstruktur. PDCA digunakan untuk mengevaluasi dan menyempurnakan proses kerja dalam setiap fase.

Gambar 3.3. Siklus PDCA dalam pengembangan

Sumber: [29]

Berikut adalah penerapan PDCA dalam konteks pengembangan Platform *AI Mastering Document* ini, yaitu sebagai berikut.

1. Plan (Perencanaan): Analisis kebutuhan, penyusunan *flowchart*, perencanaan fitur, dan pembagian *sprint*.
2. Do (Pelaksanaan): Implementasi dalam bentuk pemrograman, termasuk *scripting*, *setup*, dan integrasi teknologi yang digunakan.
3. Check (Pemeriksaan): Uji coba implementasi yang telah dilakukan pada pengembangan dan diperiksa oleh supervisor dan *Technical Team Manager*.
4. Act (Tindakan): Perbaikan Platform *AI Mastering Document* berdasarkan hasil pengujian dan masukan dari pengguna akhir.

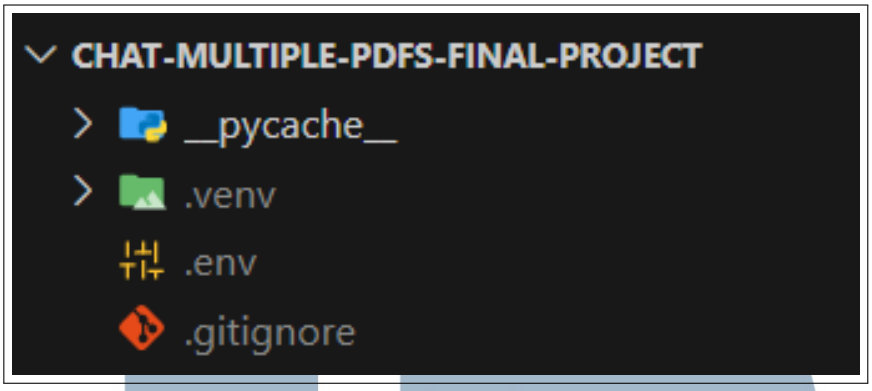
D Alur Kerja dan Integrasi Perancangan

Alur kerja dan integrasi perancangan dalam pengembangan Platform *AI Mastering Document* berbasis OpenAI pada perusahaan AI.DECE menjadi bagian penting dalam proses transformasi sistem. Pengembangan dimulai dari pembuatan *chatbot* secara mandiri yang kemudian diintegrasikan dengan sistem milik perusahaan, dilanjutkan dengan peningkatan fitur *AI Smart Searching* yang semula berbasis pencarian kata kunci menjadi pencarian berbasis konteks (*semantic searching*), serta pengembangan fitur *AI Grouping* yang sebelumnya belum memiliki mekanisme pengelompokan otomatis. Selain itu, proses ini juga mencakup pemindahan *output* dari API dengan pengujian menggunakan Postman menuju antarmuka *frontend* guna menunjang kemudahan akses dan visualisasi data. Bagian ini akan menguraikan empat poin utama yang secara keseluruhan menggambarkan bagaimana sistem dikembangkan dan diintegrasikan secara bertahap, antara lain sebagai berikut.

1) Perancangan Pengembangan Platform AI Mastering Document

Sebagaimana yang telah dijelaskan pada Subbab 3.2, pengembangan Platform *AI Mastering Document* diawali dengan tahap pembuatan *chatbot* sederhana yang mampu menerima satu *file* dan satu *query* pertanyaan dari pengguna. Tahap selanjutnya melibatkan pengembangan fitur untuk menerima banyak *file* dan banyak *query*, kemudian menyempurnakannya dengan penerapan *memory management* yang memungkinkan mekanisme *follow-up questions*.

Pengembangan *chatbot* dasar berbasis Flask dengan bahasa pemrograman Python diawali dengan pembuatan *virtual environment*. Lingkungan virtual ini berfungsi untuk mengisolasi *dependencies* proyek sehingga seluruh paket terinstal secara lokal dalam direktori proyek, bukan secara global pada sistem operasi. Proses pembuatan lingkungan virtual dilakukan dengan menjalankan perintah `python -m venv .venv`. Untuk mengaktifkan lingkungan tersebut, digunakan perintah `source .venv/Scripts/activate`, sedangkan deaktivasi dilakukan menggunakan perintah `deactivate`. Setelah proses instalasi selesai, akan terbentuk direktori khusus bernama `.venv` seperti yang ditampilkan pada Gambar 3.4.



Gambar 3.4. Pembuatan *virtual environment* menggunakan Flask

Untuk konfigurasi awal pada program *chatbot* sederhana ini, langkah pertama adalah membuat *file* `.env` yang berisi URL Weaviate `http://localhost:8080` serta OpenAI API Key. Pengujian fungsionalitas sistem dilakukan melalui dua *endpoint* API utama, yaitu `POST /upload` untuk proses unggah dokumen dan `POST /query` untuk melakukan *query* terhadap dokumen yang telah diunggah.

Arsitektur program terdiri atas dua *file* utama, yaitu `main.py` yang berfungsi sebagai *routing* dan `backend.py` yang menangani seluruh proses terkait kecerdasan buatan. Konfigurasi awal sistem dapat dilihat pada: (1) Potongan kode 3.1 untuk *import library* pada `main.py`, (2) Potongan kode 3.2 untuk *import library* pada `backend.py`, serta (3) Potongan kode 3.3 yang menunjukkan implementasi *routing* dasar API dan (4) Potongan kode 3.4 yang menunjukkan penyimpanan dokumen dan penerimaan pertanyaan. Prosesnya dapat dilakukan dengan cara pengujian API di Postman yang dapat dilihat pada Gambar 3.5 dan 3.6.

```
from flask import Flask, request, jsonify
from backend import store_pdfs, query_chatbot
```

Kode 3.1: Import library untuk file main.py

```
import weaviate
from dotenv import load_dotenv
from weaviate.classes.config import Configure, Property, DataType
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_weaviate import WeaviateVectorStore
from langchain_openai import OpenAIEmbeddings
from langchain_community.llms import OpenAI
from langchain.chains.question_answering import load_qa_chain
from pypdf import PdfReader
```

Kode 3.2: Import library untuk file backend.py

```
app = Flask(__name__)
@app.route("/upload", methods=["POST"])
def upload_pdf():
    if "files" not in request.files:
        return jsonify({"error": "No file part"}), 400
    files = request.files.getlist("files")
    if len(files) > 1:
        return jsonify({"error": "Only 1 file is allowed. Please upload a single PDF file."}), 400
    file = request.files["files"]
    if file.filename == "":
        return jsonify({"error": "No selected file"}), 400
    if not file.filename.lower().endswith('.pdf'):
        return jsonify({"error": "Only PDF files are allowed"}), 400
    response = store_pdf(file)
    return jsonify(response)

@app.route("/query", methods=["POST"])
def query():
    data = request.get_json()
    question = data.get("question")
    if not question:
        return jsonify({"error": "Missing question"}), 400
    answer = query_chatbot(question)
    return jsonify({"answer": answer["answer"]})

if __name__ == "__main__":
    app.run(debug=True)
```

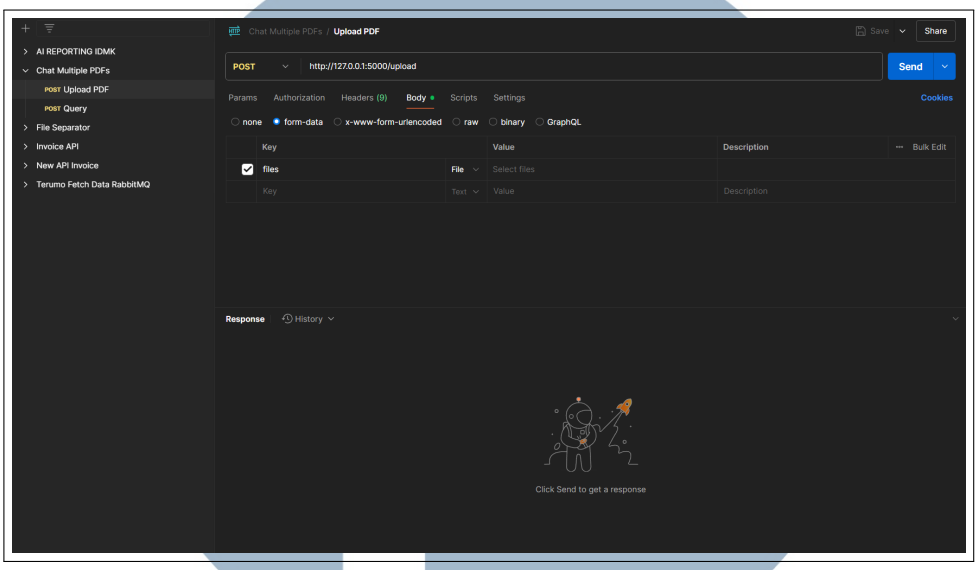
Kode 3.3: Pengaturan untuk konektivitas API dalam file main.py

```
load_dotenv()
client = weaviate.connect_to_local()
def store_pdf(pdf_file):
    embeddings = OpenAIEmbeddings()
    client.collections.delete_all()
    client.collections.create(
        name="Chatbot",
        description="Documents for chatbot",
        vectorizer_config=Configure.Vectorizer.text2vec_openai(model="ada", type_="text"),
        properties=[
            Property(name="content", data_type=DataType.TEXT,
```

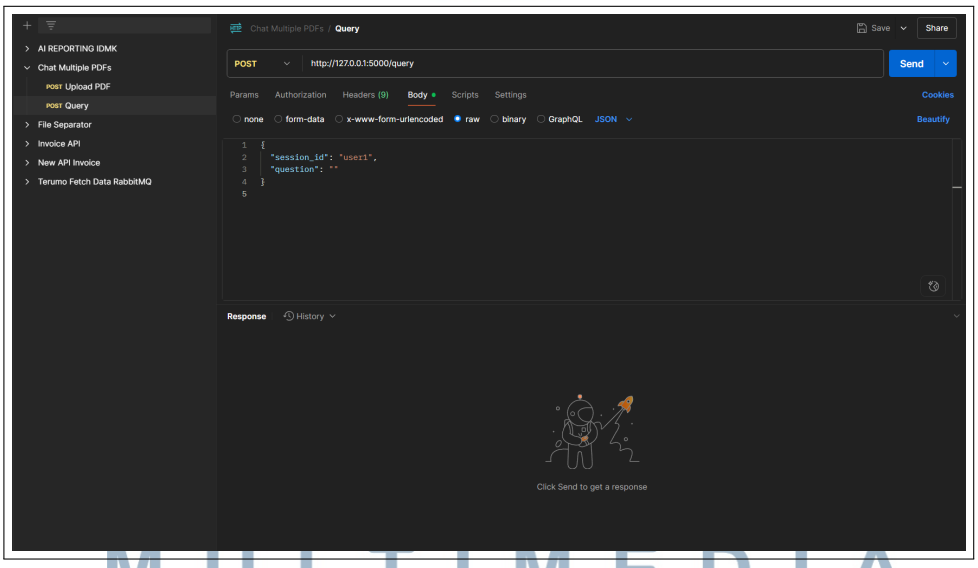
```
    description="Extracted text content"),
12              Property(name="source", data_type=DataType.TEXT,
    description="Source type: pdf"),
13              Property(name="filename", data_type=DataType.TEXT,
    description="PDF filename")
14          ]
15      )
16      vectorstore = WeaviateVectorStore(client=client, index_name="
    Chatbot", text_key="content", embedding=embeddings)
17      pdf_reader = PdfReader(pdf_file)
18      text = ""
19      for page in pdf_reader.pages:
20          extracted_text = page.extract_text()
21          if extracted_text:
22              text += extracted_text + "\n"
23      if not text.strip():
24          return {"error": "No text could be extracted from the PDF"
    }
25      text_splitter = RecursiveCharacterTextSplitter(chunk_size
    =1000, chunk_overlap=0)
26      chunks = text_splitter.split_text(text=text)
27      metadatas = [{"source": "pdf", "filename": pdf_file.filename}]
     * len(chunks)
28      vectorstore.add_texts(chunks, metadatas=metadatas)
29      return {"message": f"PDF '{pdf_file.filename}' successfully
    stored in the vector database"}
30
31  def query_chatbot(question):
32      llm = OpenAI()
33      embeddings = OpenAIEmbeddings()
34      vectorstore = WeaviateVectorStore(
35          client=client, index_name="Chatbot", text_key="content",
    embedding=embeddings
36      )
37      docs = vectorstore.similarity_search(question, k=4)
38      if not docs:
39          return {"answer": "I couldn't find relevant information in
     the uploaded PDF to answer your question."}
40      read_chain = load_qa_chain(llm=llm)
41      answer = read_chain.run(input_documents=docs, question=
    question)
```

```
42     return {"answer": answer}
```

Kode 3.4: Pengaturan untuk penyimpanan dokumen dan penerimaan pertanyaan dalam file backend.py



Gambar 3.5. Tampilan API dengan *endpoint* POST /upload

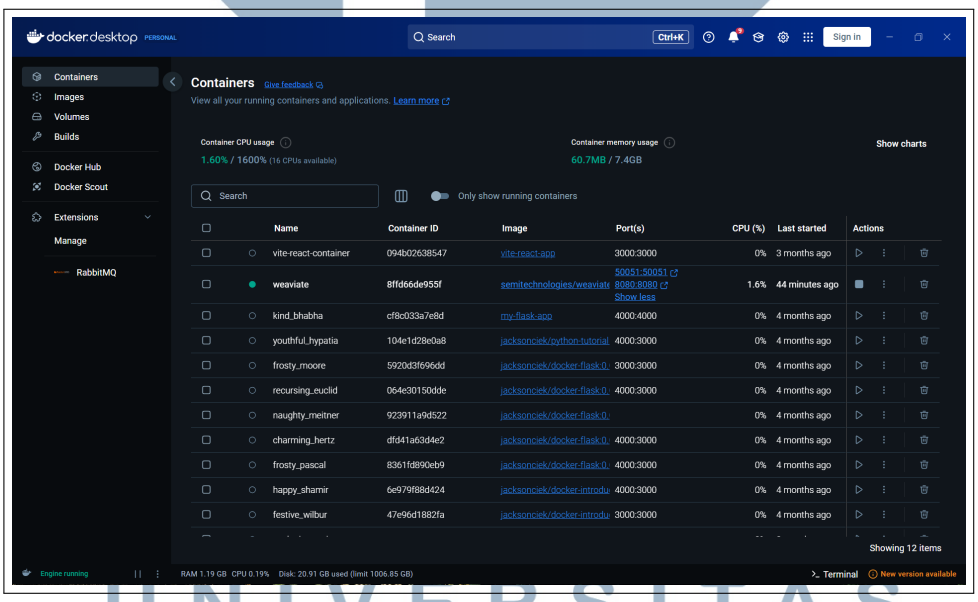


Gambar 3.6. Tampilan API dengan *endpoint* POST /query

MULTIMEDIA NUSANTARA

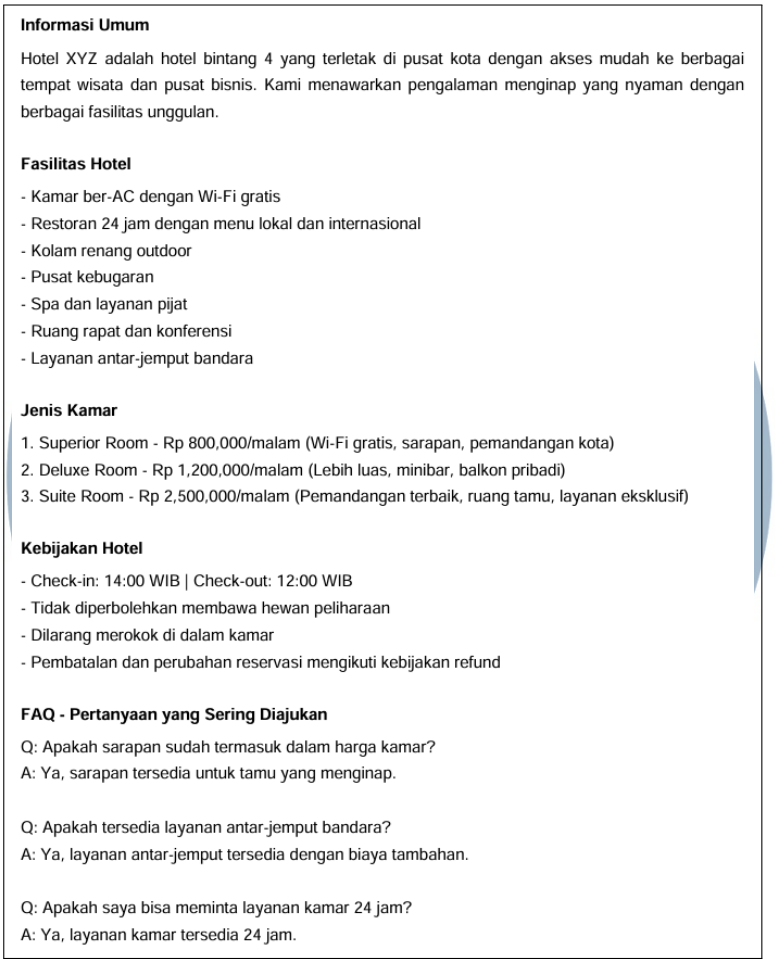
Untuk mengembangkan *chatbot* dasar yang hanya menerima satu berkas PDF dan satu *query*, diperlukan adanya sebuah dokumen terlebih dahulu. Contoh isi dokumen tersebut dapat dilihat pada Gambar 3.8. Selanjutnya, dilakukan pemanggilan API `POST /upload` yang memberikan respons 200 sebagai indikasi

keberhasilan unggah dokumen ke dalam *knowledge base chatbot*, seperti yang ditunjukkan pada Gambar 3.9. Apabila terdapat dua file atau lebih pada saat melakukan upload di API, maka respons yang dihasilkan adalah berupa `Error 400` karena pada kondisi ini hanya menerima satu *file* PDF sebagaimana yang tampak pada Gambar 3.10. Proses tersebut kemudian diikuti dengan pertanyaan dari pengguna serta pemberian jawaban yang sesuai, sebagaimana ditampilkan pada Gambar 3.11. Sebagai contoh, pertanyaannya adalah ”Apa layanan yang diberikan hotel XYZ yang berbintang 4 itu?”, di mana respons yang diberikan tentunya sesuai dengan isi dokumen pada Gambar 3.8. Apabila dilakukan mekanisme *follow-up questions* dengan memberikan pertanyaan ”Eh aku sebelumnya tanya tentang apa sih? Saya pelupa soalnya” tentu akan menghasilkan respons yang tidak diinginkan karena masih belum ada kemampuan *memory management* untuk menyimpan hasil percakapan sebelumnya. Hal ini dapat dilihat pada Gambar 3.12. Namun, untuk memanfaatkan Weaviate sebagai *vector database* yang mendukung operasi *chatbot*, sistem tersebut perlu diaktifkan terlebih dahulu yang dapat dilihat pada Gambar 3.7.



Gambar 3.7. Aktivasi Weaviate di Docker

UNIVERSITAS MULTIMEDIA NUSANTARA



Informasi Umum

Hotel XYZ adalah hotel bintang 4 yang terletak di pusat kota dengan akses mudah ke berbagai tempat wisata dan pusat bisnis. Kami menawarkan pengalaman menginap yang nyaman dengan berbagai fasilitas unggulan.

Fasilitas Hotel

- Kamar ber-AC dengan Wi-Fi gratis
- Restoran 24 jam dengan menu lokal dan internasional
- Kolam renang outdoor
- Pusat kebugaran
- Spa dan layanan pijat
- Ruang rapat dan konferensi
- Layanan antar-jemput bandara

Jenis Kamar

1. Superior Room - Rp 800,000/malam (Wi-Fi gratis, sarapan, pemandangan kota)
2. Deluxe Room - Rp 1,200,000/malam (Lebih luas, minibar, balkon pribadi)
3. Suite Room - Rp 2,500,000/malam (Pemandangan terbaik, ruang tamu, layanan eksklusif)

Kebijakan Hotel

- Check-in: 14:00 WIB | Check-out: 12:00 WIB
- Tidak diperbolehkan membawa hewan peliharaan
- Dilarang merokok di dalam kamar
- Pembatalan dan perubahan reservasi mengikuti kebijakan refund

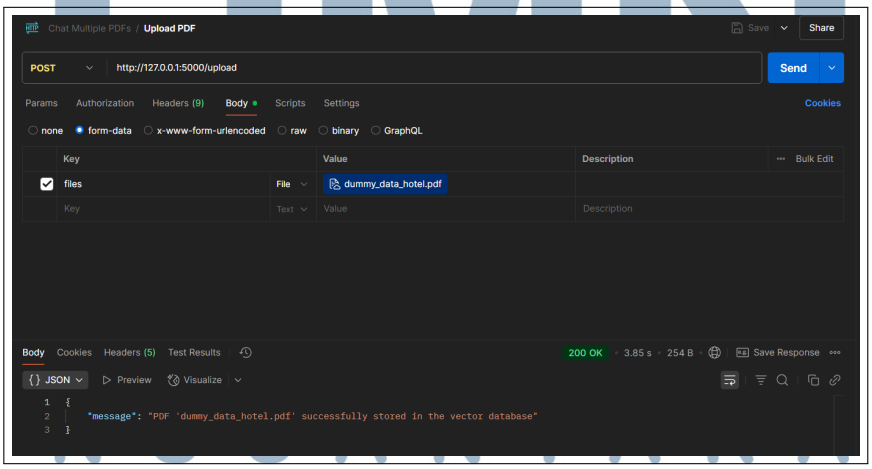
FAQ - Pertanyaan yang Sering Diajukan

Q: Apakah sarapan sudah termasuk dalam harga kamar?
A: Ya, sarapan tersedia untuk tamu yang menginap.

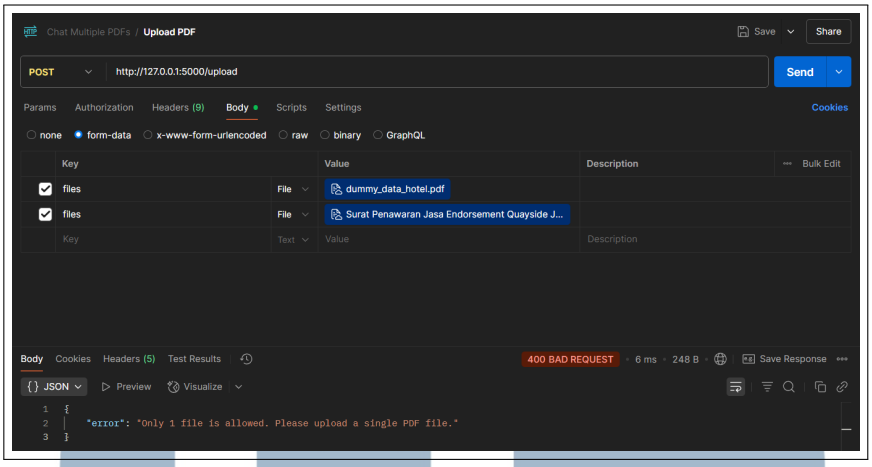
Q: Apakah tersedia layanan antar-jemput bandara?
A: Ya, layanan antar-jemput tersedia dengan biaya tambahan.

Q: Apakah saya bisa meminta layanan kamar 24 jam?
A: Ya, layanan kamar tersedia 24 jam.

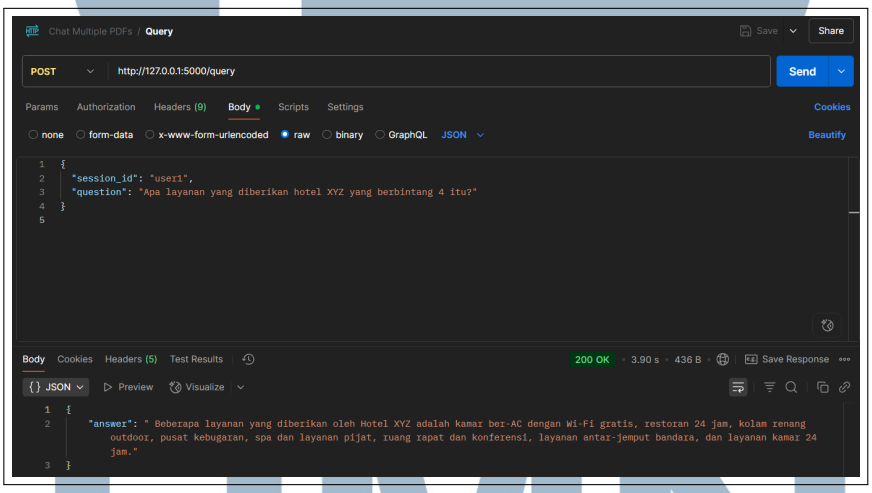
Gambar 3.8. Contoh dokumen sebagai sumber pengetahuan *chatbot*



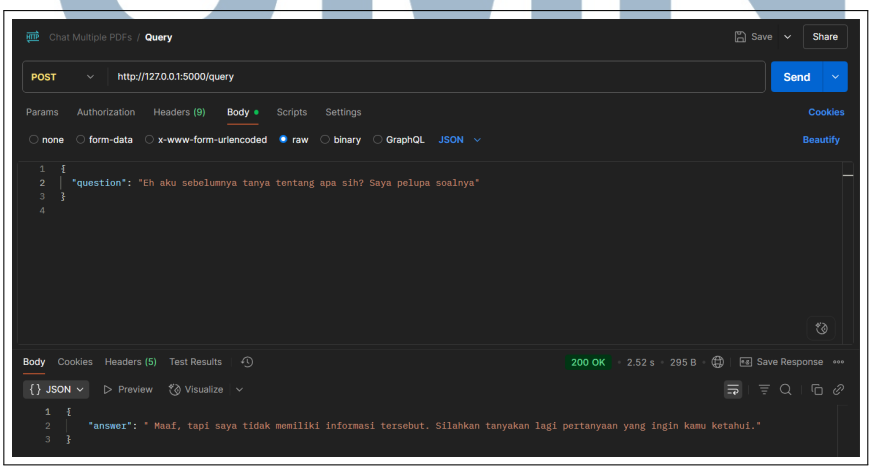
Gambar 3.9. Tampilan pengunggahan satu dokumen berhasil



Gambar 3.10. Tampilan *error* saat mengunggah beberapa dokumen



Gambar 3.11. Tampilan untuk memberikan pertanyaan



Gambar 3.12. Tampilan untuk memberikan pertanyaan lanjutan

Selanjutnya, proses ini akan dimodifikasi dengan mengimplementasikan fitur yang mampu menerima beberapa *file* PDF sekaligus serta menangani *multiple query* dari pengguna dalam bentuk *array*. Modifikasi untuk menerima *multiple file* PDF dilakukan dengan merevisi bagian *method* `store_pdf()` pada potongan kode 3.5 dengan menambahkan sebuah `for-loop` tambahan dan *method* `upload_pdf()` pada potongan kode 3.6. Sementara itu, penanganan *multiple query* dalam bentuk *array* diimplementasikan melalui modifikasi pada *method* `query` yang terdapat pada potongan kode 3.7. Contoh tampilan untuk mengunggah banyak dokumen *file* PDF dan memberikan beberapa pertanyaan masing-masing dapat dilihat pada Gambar 3.14 dan 3.15. Sebagai tambahan, Gambar 3.13 menunjukkan satu *file* PDF tambahan saat melakukan pengunggahan dokumen.

```
def store_pdf(pdf_files):
    embeddings = OpenAIEmbeddings()
    client.collections.delete_all()
    client.collections.create(
        name="Chatbot",
        description="Documents and Conversations for chatbot",
        vectorizer_config=Configure.Vectorizer.text2vec_openai(model="ada", type_="text"),
        properties=[
            Property(name="content", data_type=DataType.TEXT, description="Extracted text content"),
            Property(name="source", data_type=DataType.TEXT, description="Source type: 'pdf' or 'conversation'"),
            Property(name="filename", data_type=DataType.TEXT, description="PDF filename (if applicable)")
        ]
    )
    vectorstore = WeaviateVectorStore(client=client, index_name="Chatbot", text_key="content", embedding=embeddings)
    all_texts = []
    all_metadatas = []
    for pdf in pdf_files:
        pdf_reader = PdfReader(pdf)
        text = ""
        for page in pdf_reader.pages:
            extracted_text = page.extract_text()
            if extracted_text:
                text += extracted_text + "\n"
        if not text.strip():
            continue
```

```
        text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=0)
        chunks = text_splitter.split_text(text=text)
        all_texts.extend(chunks)
        all_metadatas.extend([{"source": "pdf", "filename": pdf.filename}] * len(chunks))
    vectorstore.add_texts(all_texts, metadatas=all_metadatas)
    return {"message": "All PDFs successfully stored in the vector database"}
```

Kode 3.5: Mengatasi pengunggahan beberapa dokumen pada store_pdf() dalam file backend.py

```
def upload_pdf():
    if "files" not in request.files:
        return jsonify({"error": "No file part"}), 400
    files = request.files.getlist("files")
    if not files or any(f.filename == "" for f in files):
        return jsonify({"error": "No selected files"}), 400
    for file in files:
        if not file.filename.lower().endswith('.pdf'):
            return jsonify({"error": "Only PDF files are allowed"}), 400
    response = store_pdf(files)
    return jsonify(response)
```

Kode 3.6: Mengatasi pengunggahan beberapa dokumen pada upload_pdf() dalam file main.py

```
def query():
    data = request.get_json()
    questions = data.get("questions")
    if not questions or not isinstance(questions, list):
        return jsonify({"error": "Missing questions array or questions is not a list"}), 400
    answers = []
    for question in questions:
        if not question or not isinstance(question, str):
            answers.append({"question": question, "answer": "Invalid question format"})
            continue
        answer = query_chatbot(question)
        answers.append({"question": question, "answer": answer["answer"]})
    return jsonify({"answers": answers})
```

Kode 3.7: Mengatasi beberapa pertanyaan pada query() dalam file main.py

Dengan hormat,

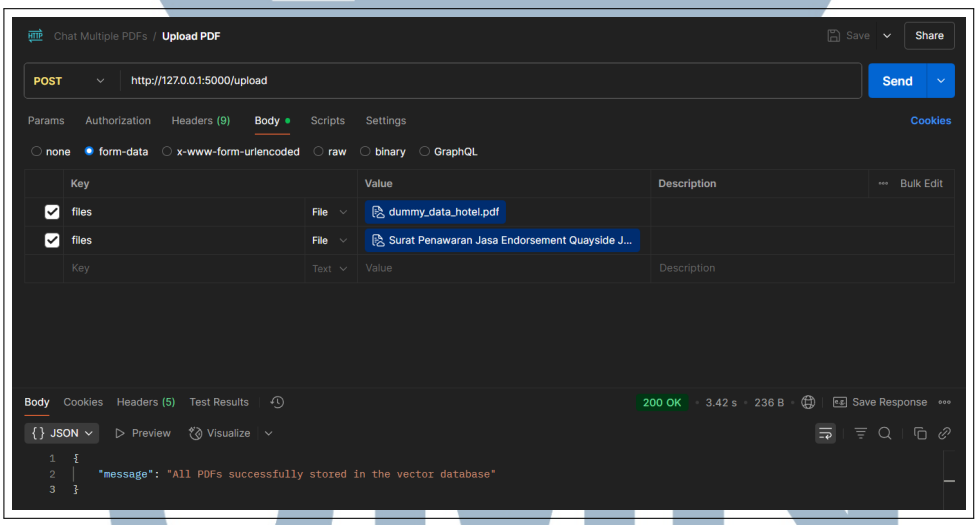
Kami Lina, Marlina, & Jessica, dengan senang hati menawarkan kerjasama endorsement untuk meningkatkan visibilitas dan engagement Ouayside JBCC di media sosial, khususnya Instagram.

Adapun rincian penawaran kami adalah sebagai berikut:

📌 **Platform:** Instagram (Reels & Stories)
📌 **Durasi Kontrak:** 2 bulan
📌 **Konten yang Ditawarkan:**

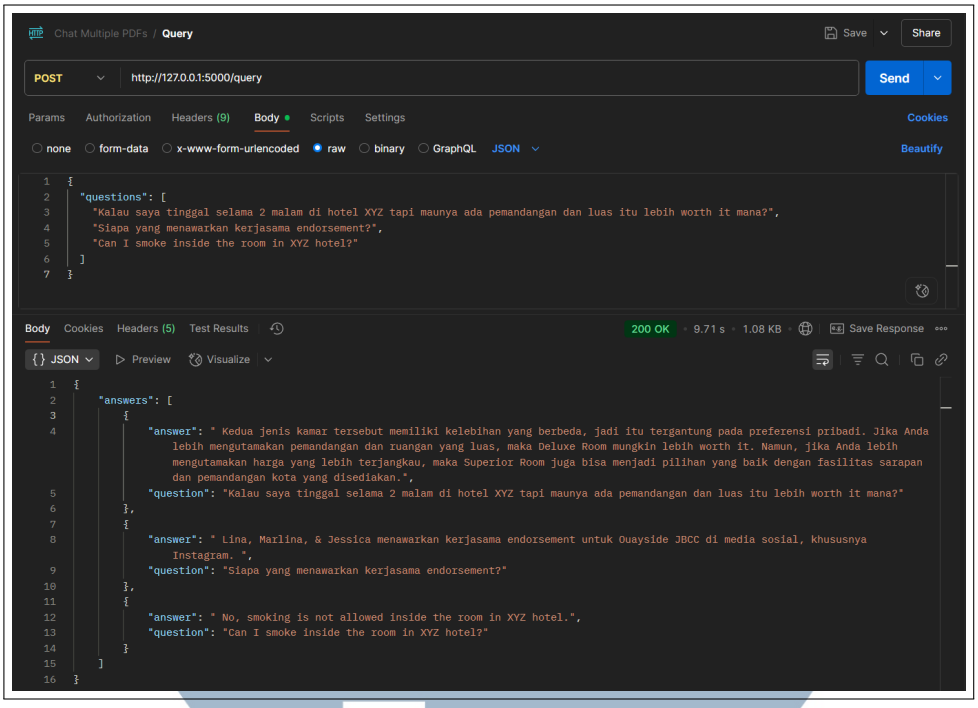
- 1 Reels setiap bulan (total 2 Reels)
- 1–2 Instagram Stories setiap minggu (total 8–16 Stories)

Gambar 3.13. Contoh dokumen tambahan



Gambar 3.14. Tampilan mengunggah banyak *file*

UNIVERSITAS MULTIMEDIA NUSANTARA



Gambar 3.15. Tampilan memberikan banyak pertanyaan

Pengembangan selanjutnya dari fitur penanganan *multiple query* adalah implementasi kemampuan *follow-up questions*. Proses ini dilakukan melalui beberapa modifikasi sistem, antara lain penambahan *method* `store_conversation()` untuk penyimpanan histori percakapan dan revisi pada *method* `query_chatbot()` agar dapat memproses pertanyaan lanjutan, seperti terlihat pada Kode 3.9. Struktur **array** sebelumnya dihilangkan sebagaimana ditunjukkan dalam Kode 3.10. Selain itu, sistem memerlukan inisialisasi variabel memori percakapan dan modifikasi *method* `query()` untuk memungkinkan pertanyaan lanjutan langsung tanpa melalui array (Kode 3.11). Tentu hal ini akan memanfaatkan *library* tambahan yang terdapat pada potongan kode 3.8 untuk kedua *file* `main.py` dan `backend.py`. Hasil implementasi menunjukkan keberhasilan fungsi *follow-up questions* (Gambar 3.16 dan 3.17), sementara tampilan unggah banyak *file* tetap konsisten dengan implementasi sebelumnya (Gambar 3.14).

```
from langchain.memory import ConversationBufferMemory
from langchain.schema import HumanMessage, AIMessage
```

Kode 3.8: Tambahan library untuk menyimpan memori percakapan untuk kedua file

```
def store_conversation(question, answer):
    embeddings = OpenAIEmbeddings()
    vectorstore = WeaviateVectorStore(client=client, index_name="Chatbot", text_key="content", embedding=embeddings)
    conversation_texts = [f"User: {question}", f"Chatbot: {answer}"]
    conversation_metadata = [{"source": "conversation"}, {"source": "conversation"}]
    vectorstore.add_texts(conversation_texts, metadatas=conversation_metadata)
```

Kode 3.9: Menyimpan memori percakapan dengan membuat store_conversation() dalam file backend.py

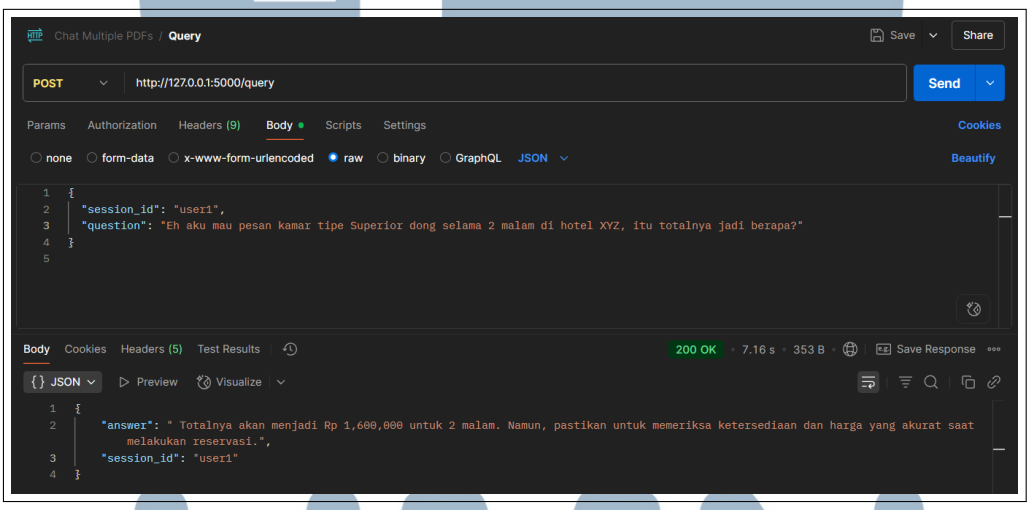
```
def query_chatbot(question, session_id, memory):
    llm = OpenAI()
    embeddings = OpenAIEmbeddings()
    vectorstore = WeaviateVectorStore(client=client, index_name="Chatbot", text_key="content", embedding=embeddings)
    chatbot_collection = client.collections.get("Chatbot")
    stored_messages = memory.chat_memory.messages
    context = " ".join([f"Q: {msg.content}" if isinstance(msg, HumanMessage) else f"A: {msg.content}" for msg in stored_messages])
    full_query = f"Context: {context} Current Question: {question}"
    search_results = chatbot_collection.query.near_vector(near_vector=embeddings.embed_query(full_query), distance=0.7, limit=4)
    docs = vectorstore.similarity_search(full_query, k=4)
    read_chain = load_qa_chain(llm=llm)
    answer = read_chain.run(input_documents=docs, question=question)
    store_conversation(question, answer)
    return {"answer": answer}
```

Kode 3.10: Memproses beberapa pertanyaan beserta pertanyaan lanjutan pada query_chatbot() dalam file backend.py

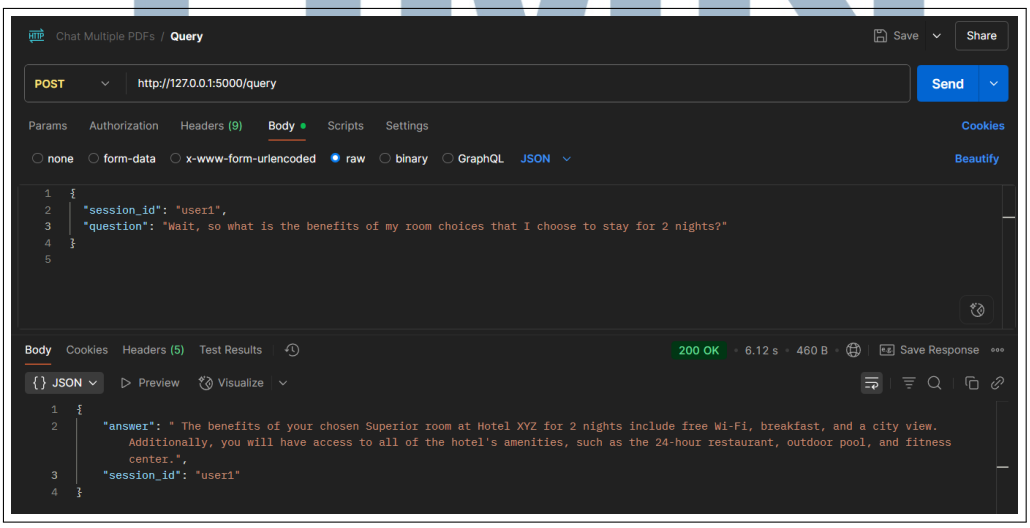
```
def query():
    data = request.get_json()
    session_id = data.get("session_id")
    question = data.get("question")
    if not question or not session_id:
        return jsonify({"error": "Missing session_id or question"}), 400
```

```
    if session_id not in conversation_memory:
        conversation_memory[session_id] = ConversationBufferMemory()
    memory = conversation_memory[session_id]
    answer = query_chatbot(question, session_id, memory)
    memory.chat_memory.add_message(HumanMessage(content=question))
    memory.chat_memory.add_message(AIMessage(content=answer["answer"]))
    return jsonify({"session_id": session_id, "answer": answer["answer"]})
```

Kode 3.11: Memberikan mekanisme pertanyaan lanjutan pada query() dalam file main.py



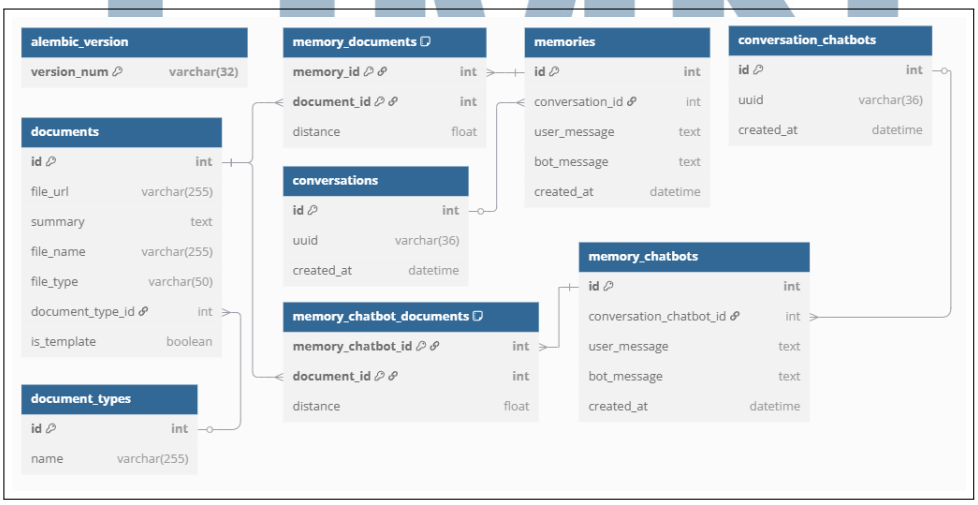
Gambar 3.16. Memberikan pertanyaan pertama



Gambar 3.17. Memberikan pertanyaan lanjutan sebagai mekanisme *follow-up questions*

Implementasi awal *chatbot* yang telah diselesaikan akan melalui proses evaluasi menyeluruh oleh supervisor dan *Technical Team Manager* untuk dilakukan analisis komparatif terhadap sistem *chatbot* yang saat ini beroperasi di perusahaan AI.DECE. Hasil dari proses evaluasi ini akan menjadi dasar untuk menentukan arahan pengembangan lebih lanjut. Berdasarkan temuan dari proses *review* tersebut, pengembangan platform akan dilanjutkan dengan fokus pada penyempurnaan aspek-aspek yang telah diidentifikasi memerlukan perbaikan atau pengayaan fitur. Seluruh rangkaian pengembangan lanjutan ini akan diuraikan secara komprehensif dalam Subbab 3.5.2, yang akan mencakup detail *Activity Diagram*, implementasi kode, dan pengujian beserta hasilnya.

Akan tetapi, sebelum melakukan pengembangan fitur *AI Smart Searching* dan *AI Grouping*, serta pengembangan lanjutan fitur *AI Smart Chatbot*, perlu dirancang terlebih dahulu sebuah sistem basis data untuk menyimpan kategori atau *folder* yang dibuat, dokumen yang diunggah, memori percakapan dan riwayat percakapan untuk fitur *AI Smart Chatbot* dan *AI Smart Searching*, serta versi `alembic` yang merupakan sistem integrasi antara Flask dengan MySQL. Adapun *Entity Relationship Diagram* (ERD) yang bertujuan untuk memberikan gambaran mengenai jalan kerja *Database Management System* beserta dengan tabel, atribut dengan `id`-nya selalu memiliki *constraint* `AUTO_INCREMENT`, dan tipe datanya, dapat dilihat pada Gambar 3.18 dan penjelasan di bawahnya. Setelah pembuatan sistem basis data ini, maka fitur *AI Smart Chatbot* yang telah dibuat harus ditambahkan mekanisme-mekanisme penyimpanan memori yang disebutkan di atas.



Gambar 3.18. *Entity Relationship Diagram* untuk Platform *AI Mastering Document*

Tabel 3.5. Struktur Tabel `alembic_version`

Atribut	Tipe Data	Constraint
`version_num`	varchar(32)	PK

Penjelasan atribut Tabel `alembic_version` yaitu sebagai berikut.

- `version_num`: Menyimpan *identifier* unik untuk versi migrasi basis data yang sedang berjalan.

Tabel 3.5 mendeskripsikan versi Alembic yang digunakan untuk melakukan integrasi antara MySQL dengan Flask oleh SQLAlchemy, serta atribut di dalam tabel ini tidak memiliki relasi (referensi) dengan tabel lain dalam skema ini.

Tabel 3.6. Struktur Tabel `conversations`

Atribut	Tipe Data	Constraint
`id`	int	PK
`uuid`	varchar(36)	
`created_at`	datetime	

Penjelasan atribut Tabel `conversations` yaitu sebagai berikut.

- `id`: Kunci primer untuk masing-masing sesi percakapan.
- `uuid`: *Identifier global* unik untuk sesi percakapan.
- `created_at`: Waktu pembuatan sesi percakapan.

Tabel 3.6 berguna untuk mengelompokkan memori di tabel `memories` berdasarkan sesi percakapan pada bagian fitur menu *AI Smart Searching*.

Tabel 3.7. Struktur Tabel `conversation_chatbots`

Atribut	Tipe Data	Constraint
`id`	int	PK
`uuid`	varchar(36)	
`created_at`	datetime	

Penjelasan atribut Tabel `conversation_chatbots` yaitu sebagai berikut.

- `id`: Kunci primer untuk setiap percakapan *chatbot*.
- `uuid`: *Identifier* unik global untuk percakapan *chatbot*.
- `created_at`: Waktu pembuatan percakapan *chatbot*.

Tabel 3.7 digunakan untuk menyimpan sesi percakapan *chatbot* pada bagian fitur menu *AI Smart Chatbot* dan menjadi acuan relasi pada tabel `memory_chatbots`.

Tabel 3.8. Struktur Tabel `document_types`

Atribut	Tipe Data	Constraint
`id`	int	PK
`name`	varchar(255)	

Penjelasan atribut Tabel `document_types` yaitu sebagai berikut.

- `id`: Kunci primer untuk jenis dokumen.
- `name`: Nama atau label jenis dokumen (Misalnya: “invoice”, ”SOP”, “report”).

Tabel 3.8 menjadi acuan untuk kolom `document_type_id` di tabel `documents`, memastikan setiap dokumen punya tipe yang valid sebagai pembuatan kategori atau *folder*.

Tabel 3.9. Struktur Tabel `documents`

Atribut	Tipe Data	Constraint
`id`	int	PK
`file_url`	varchar(255)	
`summary`	text	
`file_name`	varchar(255)	
`file_type`	varchar(50)	
`document_type_id`	int	FK → Tabel 3.8
`is_template`	boolean	

Penjelasan atribut Tabel `documents` yaitu sebagai berikut.

- `id`: Kunci primer tiap dokumen.
- `file_url`: Lokasi penyimpanan *file* dokumen pada Cloudinary.
- `summary`: Ringkasan isi dokumen.
- `file_name`: Nama *file* asli.
- `file_type`: Format dokumen.
- `document_type_id`: Mengacu ke jenis dokumen di 3.8.
- `is_template`: Penanda format *file*.

Tabel 3.9 menunjukkan informasi dari dokumen yang diunggah dan direferensikan oleh tabel `memory_documents` dan `memory_chatbot_documents` untuk mengaitkan dokumen dengan entitas memori.

Tabel 3.10. Struktur Tabel `memories`

Atribut	Tipe Data	Constraint
`id`	int	PK
`conversation_id`	int	FK → Tabel 3.6
`user_message`	text	
`bot_message`	text	
`created_at`	datetime	

Penjelasan atribut Tabel `memories` yaitu sebagai berikut.

- `id`: Kunci primer untuk tiap memori.
- `conversation_id`: Mengacu ke sesi percakapan di 3.6.
- `user_message`: Pesan yang dikirim pengguna.
- `bot_message`: Balasan yang dihasilkan bot.
- `created_at`: Waktu pencatatan memori.

UNIVERSITAS MULTIMEDIA NUSANTARA

Tabel 3.10 menjadi dasar untuk tabel `memory_documents` dalam menyambungkan dokumen dengan memori tertentu dan memori ini merupakan bagian pada fitur menu *AI Smart Searching*.

Tabel 3.11. Struktur Tabel `memory_chatbots`

Atribut	Tipe Data	Constraint
`id`	int	PK
`conversation_chatbot_id`	int	FK → Tabel 3.7
`user_message`	text	
`bot_message`	text	
`created_at`	datetime	

Penjelasan atribut Tabel `memory_chatbots` yaitu sebagai berikut.

- `id`: Kunci primer memori *chatbot*.
- `conversation_chatbot_id`: Mengacu ke sesi *chatbot* di Tabel 3.7.
- `user_message`: Pesan pengguna pada sesi *chatbot*.
- `bot_message`: Respon *chatbot*.
- `created_at`: Waktu pencatatan memori *chatbot*.

Pada tabel 3.11 menjadi basis untuk `memory_chatbot_documents` dalam merekam relasi dokumen dan memori ini merupakan bagian pada fitur menu *AI Smart Chatbot*.

Tabel 3.12. Struktur Tabel `memory_chatbot_documents`

Atribut	Tipe Data	Constraint
`memory_chatbot_id`	int	PK, FK → Tabel 3.11
`document_id`	int	PK, FK → Tabel 3.9
`distance`	float	

Penjelasan atribut Tabel `memory_chatbot_documents` yaitu sebagai berikut.

- `memory_chatbot_id`: Mengaitkan dokumen ke entri `memory_chatbots`.
- `document_id`: Mengaitkan entri `memory_chatbots` ke dokumen di 3.9.

- `distance`: Skor jarak/kemiripan antara memori *chatbot* dan dokumen.

Tabel 3.12 adalah tabel yang menghubungkan `memory_chatbots` dengan `documents`, beserta skor kemiripannya untuk dikeluarkan hasil keluarannya yang paling sesuai berdasarkan pencarian semantik pada bagian fitur menu *AI Smart Chatbot*.

Tabel 3.13. Struktur Tabel `memory_documents`

Atribut	Tipe Data	Constraint
`memory_id`	int	PK, FK → Tabel 3.10
`document_id`	int	PK, FK → Tabel 3.9
`distance`	float	

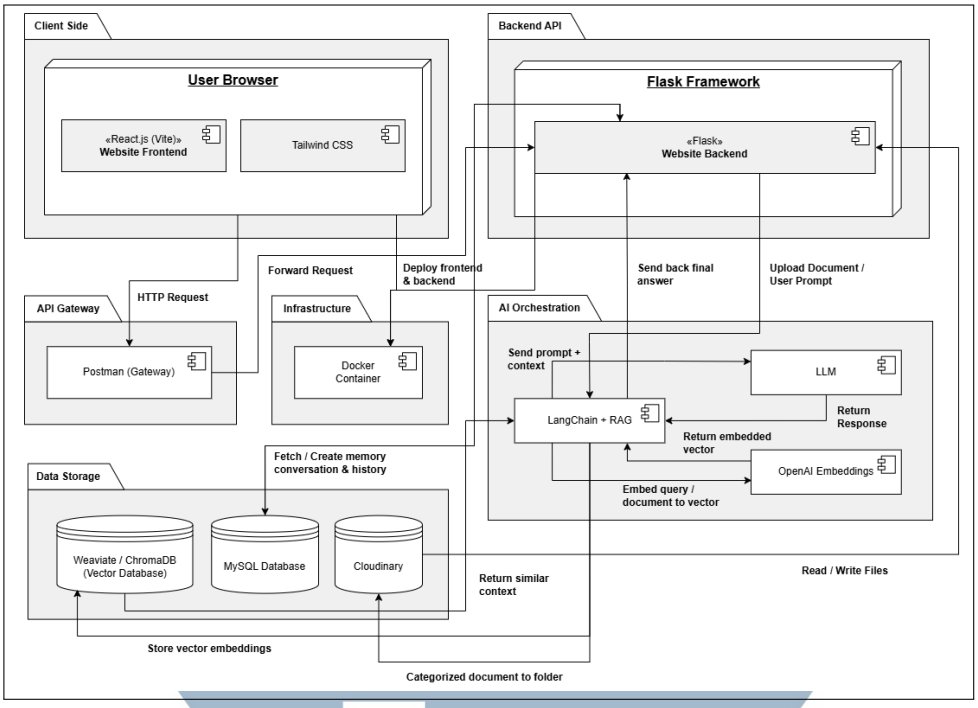
Penjelasan atribut Tabel `memory_documents` yaitu sebagai berikut.

- `memory_id`: Mengaitkan dokumen ke entri `memories`.
- `document_id`: Mengaitkan entri `memories` ke dokumen di Tabel 3.9.
- `distance`: Skor jarak atau kemiripan antara memori dan dokumen.

Tabel 3.13 berfungsi sebagai tabel yang menghubungkan `memories` dengan `documents`, beserta skor kemiripannya untuk dikeluarkan hasil keluarannya yang paling sesuai berdasarkan pencarian semantik pada bagian fitur menu *AI Smart Searching*.

2) Diagram Arsitektur

Diagram arsitektur adalah gambaran visual yang menampilkan komponen-komponen utama dalam suatu perangkat lunak beserta keterkaitan di antara komponen-komponen tersebut. Tujuannya adalah untuk memberikan klarifikasi mengenai struktur teknis sistem, mencakup hubungan antara berbagai elemen seperti antarmuka pengguna, server, basis data, dan manajemen konten. Keberadaan diagram ini sangat krusial dalam memahami aliran data serta proses yang berlangsung dalam sistem, sekaligus mempermudah perancangan, pengembangan, dan perawatan sistem dengan lebih efektif [81]. Adapun diagram arsitektur yang dirancang pada Platform *AI Mastering Document* dapat dilihat pada Gambar 3.19.



Gambar 3.19. Diagram Arsitektur Platform AI Mastering Document

Platform *AI Mastering Document* merupakan produk unggulan dari perusahaan AI.DECE yang dirancang untuk mengotomatisasi pengelolaan dokumen dengan kecerdasan buatan. Sistem ini menggabungkan berbagai teknologi modern seperti React.js dengan Vite dan Tailwind CSS di sisi klien, serta Flask sebagai kerangka kerja *backend* untuk membangun API dan logika aplikasi. Untuk mendukung kebutuhan orkestra AI, platform ini mengintegrasikan pendekatan *Retrieval-Augmented Generation* (RAG), LangChain, dan OpenAIEmbeddings yang memungkinkan pencarian dan pemrosesan konteks yang relevan secara efisien. Sistem ini juga memanfaatkan layanan penyimpanan vektor seperti ChromaDB atau Weaviate serta penyimpanan objek berbasis *cloud* melalui Cloudinary.

Secara umum, sistem terbagi ke dalam dua sisi utama yaitu *Client Side* dan *Backend API*. Di sisi klien, pengguna berinteraksi melalui antarmuka *frontend* yang dibangun menggunakan React.js dengan Vite dan Tailwind CSS. Permintaan pengguna seperti unggah dokumen atau pengiriman *prompt* dikirim melalui API Gateway menggunakan Postman yang berfungsi sebagai penghubung awal untuk pengujian dan pengelolaan permintaan HTTP. Permintaan tersebut kemudian diteruskan ke infrastruktur *backend* yang berjalan di dalam Docker untuk memastikan isolasi lingkungan dan kemudahan *deployment*.

Di sisi *Backend API*, permintaan pengguna ditangani oleh server berbasis Flask, yang menjadi pusat dari pengelolaan logika aplikasi dan pengolahan permintaan. Ketika pengguna mengunggah dokumen atau memberikan *input prompt*, data tersebut diteruskan ke komponen *AI Orchestration*. Komponen ini bertugas mengelola proses *Retrieval-Augmented Generation* menggunakan LangChain yang akan menggabungkan konteks eksternal ke dalam permintaan pengguna. Untuk keperluan *embedding*, sistem menggunakan OpenAIEmbeddings guna mengubah dokumen maupun pertanyaan ke dalam bentuk vektor numerik. Vektor-vektor tersebut disimpan dalam ChromaDB atau Weaviate sebagai basis data vektor. Jika diperlukan, vektor yang serupa akan diambil dari basis data tersebut untuk dikombinasikan dengan *prompt* pengguna sebelum dikirim ke model LLM (Large Language Model) guna menghasilkan respons akhir.

Selain itu, sistem juga mencatat riwayat interaksi pengguna serta metadata dokumen ke dalam *database* MySQL, dan dokumen asli yang telah dikategorikan akan disimpan secara terstruktur ke dalam *folder* menggunakan Cloudinary. Seluruh proses embedding, penyimpanan, pengambilan konteks, hingga generasi jawaban dikemas dalam arsitektur yang modular dan *containerized* dengan bantuan Docker, sehingga mempermudah pengelolaan versi dan skalabilitas sistem.

Dengan arsitektur ini, platform mampu menjalankan ketiga fitur unggulannya, yaitu *AI Smart Chatbot*, *AI Smart Searching*, dan *AI Grouping*, secara efisien dan dapat diskalakan. Perancangan ini juga memfasilitasi integrasi layanan AI dengan pendekatan *modular orchestration*, memisahkan tanggung jawab antara pemrosesan data, interaksi pengguna, dan manajemen model AI. Pendekatan ini menandai pengembangan signifikan dibandingkan dengan sistem sebelumnya yang tidak mendukung orkestrasi berbasis RAG, integrasi LLM, maupun pengelolaan vektor *embedding*, dan menjadi dasar utama transformasi sistem dari fase *before* menuju fase *after* dalam proyek magang ini.

3) Use Case Diagram

Diagram *Unified Modelling Language*, atau sering dikenal dengan diargam UML adalah bahasa pemodelan menggunakan notasi bersifat standar untuk merancang, memvisualisasikan, dan memberikan dokumentasi sistem informasi berbasis objek. UML menyediakan beragam diagram yang menawarkan perspektif berbeda dalam pengembangan sistem, membantu memahami aspek statis dan dinamis secara visual. Salah satu jenis diagram UML adalah *Use Case Diagram*, yang berfungsi untuk menggambarkan fungsionalitas sistem dari pihak *user*.

Use Case Diagram mendefinisikan interaksi antara pengguna sebagai aktor dengan sistem untuk mencapai tujuan tertentu, sekaligus menjadi alat penting dalam analisis kebutuhan fungsional. Dengan demikian, *Use Case Diagram* membantu menyelaraskan ekspektasi pengguna dan pengembang sejak tahap awal pengembangan perangkat lunak [82]. Oleh karena itu, pengembangan Platform *AI Mastering Document* memiliki *Use Case Diagram* yang dapat dilihat pada Gambar 3.20.

Gambar 3.20. *Use Case Diagram* pada Platform *AI Mastering Document*

Pada Platform *AI Mastering Document* yang dikembangkan oleh perusahaan AI.DECE, terdapat beberapa skenario interaksi pengguna yang dimodelkan dalam bentuk *Use Case Diagram*. Diagram ini menggambarkan hubungan antara aktor bersama dengan sistem serta fitur-fitur utama yang tersedia di dalam platform. Aktor utama dalam sistem ini adalah pengguna yang dapat menjalankan berbagai aktivitas seperti mengelola dokumen, melakukan pencarian dokumen, dan berinteraksi melalui *chatbot* cerdas. *Use Case Diagram* diawali dengan aktivitas pengguna dalam melihat halaman utama (*View Home*) sebagai pintu masuk menuju seluruh fitur yang disediakan. Salah satu fitur sentral adalah *Documents Management*, yaitu proses untuk mengelola dokumen dalam platform. Fitur lainnya yaitu aktor dapat mengakses fitur *AI Smart Chatbot* dan fitur *AI Smart Searching* secara langsung.

Use case Documents Management memiliki tiga relasi `<<extend>>` yang menandakan aktor bersifat opsional saat ingin mengakses *Manage Category (Folder)*, *Upload Document*, dan *AI Grouping Manage Document (Foldering)*. Di sisi *Manage Category (Folder)*, terdapat `<<extend>>` use case *Create Category*, *View Category*, *Edit Category*, dan *Delete Category*, yang merupakan perilaku opsional yang dapat dipicu ketika pengguna ingin mengatur kategori secara terstruktur.

Use case Upload Document dengan relasi `<<include>>` *Manage Category (Folder)* berguna untuk memastikan setiap dokumen baru diasosiasikan dengan *folder* yang sesuai, dan sekaligus relasi `<<include>>` *AI Grouping Manage Document (Foldering)* agar dokumen dapat diproses oleh mekanisme AI setelah diunggah. Di sisi *AI Grouping Manage Document (Foldering)*, terdapat `<<extend>>` *use case View AI Grouping Document List*, *View AI Grouping Document File*, dan *Delete AI Grouping Document*, yang merupakan perilaku opsional yang dapat dipicu ketika pengguna ingin mengatur dokumen secara terstruktur.

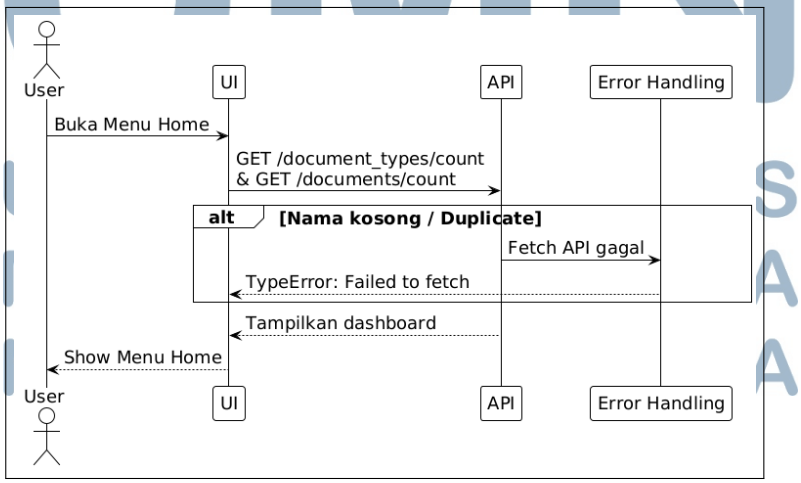
Untuk fitur *AI Smart Chatbot* dan *AI Smart Searching*, relasi `<<extend>>` *Create New Conversation ID* menandakan bahwa setiap sesi percakapan baru mendapatkan identitas unik dan tergantung apakah aktor ingin mengakses fiturnya atau tidak. Kemudian *User's Prompt as Questions* dihubungkan melalui relasi `<<include>>` kepada *use case Create New Conversation ID*, yang menggambarkan alur bahwa obrolan dapat dilakukan apabila sudah ada *Conversation ID*.

Terakhir, *use case View Response* memiliki relasi `<<include>>` pada *use case User's Prompt as Questions* yang berarti harus ada *prompt* dari pengguna

terlebih dahulu untuk melihat hasil respons dari obrolan yaitu pada *use case View Response*. Hal ini juga berlaku sama pada *use case Follow Up Questions* memiliki relasi `<<include>>` pada *use case View Response* yang menandakan kebutuhan *memory management* untuk menangani pertanyaan lanjutan apabila sudah ada respons dari obrolan yang dilakukan.

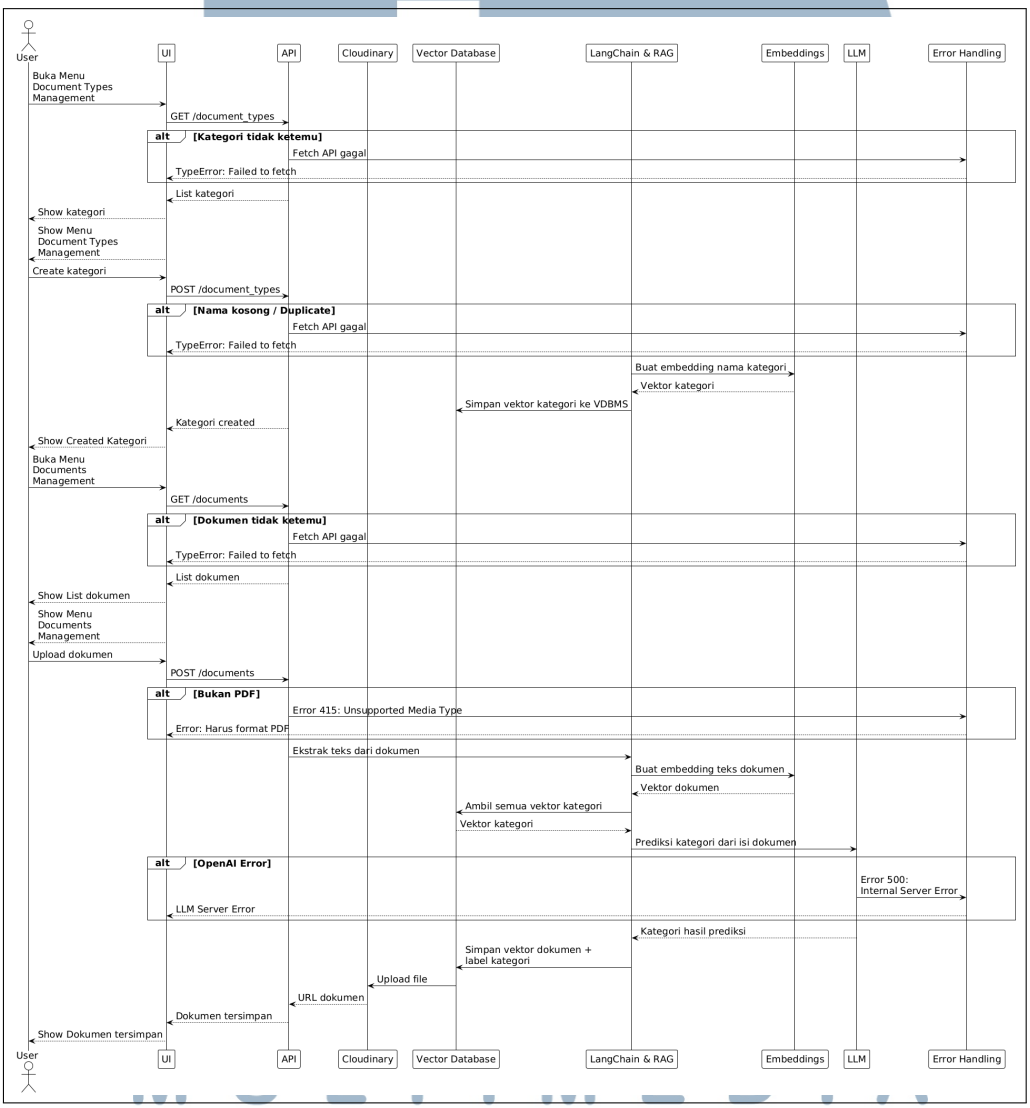
4) Sequence Diagram

Selain *Use Case Diagram*, UML juga menyediakan representasi perilaku sistem atau platform melalui *Sequence Diagram*, yang merupakan diagram UML kedua paling umum digunakan. *Sequence Diagram* menggambarkan interaksi antar objek dalam pertukaran pesan secara kronologis, sehingga membantu pemahaman alur komunikasi selama eksekusi suatu tugas. Diagram ini banyak digunakan pada fase desain yang detail untuk menetapkan protokol komunikasi antarproses secara presisi, sekaligus menjadi acuan dalam pengujian sistem apabila terjadi *error*. Keunggulan *Sequence Diagram* terletak pada sifatnya yang intuitif dan kemampuannya memvisualisasikan perilaku parsial sistem, sehingga lebih mudah dipahami dibandingkan diagram formal seperti *State Chart* [83]. Oleh karena itu, pengembangan Platform *AI Mastering Document* membagikan fitur-fiturnya ke dalam empat menu dengan tiap menu memiliki *Sequence Diagram* tersendiri, seperti Menu *Home* (Gambar 3.21), Menu *Documents Management* beserta *Document Types Management* (Gambar 3.22) dengan implementasi fitur *AI Grouping*, Menu *AI Smart Chatbot* (Gambar 3.23) dengan implementasi fitur *AI Smart Chatbot*, dan Menu *AI Smart Searching* (Gambar 3.24) dengan implementasi fitur *AI Smart Searching*.



Gambar 3.21. *Sequence Diagram* pada Menu *Home*

Gambar 3.21 menunjukkan *Sequence Diagram* untuk menampilkan informasi-informasi yang berguna untuk Platform *AI Mastering Document* seperti total dokumen yang telah diunggah dan total kategori atau *folder* yang dibuat. Selain itu, terdapat *error handling* dengan tujuan untuk memberikan notifikasi pesan apabila API-nya gagal untuk dilakukan koneksi. Untuk penjelasan lebih lanjut terkait dengan API *endpoint*-nya akan diutarakan pada Subbab 3.5.2 bagian Tampilan Antarmuka Pengguna yang menjelaskan tentang API Endpoint Postman.



Gambar 3.22. *Sequence Diagram* pada fitur *AI Grouping*

Gambar 3.22 menunjukkan *Sequence Diagram* untuk menampilkan jalan kerja dari fitur *AI Grouping*. Pada gambar tersebut mencerminkan proses alur interaksi antara aktor utama yaitu *user*, dengan berbagai bagian komponen

sistem seperti *UI*, *Backend API (Flask)*, *Cloudinary*, *vector database*, *LangChain dan RAG*, *OpenAIEmbeddings*, *LLM (OpenAI)*, serta *Error Handling*. Setiap komponen memainkan peran spesifik yang membentuk satu kesatuan proses otomasi pengelolaan dokumen berbasis kecerdasan buatan. Untuk penjelasan lebih lanjut terkait dengan API *endpoint*-nya akan diutarakan pada Subbab 3.5.2 bagian Tampilan Antarmuka Pengguna yang menjelaskan tentang API Endpoint Postman.

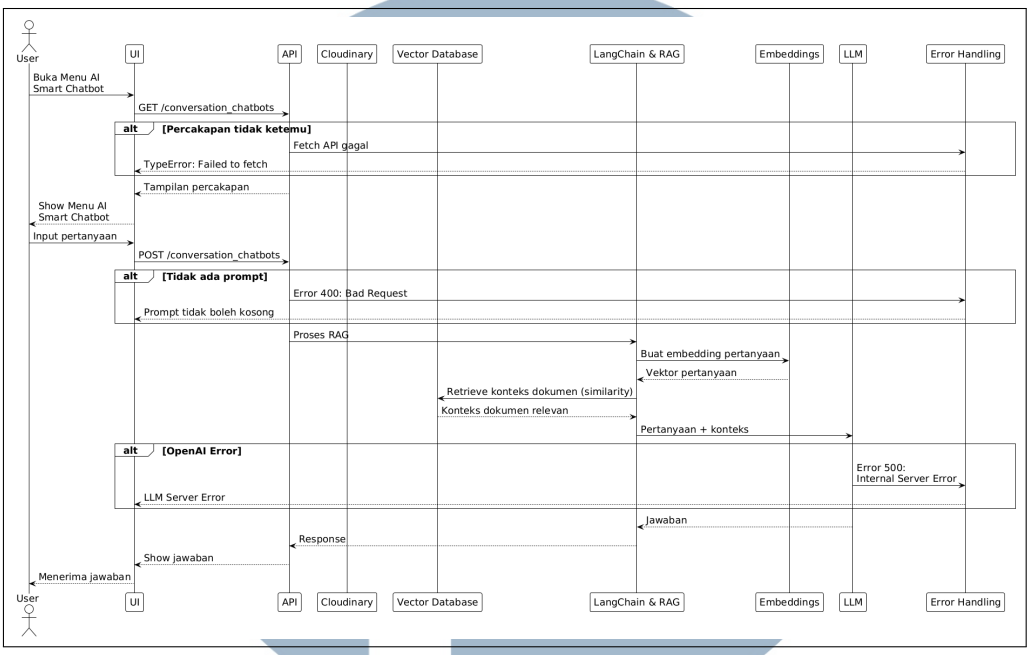
Proses dimulai ketika pengguna membuka halaman utama melalui antarmuka pengguna (*UI*), yang akan merender tampilan beranda. Setelah itu, pengguna mengakses menu *Document Types Management*, yang akan mengirimkan permintaan ke API untuk mendapatkan daftar kategori dokumen melalui `GET /document_types`. API akan memberikan respons berupa daftar kategori yang kemudian ditampilkan kepada pengguna dengan *error handling* yang bertujuan untuk memberikan notifikasi pesan apabila API-nya gagal untuk dilakukan koneksi.

Pengguna kemudian dapat membuat kategori baru melalui permintaan `POST /document_types`. Pada tahap ini, validasi dilakukan untuk memeriksa apakah nama kategori kosong atau duplikat. Jika terjadi kesalahan, komponen *Error Handling* akan mengembalikan pesan "Kategori tidak valid". Jika validasi berhasil, kategori akan diproses dari bentuk teks menjadi vektor oleh OpenAIEmbeddings dan vektor tersebut dialihkan kepada *vector database* untuk disimpan nilai-nilai vektor yang akan digunakan nantinya.

Selanjutnya, pengguna mengunggah dokumen. Sistem hanya menerima dokumen dalam format PDF. Jika format yang diunggah tidak sesuai, maka akan ditangani oleh *Error Handling* dengan mengembalikan kode status `415 Unsupported Media Type`, serta pesan kesalahan "TypeError: Failed to fetch". Jika format valid, dokumen akan diekstraksi dan diproses menggunakan LangChain, dimana isi dokumen dikonversi menjadi vektor menggunakan OpenAIEmbeddings. Vektor dokumen yang terbentuk kemudian dikirim ke LLM OpenAI untuk memprediksi kategori dokumen dan tentunya sebelumnya harus dicocokan dengan vektor kategori agar LLM dapat mengklasifikasikan dokumen yang diunggah masuk ke kategori mana.

Apabila terjadi kesalahan pada proses LLM oleh OpenAI, maka pesan *error* akan ditampilkan kepada pengguna dan tidak dapat berbuat apapun dikarenakan hal tersebut bersifat internal dari pihak perusahaan OpenAI. Jika proses LLM berhasil, hasil prediksi kategori dikembalikan ke LangChain, yang selanjutnya menyimpan *metadata* beserta vektor dokumen ke dalam Weaviate ataupun ChromaDB sebagai basis data vektor. Dokumen secara fisik diunggah ke Cloudinary, dan sistem akan

menerima URL dari dokumen yang tersimpan. Informasi bahwa dokumen telah berhasil disimpan dengan kategori tertentu akan ditampilkan kepada pengguna.

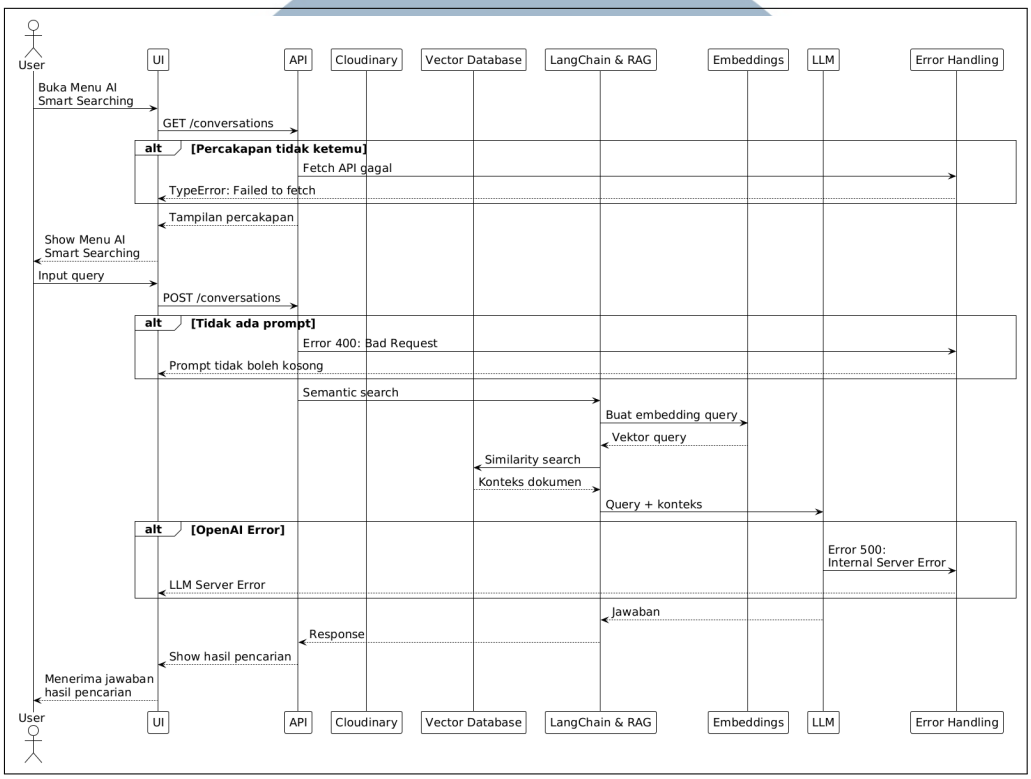


Gambar 3.23. *Sequence Diagram* pada fitur *AI Smart Chatbot*

Gambar 3.23 menunjukkan *Sequence Diagram* untuk menampilkan jalan kerja dari fitur *AI Smart Chatbot*. Pada gambar tersebut mencerminkan proses alur interaksi antara aktor utama yaitu *user*, dengan berbagai bagian komponen sistem seperti *UI*, *Backend API (Flask)*, *Cloudinary*, *vector database*, *LangChain dan RAG*, *OpenAIEmbeddings*, *LLM (OpenAI)*, serta *Error Handling*. Setiap komponen memainkan peran spesifik yang membentuk satu kesatuan proses otomasi pengelolaan dokumen berbasis kecerdasan buatan. Untuk penjelasan lebih lanjut terkait dengan API *endpoint*-nya akan diutarakan pada Subbab 3.5.2 bagian Tampilan Antarmuka Pengguna yang menjelaskan tentang API Endpoint Postman.

Setelah penyimpanan dokumen berhasil yang dilakukan pada fitur *AI Grouping* sebelumnya, pengguna dapat menggunakan fitur *AI Smart Chatbot*. Tampilan untuk memunculkan fitur *AI Smart Chatbot* diambil dengan menggunakan API dan terdapat *error handling* yang bertujuan untuk memberikan notifikasi pesan apabila API-nya gagal untuk dilakukan koneksi. Ketika pengguna memasukkan pertanyaan, sistem akan memprosesnya melalui LangChain dan RAG (Retrieval-Augmented Generation). Pertanyaan dikonversi menjadi vektor menggunakan *OpenAIEmbeddings*, kemudian sistem mengambil konteks dokumen terkait dari *vector database*. Konteks dan pertanyaan tersebut dikirim ke *LLM*

OpenAI untuk menghasilkan jawaban. Jika terjadi kegagalan pada server OpenAI, maka pesan kesalahan akan ditampilkan. Jika berhasil, jawaban dikembalikan secara bertahap melalui *LangChain*, API, dan ditampilkan ke pengguna.



Gambar 3.24. *Sequence Diagram* pada fitur *AI Smart Searching*

Gambar 3.24 menunjukkan *Sequence Diagram* untuk menampilkan jalan kerja dari fitur *AI Smart Searching*. Pada gambar tersebut mencerminkan proses alur interaksi antara aktor utama yaitu *user*, dengan berbagai bagian komponen sistem seperti *UI*, *Backend API (Flask)*, *Cloudinary*, *vector database*, *LangChain dan RAG*, *OpenAIEmbeddings*, *LLM (OpenAI)*, serta *Error Handling*. Setiap komponen memainkan peran spesifik yang membentuk satu kesatuan proses otomasi pengelolaan dokumen berbasis kecerdasan buatan. Untuk penjelasan lebih lanjut terkait dengan API *endpoint*-nya akan diutarakan pada Subbab 3.5.2 bagian Tampilan Antarmuka Pengguna yang menjelaskan tentang API Endpoint Postman.

Setelah penyimpanan dokumen berhasil yang dilakukan pada fitur *AI Grouping* sebelumnya, pengguna juga dapat menggunakan fitur *AI Smart Searching*. Tampilan untuk memunculkan fitur *AI Smart Searching* diambil dengan menggunakan API dan terdapat *error handling* yang bertujuan untuk memberikan notifikasi pesan apabila API-nya gagal untuk dilakukan koneksi.

Pengguna memasukkan *query* pencarian yang kemudian diproses menggunakan pendekatan *semantic search*. LangChain mengonversi *query* menjadi vektor dengan OpenAIEmbeddings, dan mencari kemiripan vektor dalam *vector database* seperti ChromaDB atau Weaviate menggunakan metode *vector similarity search*. Konteks hasil pencarian dikirim ke LLM OpenAI untuk mendapatkan jawaban akhir. Jika terjadi kegagalan pada server OpenAI, maka pesan kesalahan akan ditampilkan. Apabila berhasil, jawaban dan hasil pencarian dikembalikan secara bertahap melalui *LangChain*, API, dan ditampilkan ke pengguna.

Penggunaan pendekatan RAG (Retrieval-Augmented Generation) pada platform ini merupakan inovasi penting yang menggabungkan kemampuan pencarian dari sistem basis data vektor dan kemampuan generatif dari LLM OpenAI, sehingga jawaban yang diberikan bersifat kontekstual dan lebih akurat. Sistem ini tidak lagi bergantung pada pencarian berbasis kata kunci, melainkan sudah memanfaatkan representasi semantik berbasis *vector embeddings*. Selain itu, dengan memanfaatkan *Cloudinary* untuk penyimpanan *file* dan *ChromaDB* untuk penyimpanan vektor dan *metadata*, sistem menjadi lebih modular,*scalable*, dan terstruktur. Seluruh proses sebelumnya yang masih dijalankan secara manual melalui API dengan pengujian menggunakan Postman, kini telah dikembangkan ke dalam sistem *frontend* yang terintegrasi sepenuhnya, menjadikan platform ini lebih *user-friendly* dan siap digunakan oleh pengguna akhir secara langsung.

3.5.2 Implementasi Pengembangan

Sebagaimana telah diuraikan pada poin A dalam Subbab 3.5.1, terdapat beberapa kekurangan fitur sebelum pengembangan yang menjadi latar belakang dilaksanakannya kerja magang ini di perusahaan AI.DECE. Oleh karena itu, pengembangan solusi terhadap empat permasalahan utama yang diidentifikasi tersebut dijelaskan sebagai berikut.

A Tampilan Antarmuka Pengguna

1) Halaman dan Komponen yang Diperlukan

Pengembangan tampilan antarmuka pengguna (UI) memerlukan penyusunan beberapa halaman dan komponen utama yang dirancang untuk memudahkan navigasi dan fungsionalitas sistem. Penjelasan rinci mengenai empat menu inti yang harus dikembangkan adalah sebagai berikut.

- **Menu Home**

  Menu ini berfungsi sebagai *landing page* utama yang menampilkan beranda situs web. Pengguna dapat melihat ikhtisar fitur sistem, informasi terbaru, serta akses cepat ke berbagai layanan yang tersedia. Desainnya harus intuitif dan menyajikan tampilan visual yang menarik untuk memberikan kesan profesional sekaligus ramah pengguna.

- **Menu Documents Management**

  Menu ini memungkinkan pengguna untuk mengelola dokumen secara terstruktur. Fitur utamanya meliputi pembuatan *folder* atau kategori untuk mengelompokkan dokumen berdasarkan jenis, proyek, atau kebutuhan tertentu. Selain itu, pengguna dapat mengunggah (*upload*), mengedit, menghapus, atau melihat dokumen dengan antarmuka yang mudah dipahami. Menu inilah yang merupakan hasil pengembangan fitur *AI Grouping* pada Platform *AI Mastering Document*.

- **Menu AI Smart Chatbot**

  Menu ini menyediakan layanan *chatbot* cerdas berbasis kecerdasan buatan yang mampu merespons pertanyaan pengguna secara interaktif, mirip dengan percakapan antar manusia. Pengguna dapat memasukkan *prompt* atau pertanyaan, dan sistem akan memberikan jawaban yang relevan berdasarkan pemrosesan bahasa alami (*Natural Language Processing/NLP*). Antarmukanya harus dirancang agar terlihat modern dengan fitur riwayat percakapan dan opsi personalisasi respons.

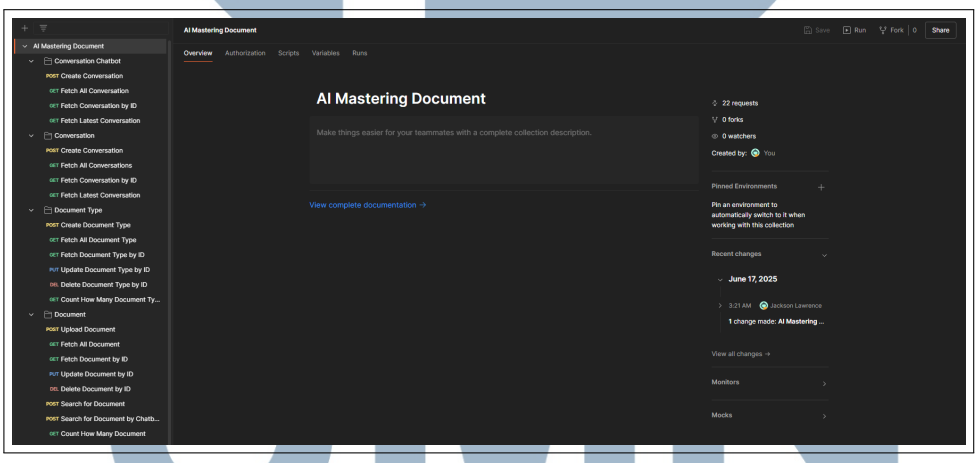
- **Menu AI Smart Searching**

  Berbeda dengan menu *AI Smart Chatbot* yang menghasilkan teks, menu ini difokuskan untuk pencarian dokumen berbasis AI. Ketika pengguna memasukkan pertanyaan, sistem akan menganalisis konten dokumen yang tersimpan dan menampilkan hasil pencarian yang paling sesuai dalam bentuk dokumen.

Selain keempat menu di atas, tampilan antarmuka juga harus mempertimbangkan aspek *user experience (UX)*, seperti kecepatan akses, konsistensi desain, dan responsivitas di berbagai perangkat. Detail lengkap mengenai *wireframe* dapat dilihat pada bagian 3).

2) API Endpoint Postman

Pada tahap awal pengembangan, seluruh interaksi dengan layanan *backend* diuji menggunakan *Postman*. Metode ini memudahkan verifikasi fungsi masing-masing API sebelum integrasi ke antarmuka *frontend*. Pada bagian ini dijelaskan daftar API *endpoints* yang telah didefinisikan di *Postman* untuk mendukung pengembangan *frontend* pada Platform *AI Mastering Document*. Setiap *endpoint* dirancang agar *frontend* dapat mengambil data secara langsung dari server, mulai dari *chatbot* dengan *memory management*, pencarian dokumen secara *semantic searching*, hingga pengelompokan otomatis dokumen melalui fitur *AI Grouping*. Selain itu, penjabaran dari API *endpoint* terdiri atas empat pembagian yaitu *Conversation Chatbots* untuk percakapan pada fitur menu *AI Smart Chatbot*, *Conversation* untuk percakapan pada fitur menu *AI Smart Searching*, *Document Type* untuk bagian kategori atau *folder*, dan *Document* untuk bagian dokumen PDF. Adapun tampilan yang masih berada pada API *endpoint* menggunakan Postman dapat dilihat pada Gambar 3.25.



Gambar 3.25. Tampilan API *endpoint* menggunakan Postman

Perincian API *endpoint* yang dapat diakses oleh *frontend* beserta dengan penjelasannya akan diutarakan sebagai berikut.

1. **Conversation Chatbot**

   - `POST /conversation_chatbots`
     API *Endpoint* ini digunakan untuk membuat sesi percakapan baru pada fitur menu *AI Smart Chatbot*.
   - `GET /conversation_chatbots`
     API *Endpoint* ini mengembalikan daftar seluruh sesi percakapan yang

pernah dibuat pada fitur menu *AI Smart Chatbot*. Respon berisi *array object* berupa JSON `conversation` dengan informasi seperti `id`, `created_at`, dan ringkasan pesan terakhir.

- `GET /conversation_chatbots/<<id>>`
  API *Endpoint* ini mengambil detail lengkap suatu sesi percakapan berdasarkan `id` unik pada fitur menu *AI Smart Chatbot*. Berguna untuk menampilkan riwayat chat tergantung dengan `id` yang dipilih.
- `GET /conversation_chatbots/latest`
  API *Endpoint* ini memanggil sesi percakapan paling baru yang dibuat pada fitur menu *AI Smart Chatbot*. Ideal untuk antarmuka *chat widget* yang ingin melanjutkan interaksi terakhir tanpa harus memilih `id` manual.

2. **Conversation**

- `POST /conversations`
  API *Endpoint* ini digunakan untuk membuat sesi percakapan baru pada fitur menu *AI Smart Searching*.
- `GET /conversations`
  API *Endpoint* ini mengembalikan daftar seluruh sesi percakapan yang pernah dibuat pada fitur menu *AI Smart Searching*. Respon berisi *array object* berupa JSON `conversation` dengan informasi seperti `id`, `created_at`, dan ringkasan pesan terakhir.
- `GET /conversations/<<id>>`
  API *Endpoint* ini mengambil detail lengkap suatu sesi percakapan berdasarkan `id` unik pada fitur menu *AI Smart Searching*. Data yang dikembalikan meliputi semua pesan, waktu, dan lainnya tergantung dengan `id` yang dipilih.
- `GET /conversations/latest`
  API *Endpoint* ini memanggil sesi percakapan paling baru yang dibuat pada fitur menu *AI Smart Searching*. Cocok untuk menampilkan diskusi terkini di *dashboard* atau ringkasan aktivitas pengguna.

3. **Document Type**

- `POST /document_types`
  API *Endpoint* ini berfungsi menambahkan kategori tipe dokumen baru

pada menu *Document Types Management*. Permintaan harus memuat nama kategori agar dapat dikenali pada proses unggah dokumen.

- `GET /document_types`
  API *Endpoint* ini berfungsi untuk mengembalikan daftar semua tipe dokumen yang telah didefinisikan pada menu *Document Types Management*.
- `GET /document_types/<<id>>`
  API *Endpoint* ini berfungsi untuk mengambil detail satu tipe dokumen berdasarkan `id` spesifik pada menu *Document Types Management*. Digunakan untuk menampilkan data lengkap sebelum proses *update* atau *delete*.
- `PUT /document_types/<<id>>`
  API *Endpoint* ini bertujuan untuk memperbarui nama tipe dokumen yang sudah ada berdasarkan `id` pada menu *Document Types Management*.
- `DELETE /document_types/<<id>>`
  API *Endpoint* ini bertujuan untuk menghapus tipe dokumen tertentu dari sistem berdasarkan `id` pada menu *Document Types Management*. Setelah operasi berhasil, dokumen dengan tipe tersebut tidak dapat diunggah lagi hingga dibuat ulang.
- `GET /document_types/count`
  API *Endpoint* ini berfungsi untuk menghitung total jumlah tipe dokumen yang tersedia dan ditampilkan pada menu *Home*. Memudahkan pengguna dengan menampilkan diskusi terkini di *dashboard* atau ringkasan aktivitas pengguna.

4. **Document**

- `POST /documents`
  API *Endpoint* ini digunakan untuk mengunggah berkas dokumen ke server pada menu *Documents Management*.
- `GET /documents`
  API *Endpoint* ini bertujuan untuk menampilkan daftar seluruh dokumen yang tersimpan pada menu *Documents Management*.

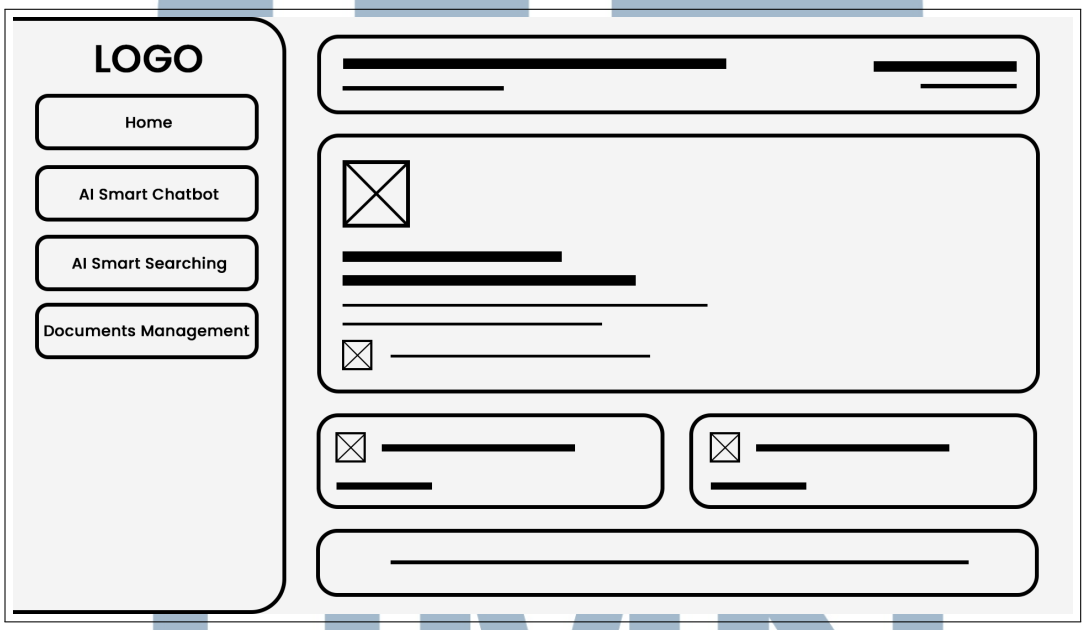
- `GET /documents/<<id>>`
  API *Endpoint* ini bertujuan untuk mengambil dokumen spesifik berdasarkan `id` pada menu *Documents Management*.
- `PUT /documents/<<id>>`
  API *Endpoint* ini bertujuan untuk memperbarui dokumen yang sudah diunggah berdasarkan `id`, baik isi file maupun *metadata* pada menu *Documents Management*.
- `DELETE /documents/<<id>>`
  API *Endpoint* ini berfungsi sebagai penghapusan dokumen beserta seluruh *metadata*-nya dari *database* berdasarkan `id` pada menu *Documents Management*. Operasi ini bersifat permanen dan tidak dapat dikembalikan.
- `POST /documents/search`
  API *Endpoint* ini melakukan pencarian dokumen berdasarkan kata kunci atau *metadata* saat memberikan *prompt* pada menu *AI Smart Searching*. Hasil pencarian berisi *file* dokumen yang relevan sebagai jawabannya.
- `POST /documents/smart-chatbot/search`
  API *Endpoint* ini melakukan pencarian dokumen berdasarkan kata kunci atau *metadata* saat memberikan *prompt* pada menu *AI Smart Chatbot*. Hasil pencarian berisi percakapan seperti percakapan antar manusia dengan jawaban yang relevan.
- `GET /documents/count`
  API *Endpoint* ini bertujuan untuk menghitung jumlah total dokumen yang telah diunggah dan ditampilkan pada menu *Home*. Berguna untuk memantau pertumbuhan koleksi dokumen dan merencanakan kapasitas penyimpanan.

3) Wireframe dan Tampilan User Interface

Pada bagian ini akan dibahas proses transisi dari pengujian API melalui *Postman* ke implementasi antarmuka pengguna (*UI*) pada *frontend*. Pembahasan dimulai dengan konsep dasar dan tujuan perancangan antarmuka, dilanjutkan dengan tahapan pembuatan *wireframe* sebagai kerangka visual, dan diakhiri dengan pemilihan elemen-elemen *UI* serta gaya visual yang mendukung pengalaman pengguna. Dengan struktur ini, diharapkan pembaca memahami alur berpikir dan

keputusan desain yang diambil dalam menghadirkan fitur *AI Smart Chatbot*, *AI Smart Searching*, dan *AI Grouping* secara interaktif dan intuitif.

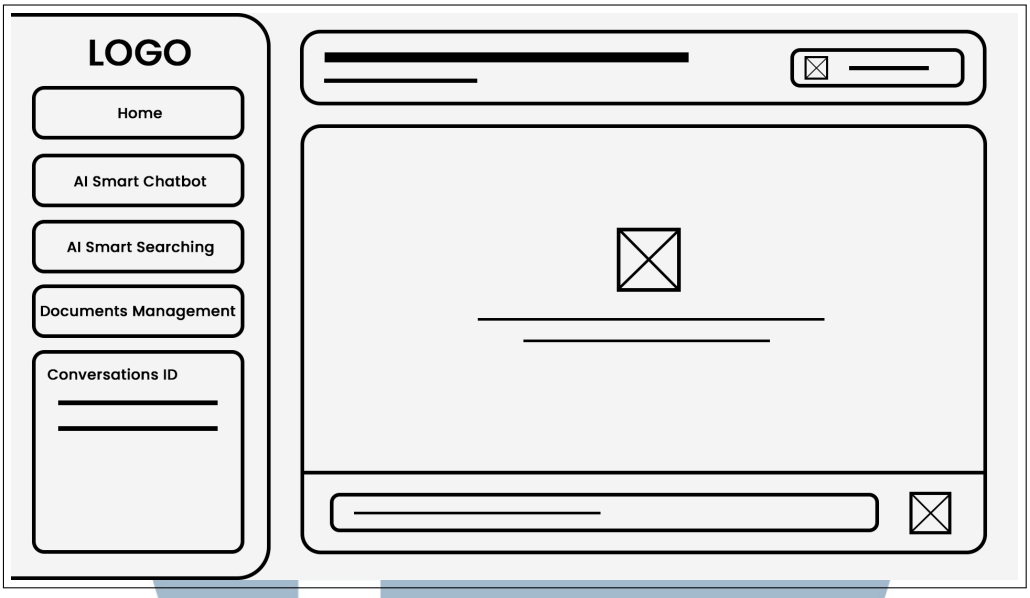
Proses pembuatan *wireframe* dimulai dari merancang sketsa kasar (*low-fidelity*) yang memetakan susunan komponen utama seperti panel obrolan dan area pengelompokan dokumen. Setiap elemen diletakkan berdasarkan prioritas fungsional dan alur kerja pengguna. Perancangan *wireframe* ini menggunakan *tools* seperti Figma dan Adobe Photoshop untuk membuat *wireframe* ini, memastikan konsistensi tata letak sebelum beralih ke tahap pengembangan sebenarnya. Semua visualisasi *wireframe* yang dibuat akan diutarakan pada gambar-gambar berikut.



Gambar 3.26. *Wireframe* Menu *Home*

Gambar 3.26 menunjukkan sebuah prototipe tampilan menu *Home* dengan tambahan informasi berupa total dokumen dan total kategori (folder) yang dibuat. Di bagian samping kiri terdapat *sidebar* yang berguna untuk pindah halaman sesuai dengan tujuan yang diinginkan antara empat menu yang tersedia.

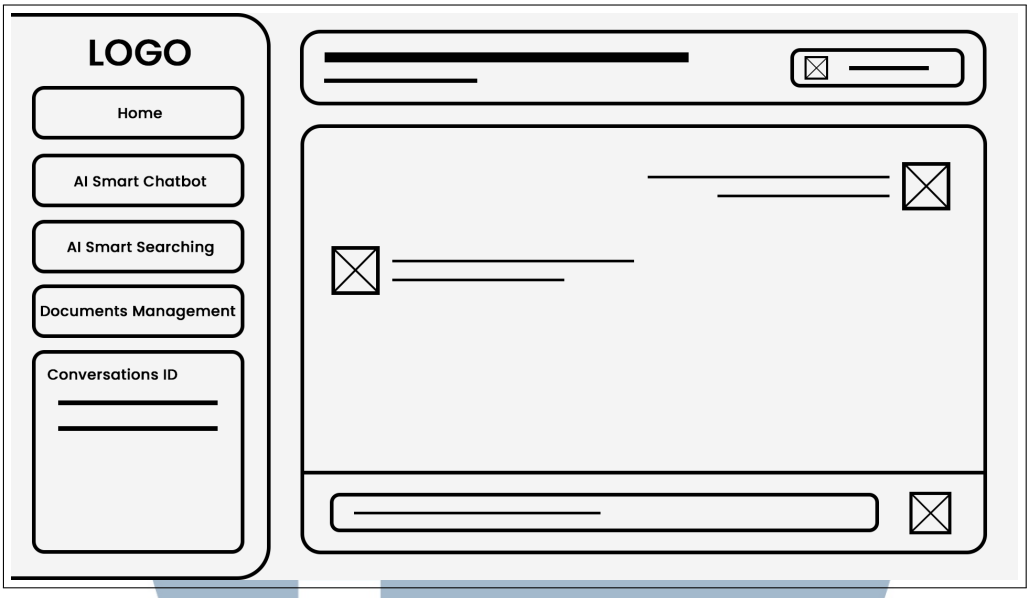
UNIVERSITAS MULTIMEDIA NUSANTARA



Gambar 3.27. *Wireframe* Menu *AI Smart Chatbot* dan *AI Smart Searching* sebelum melakukan percakapan

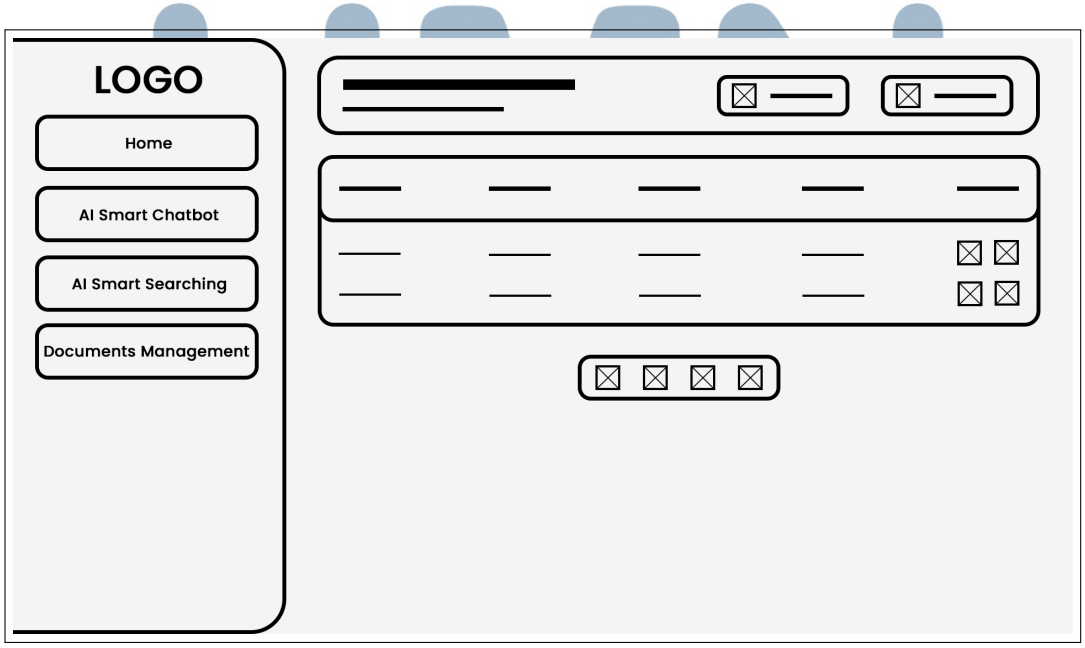
Gambar 3.27 menunjukkan sebuah prototipe tampilan menu *AI Smart Chatbot* dan *AI Smart Searching* saat belum melakukan percakapan dengan komponen sudut kanan atas merupakan tombol untuk menambahkan *id* percakapan atau *New Chat*. Di bagian samping kiri terdapat *sidebar* yang berguna untuk pindah halaman sesuai dengan tujuan yang diinginkan antara empat menu yang tersedia.

UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA



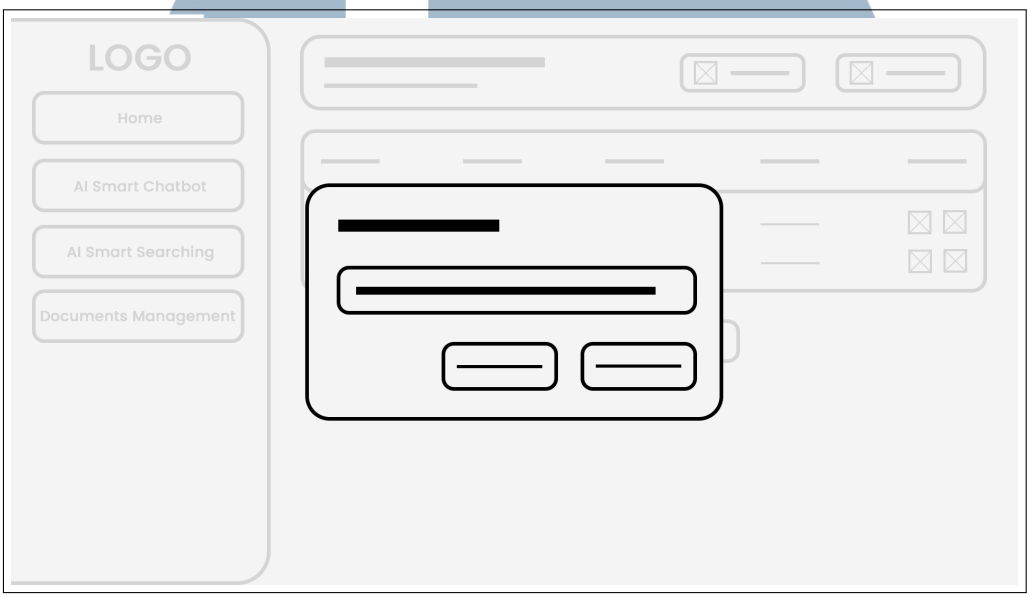
Gambar 3.28. *Wireframe* Menu *AI Smart Chatbot* dan *AI Smart Searching* setelah melakukan percakapan

Gambar 3.28 menunjukkan sebuah prototipe tampilan menu *AI Smart Chatbot* dan *AI Smart Searching* setelah melakukan percakapan dengan komponen sudut kanan atas merupakan tombol untuk menambahkan *id* percakapan atau *New Chat*. Di bagian samping kiri terdapat *sidebar* yang berguna untuk pindah halaman sesuai dengan tujuan yang diinginkan antara empat menu yang tersedia.



Gambar 3.29. *Wireframe* Menu *Documents Management*

Gambar 3.29 menunjukkan sebuah prototipe tampilan menu *Documents Management* dengan komponen sudut atas merupakan tombol untuk memindahkan halaman ke bagian membuat kategori atau *folder* yang dinamakan menu *Document Types Management* dan tombol untuk melakukan pengunggahan dokumen. Di bagian samping kiri terdapat *sidebar* yang berguna untuk pindah halaman sesuai dengan tujuan yang diinginkan antara empat menu yang tersedia.



Gambar 3.30. *Wireframe* Menu *Documents Management* saat mengunggah dokumen

Gambar 3.30 menunjukkan sebuah prototipe tampilan menu *Documents Management* dengan komponen *Pop Up* berupa isi tampilan untuk melakukan pengunggahan dokumen.

UNIVERSITAS MULTIMEDIA NUSANTARA

Gambar 3.31. *Wireframe* Menu *Documents Management* saat menghapus dokumen

Gambar 3.31 menunjukkan sebuah prototipe tampilan menu *Documents Management* dengan komponen *Pop Up* berupa isi tampilan untuk melakukan konfirmasi penghapusan dokumen.

Gambar 3.32. *Wireframe* Menu *Document Types Management*

Gambar 3.32 menunjukkan sebuah prototipe tampilan menu *Document Types Management* dengan komponen sudut atas merupakan tombol untuk memindahkan halaman ke bagian mengunggah dokumen dan tombol untuk

melakukan penambahan kategori atau *folder*. Di bagian samping kiri terdapat *sidebar* yang berguna untuk pindah halaman sesuai dengan tujuan yang diinginkan antara empat menu yang tersedia.

Gambar 3.33. *Wireframe* Menu *Document Types Management* saat menambah atau mengganti nama kategori atau folder

Gambar 3.33 menunjukkan sebuah prototipe tampilan menu *Document Types Management* dengan komponen *Pop Up* berupa isi tampilan untuk melakukan penambahan atau pengubahan nama kategori atau *folder* yang dibuat.

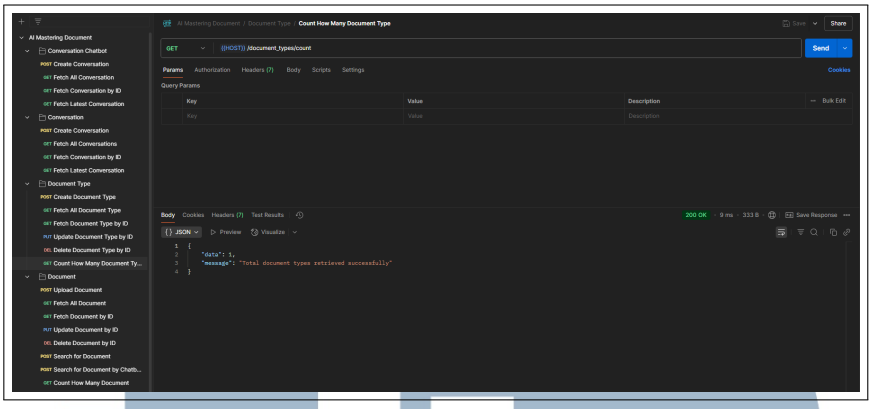
UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA

Gambar 3.34. *Wireframe* Menu *Document Types Management* saat nama kategori atau folder

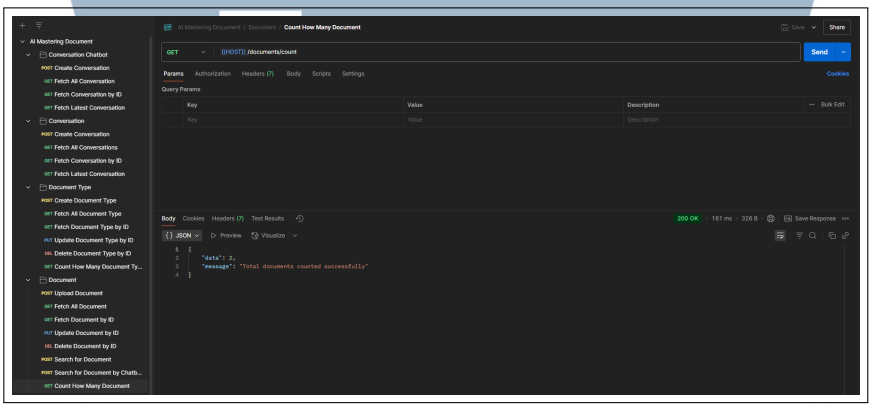
Gambar 3.34 menunjukkan sebuah prototipe tampilan menu *Document Types Management* dengan komponen *Pop Up* berupa isi tampilan untuk melakukan konfirmasi penghapusan nama kategori atau *folder* yang dibuat.

Pada tahap tampilan *UI* (*high-fidelity*), *wireframe* dihidupkan dengan penambahan palet warna, tipografi, ikonografi, dan interaksi mikro (*micro-interactions*). Antarmuka menu *AI Smart Chatbot* diperkaya dengan balon pesan berwarna lembut dan animasi pengetikan untuk memberikan rasa responsif, sedangkan menu *AI Smart Searching* menyajikan hasil pencarian dalam bentuk percakapan juga yang memuat cuplikan konteks dokumen. Untuk *AI Grouping*, *UI* menampilkan *folder* otomatis dalam ukuran responsif yang berada pada menu *Documents Management*. Seluruh desain mengacu pada prinsip *usability* dan *accessibility* agar Platform *AI Mastering Document* mudah digunakan oleh berbagai kalangan. Untuk tampilan sebelum dan sesudah pengembangan dapat dilihat pada beberapa gambar berikut.

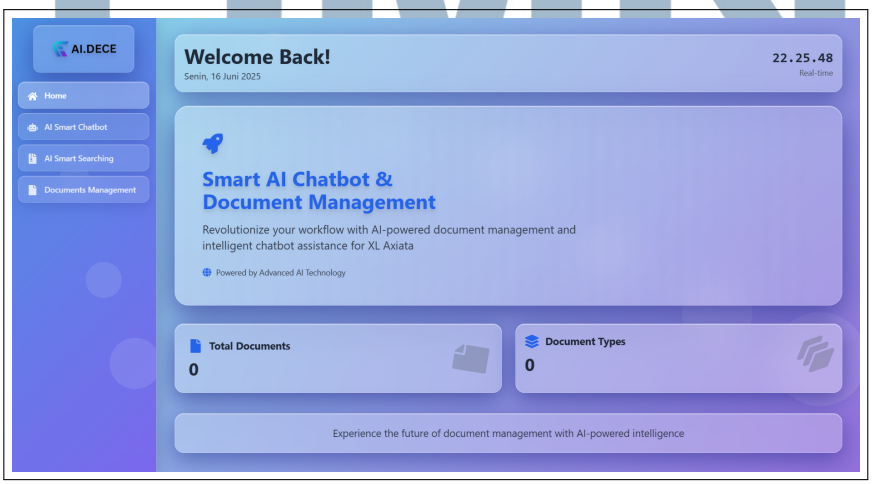
UNIVERSITAS MULTIMEDIA NUSANTARA



Gambar 3.35. Tampilan sebelum pengembangan pada menu *Home* pada pengambilan banyak kategori atau *folder*

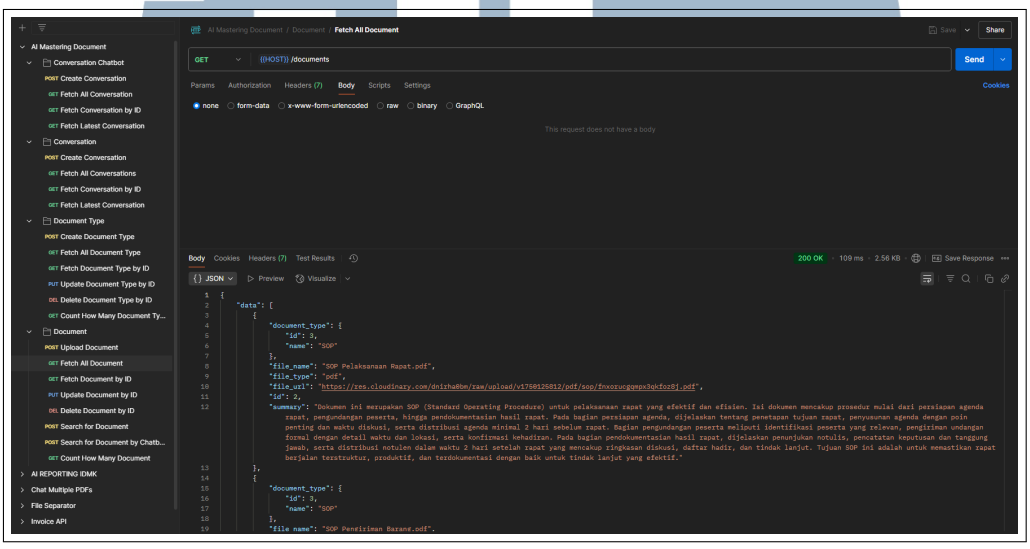


Gambar 3.36. Tampilan sebelum pengembangan pada menu *Home* pada pengambilan banyak dokumen yang telah diunggah

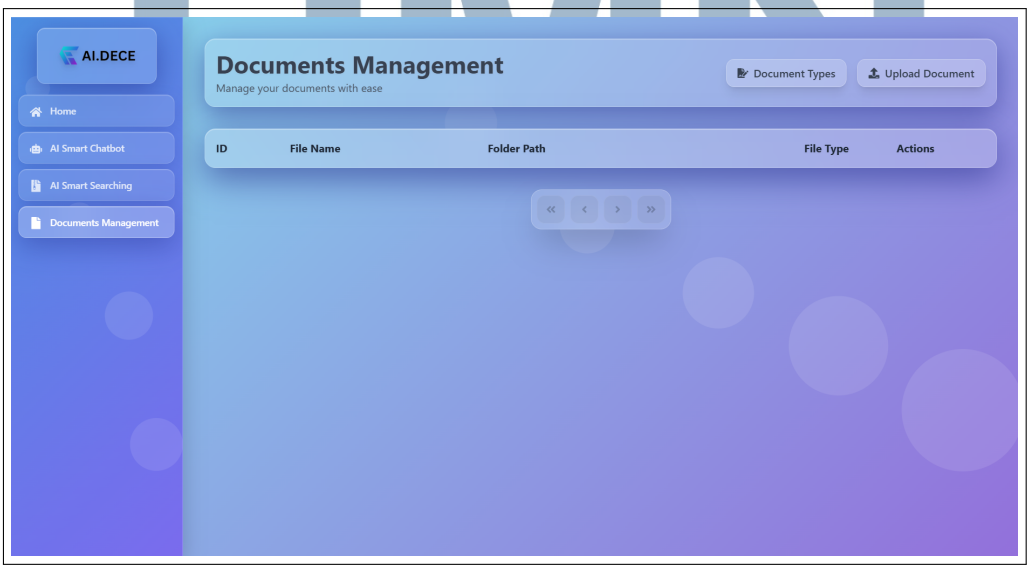


Gambar 3.37. Tampilan sesudah pengembangan pada menu *Home*

Gambar 3.35 dan 3.36 menunjukkan tampilan sebelum adanya pengembangan pada menu *Home* yang hanya dapat dilihat pada Postman sebagai *tools* untuk pengujian API. API *endpoint* yang digunakan untuk mengambil data tersebut dan ditampilkan ke *frontend* adalah `GET document_types/count` untuk melihat total kategori atau *folder* yang telah dibuat dan `GET documents/count` untuk melihat total dokumen yang telah diunggah. Selain itu, Gambar 3.37 menunjukkan tampilan yang telah diintegrasikan dengan mekanisme *frontend* menggunakan API *endpoint* yang telah disebutkan sebelumnya.

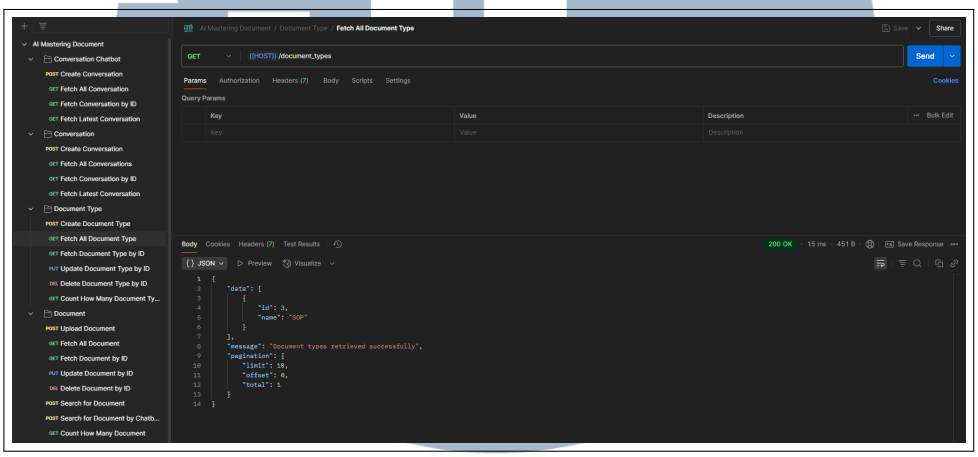


Gambar 3.38. Tampilan sebelum pengembangan pada menu *Documents Management*

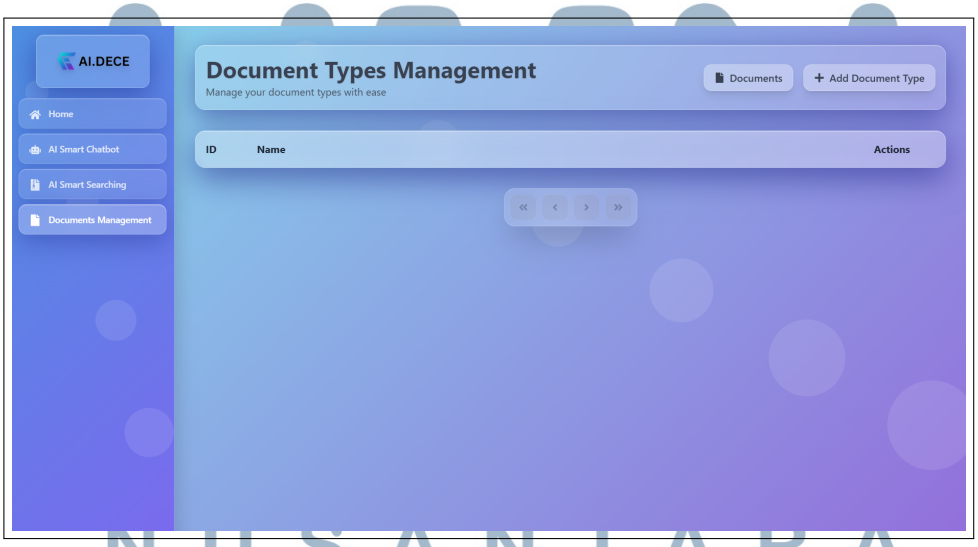


Gambar 3.39. Tampilan sesudah pengembangan pada menu *Documents Management*

Gambar 3.38 menunjukkan tampilan sebelum adanya pengembangan pada menu *Document Types Management* yang hanya dapat dilihat pada Postman sebagai *tools* untuk pengujian API. API *endpoint* yang digunakan untuk mengambil data tersebut dan ditampilkan ke *frontend* adalah `GET documents` untuk melihat semua kategori atau *folder* yang telah dibuat dan `GET documents/<<id>>` untuk melihat kategori atau *folder* berdasarkan `id`. Selain itu, Gambar 3.39 menunjukkan tampilan yang telah diintegrasikan dengan mekanisme *frontend* menggunakan API *endpoint* `GET document_types`.



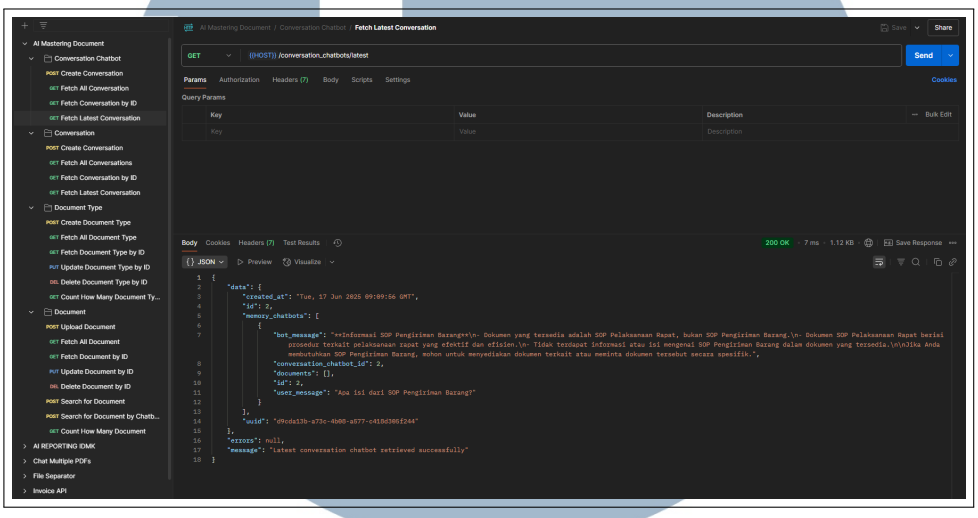
Gambar 3.40. Tampilan sebelum pengembangan pada menu *Document Types Management*



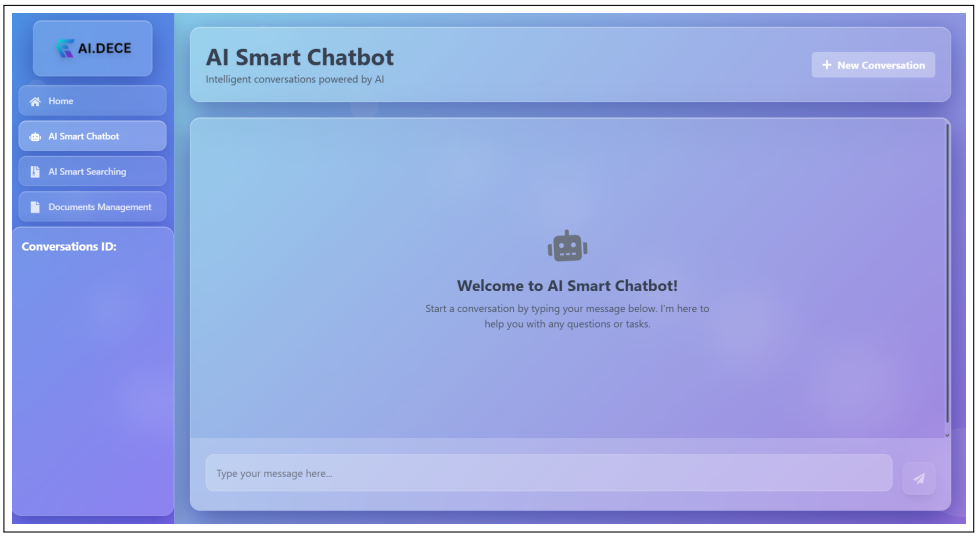
Gambar 3.41. Tampilan sesudah pengembangan pada *Document Types Management*

Gambar 3.40 menunjukkan tampilan sebelum adanya pengembangan pada menu *Document Types Management* yang hanya dapat dilihat pada Postman sebagai

tools untuk pengujian API. API *endpoint* yang digunakan untuk mengambil data tersebut dan ditampilkan ke *frontend* adalah `GET document_types` untuk melihat semua kategori atau *folder* yang telah dibuat dan `GET document_types/<<id>>` untuk melihat kategori atau *folder* berdasarkan `id`. Selain itu, Gambar 3.41 menunjukkan tampilan yang telah diintegrasikan dengan mekanisme *frontend* menggunakan API *endpoint* `GET document_types`.



Gambar 3.42. Tampilan sebelum pengembangan pada menu *AI Smart Chatbot*

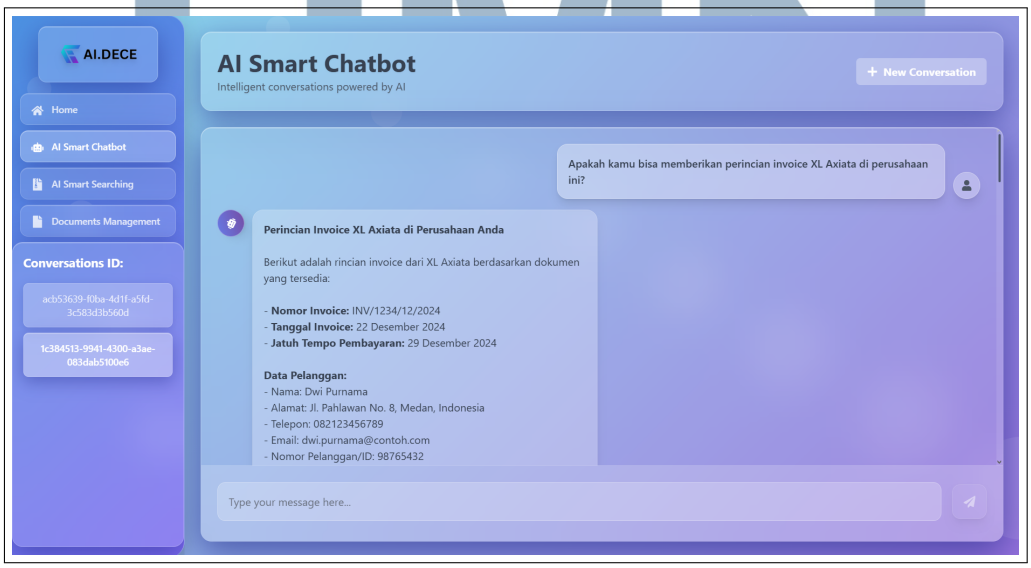


Gambar 3.43. Tampilan sesudah pengembangan pada menu *AI Smart Chatbot*

Gambar 3.42 menunjukkan tampilan sebelum adanya pengembangan pada menu *AI Smart Chatbot* yang hanya dapat dilihat pada Postman sebagai *tools* untuk pengujian API. API *endpoint* yang digunakan untuk mengambil data

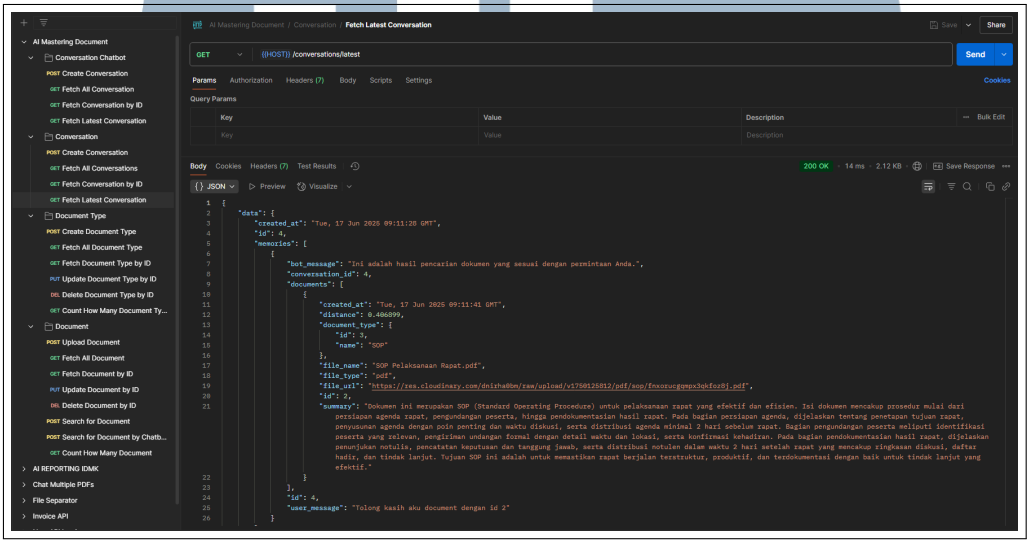
tersebut dan ditampilkan ke *frontend* adalah `GET conversation_chatbots` untuk melihat percakapan pada menu *AI Smart Chatbot*, dan `GET conversation_chatbots/<<id>>` untuk melihat percakapan pada menu *AI Smart Chatbot* berdasarkan `id`, serta `GET conversation_chatbots/latest` untuk melihat percakapan pada menu *AI Smart Chatbot* yang terbaru. Selain itu, Gambar 3.43 menunjukkan tampilan yang telah diintegrasikan dengan mekanisme *frontend* menggunakan API *endpoint* `GET conversation_chatbots`.

Gambar 3.44. Tampilan sebelum pengembangan pada menu *AI Smart Chatbot* saat memberikan respons

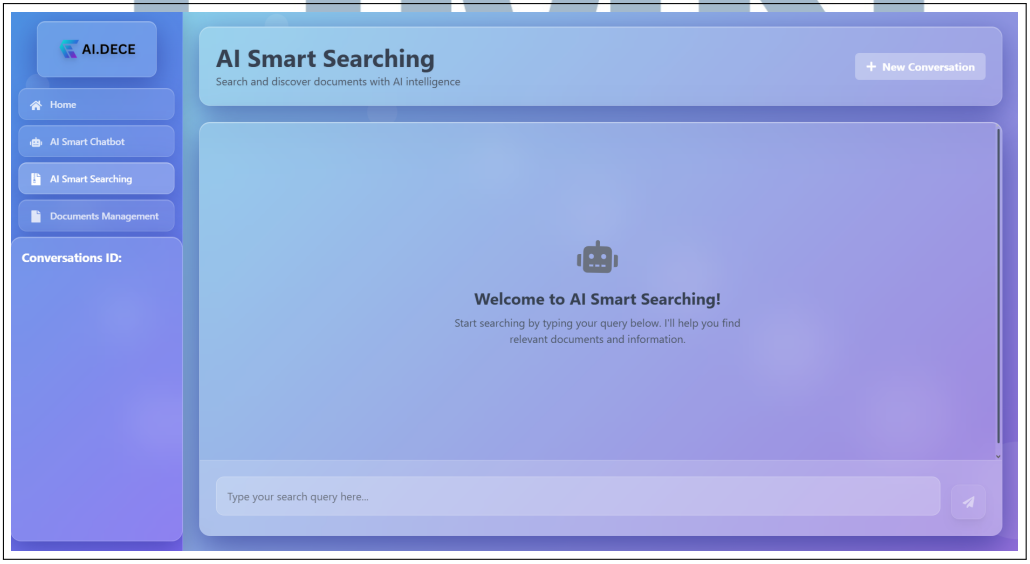


Gambar 3.45. Tampilan sesudah pengembangan pada menu *AI Smart Chatbot* saat memberikan respons

Gambar 3.44 menunjukkan tampilan sebelum adanya pengembangan pada menu *AI Smart Chatbot* yang dapat dilihat bahwa jawaban yang dihasilkan adalah tampak pada terminal yang dinilai sangat susah untuk dilihat. Selain itu, Gambar 3.45 menunjukkan tampilan yang telah diintegrasikan dengan mekanisme *frontend* menggunakan API *endpoint* `GET conversation_chatbots/<<id>>` dan saat menekan tombol kirim untuk mendapatkan jawaban atas *prompt* yang sudah diberikan, API *endpoint* yang digunakan adalah `POST documentssmart-chatbot/search`.

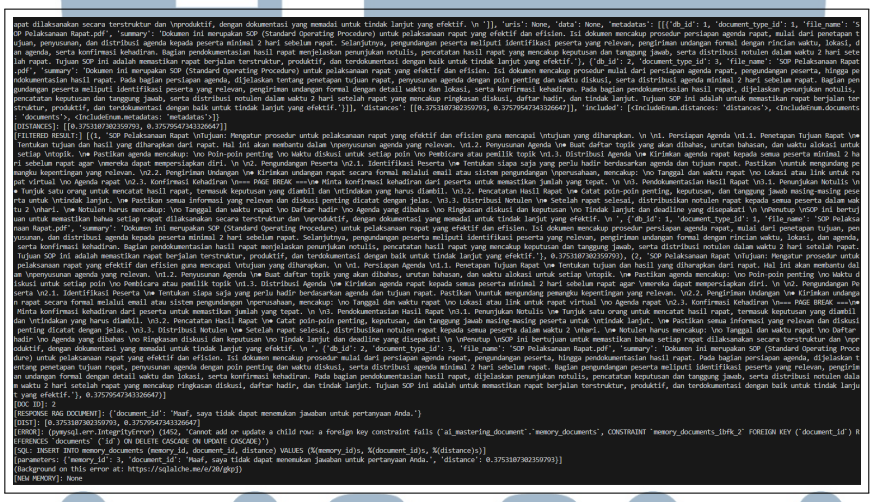


Gambar 3.46. Tampilan sebelum pengembangan pada menu *AI Smart Searching*

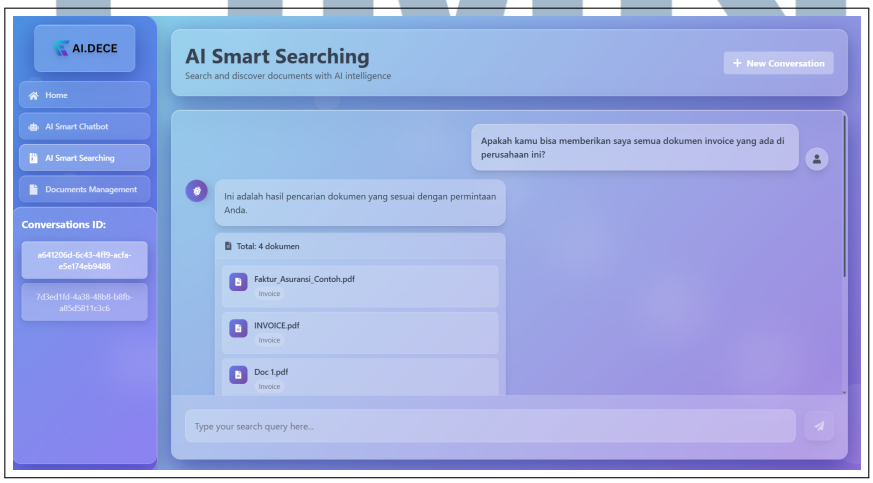


Gambar 3.47. Tampilan sesudah pengembangan pada menu *AI Smart Searching*

Gambar 3.46 menunjukkan tampilan sebelum adanya pengembangan pada menu *AI Smart Searching* yang hanya dapat dilihat pada Postman sebagai *tools* untuk pengujian API. API *endpoint* yang digunakan untuk mengambil data tersebut dan ditampilkan ke *frontend* adalah `GET conversations` untuk melihat percakapan pada menu *AI Smart Searching*, dan `GET conversations/<<id>>` untuk melihat percakapan pada menu *AI Smart Searching* berdasarkan `id`, serta `GET conversations/latest` untuk melihat percakapan pada menu *AI Smart Searching* yang terbaru. Selain itu, Gambar 3.47 menunjukkan tampilan yang telah diintegrasikan dengan mekanisme *frontend* menggunakan API *endpoint* `GET conversations`.



Gambar 3.48. Tampilan sebelum pengembangan pada menu *AI Smart Searching* saat memberikan respons



Gambar 3.49. Tampilan sesudah pengembangan pada menu *AI Smart Searching* saat memberikan respons

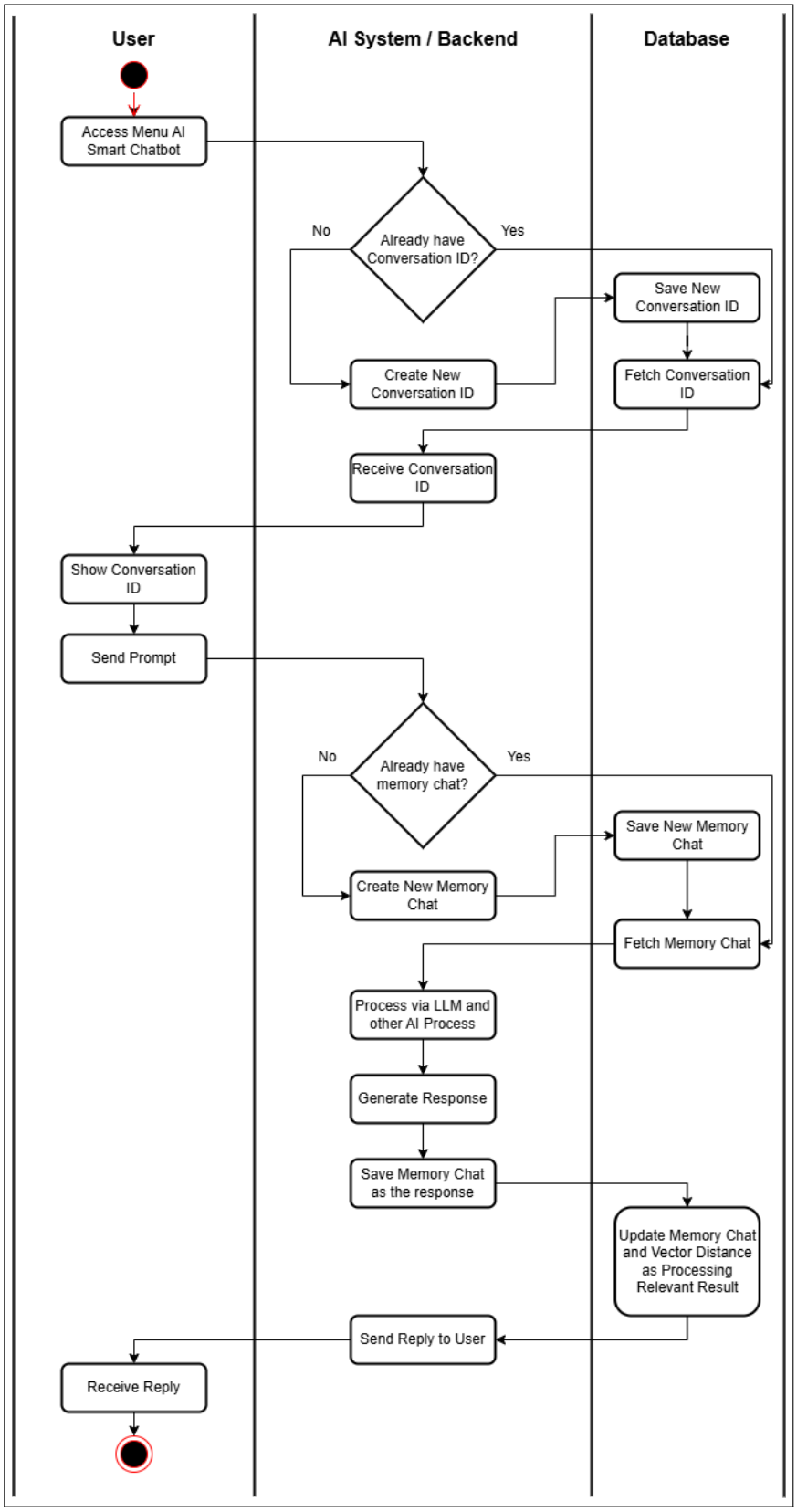
Gambar 3.48 menunjukkan tampilan sebelum adanya pengembangan pada menu *AI Smart Searching* yang dapat dilihat bahwa jawaban yang dihasilkan adalah tampak pada terminal yang dinilai sangat susah untuk dilihat. Selain itu, Gambar 3.49 menunjukkan tampilan yang telah diintegrasikan dengan mekanisme *frontend* menggunakan API *endpoint* `GET conversations/<<id>>` dan saat menekan tombol kirim untuk mendapatkan jawaban atas *prompt* yang sudah diberikan, API *endpoint* yang digunakan adalah `POST documentssearch`.

B Fitur AI Smart Chatbot

1) Activity Diagram AI Smart Chatbot

Activity Diagram merupakan salah satu *Unified Modeling Language* (UML) yang berfungsi untuk memodelkan proses alur aktivitas di dalam suatu sistem, dengan memperlihatkan rangkaian langkah yang dilakukan oleh aktor untuk mencapai suatu tujuan tertentu. Diagram ini merepresentasikan hubungan antar aktivitas, proses pengambilan keputusan, serta kondisi paralel yang dapat memengaruhi jalannya alur proses [84]. Oleh karena itu, *Activity Diagram* memberikan gambaran yang jelas mengenai pengelolaan alur kerja, mekanisme pengambilan keputusan, serta hasil dari interaksi yang berlangsung dalam sistem [85].

UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA



Gambar 3.50. *Activity Diagram* pada fitur *AI Smart Chatbot*

Gambar 3.50 menggambarkan alur interaksi antara pengguna, sistem AI, dan basis data dalam menjalankan sesi percakapan cerdas. Proses dimulai ketika pengguna memilih menu *AI Smart Chatbot* pada antarmuka aplikasi. Sistem AI kemudian memeriksa apakah terdapat *Conversation ID* yang aktif. Apabila belum ada, sistem akan membuat *Conversation ID* baru dan menyimpannya ke dalam basis data, sedangkan jika sudah ada, sistem menarik kembali *Conversation ID* yang tersimpan. Setelah *Conversation ID* tersedia, sistem mengambil riwayat memori percakapan dari basis data untuk memastikan konteks obrolan tetap terjaga.

Selanjutnya, pengguna mengirimkan prompt atau pertanyaan ke dalam sesi tersebut, yang kemudian diproses oleh mesin *Large Language Model* (LLM) dan teknologi AI lainnya. Hasil pemrosesan LLM berupa respons kemudian dihasilkan oleh sistem. Sebelum mengirimkan respons kepada pengguna, sistem memperbarui memori percakapan dengan mencatat input pengguna dan respons AI, serta menghitung dan menyimpan jarak vektor kontekstual dari pencarian semantik pada *vector database* menggunakan ChromaDB. Akhirnya, respons yang dihasilkan dikirimkan kembali ke pengguna, menutup satu iterasi alur aktivitas dan menyiapkan sistem untuk pertanyaan lanjutan berikutnya.

2) Pengujian dan Output

Pengujian terhadap pengembangan fitur *AI Smart Chatbot* pada Platform *AI Mastering Document* telah dilengkapi dengan kemampuan manajemen memori agar dapat menggunakan mekanisme *follow-up questions* dalam berbagai skenario penggunaan. Pada Tabel 3.14 menyajikan deskripsi pengujian, hasil yang diharapkan, hasil pengujian, dan kesimpulan secara lebih lengkap dan komprehensif.

Tabel 3.14. Hasil pengujian dan evaluasi fitur *AI Smart Chatbot*

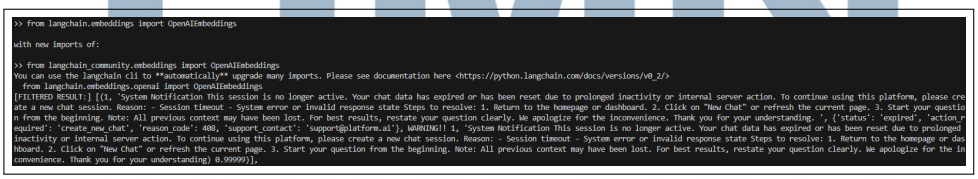
No.	Pengujian	Skenario	Hasil Pengujian	Kesimpulan
1	Memeriksa kemampuan *memory management* untuk mekanisme *follow-up questions*	Pengguna dapat melakukan *follow-up questions*	Pengguna dapat memberikan pertanyaan lanjutan	Sesuai dan Berhasil

Lanjut pada halaman berikutnya

Lanjutan Tabel 3.14

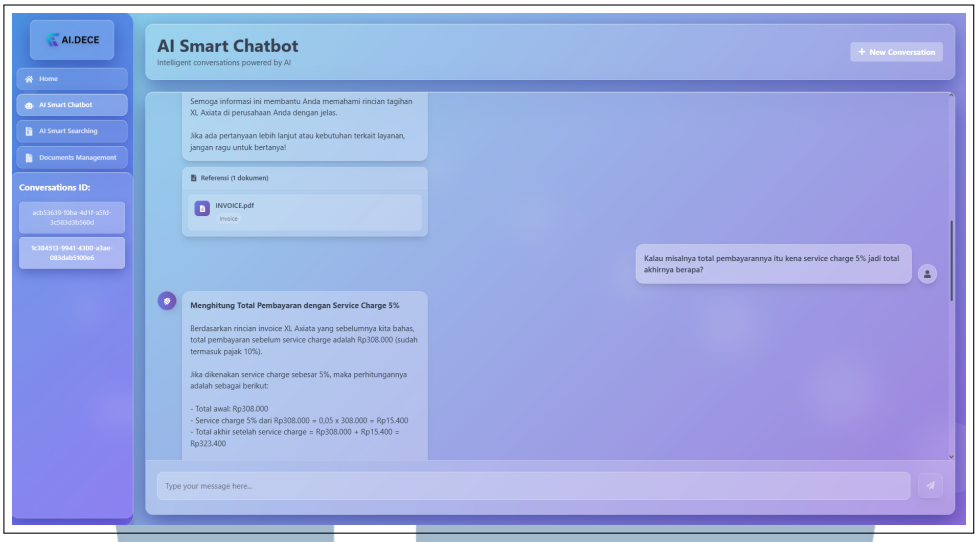
No.	Pengujian	Skenario	Hasil Pengujian	Kesimpulan
2	Memastikan tampilan menu *AI Smart Chatbot* sesuai	Pengguna dapat mengakses menu *AI Smart Chatbot* sesuai dengan desain	Tampilan menu *AI Smart Chatbot* sesuai dengan desain	Sesuai dan Berhasil
3	Memastikan adanya *error messages*	Pengguna dapat melihat informasi *error messages*	Tampilan dan informasi *error messages* muncul	Sesuai dan Berhasil
4	Memeriksa konsistensi tampilan yang responsif	Pengguna dapat mengakses menu *AI Smart Chatbot* dengan tampilan responsif	Tampilan menu *AI Smart Chatbot* responsif untuk semua ukuran *device*	Sesuai dan Berhasil

Adapun contoh *output* yang dapat dilihat adalah pada Gambar 3.52 yang berisikan tampilan dengan mekanisme *follow-up questions* dan telah dilengkapi dengan tampilan antarmuka pengguna yang sesuai. Selain itu, Gambar 3.51 merupakan tampilan dengan mekanisme yang belum mengizinkan pengguna untuk memberikan pertanyaan lanjutan.



Gambar 3.51. Tampilan sebelum pengembangan pada fitur *AI Smart Chatbot*

UNIVERSITAS MULTIMEDIA NUSANTARA



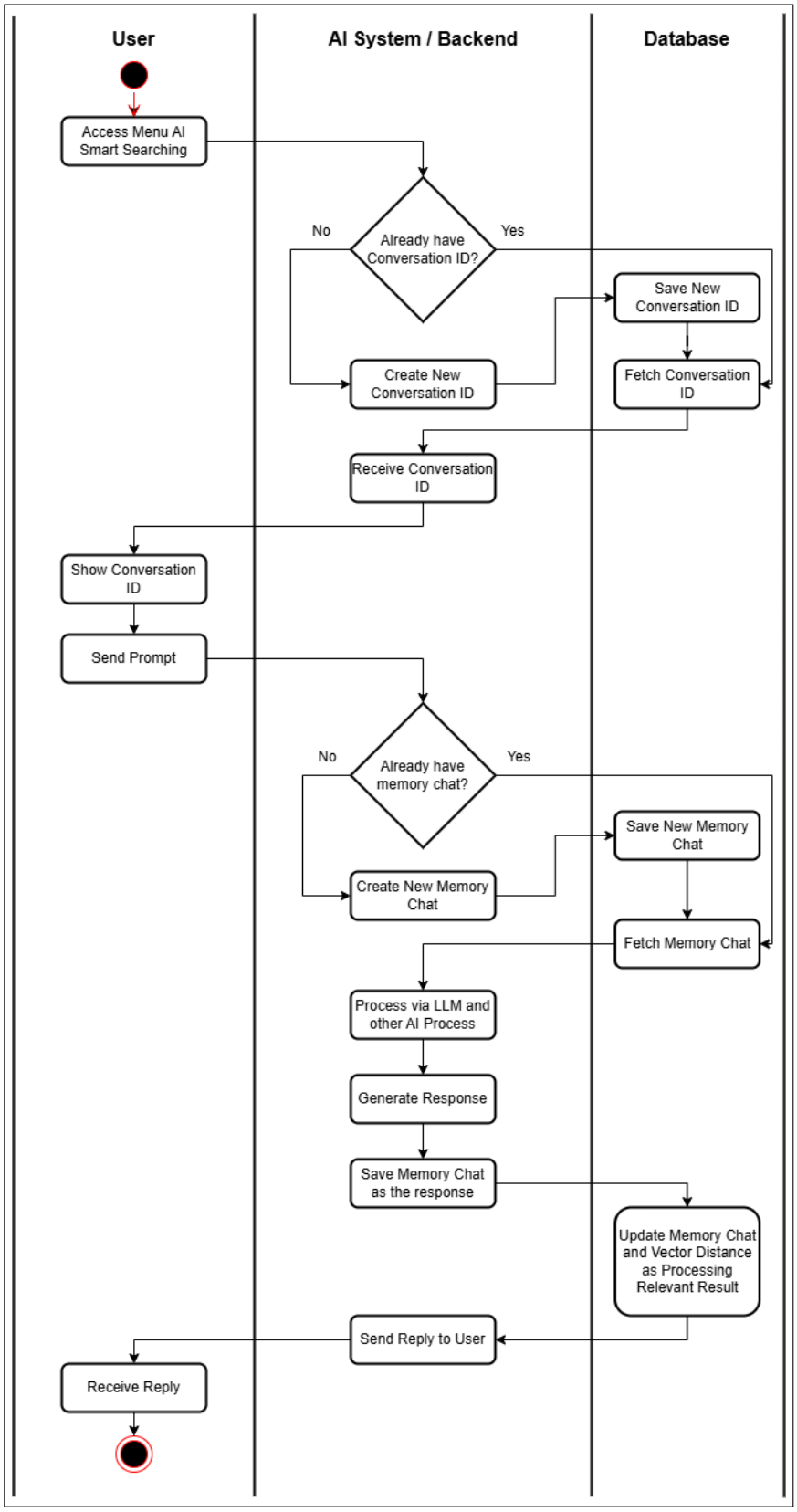
Gambar 3.52. Tampilan sesudah pengembangan pada fitur *AI Smart Chatbot*

C Fitur AI Smart Searching

1) Activity Diagram AI Smart Searching

Selain *Activity Diagram* yang tersedia pada fitur *AI Smart Chatbot*, tentu pada fitur *AI Smart Searching* juga memiliki *Activity Diagram*-nya sendiri yaitu sebagai berikut.

UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA



Gambar 3.53. *Activity Diagram* pada fitur *AI Smart Searching*

Gambar 3.53 menggambarkan alur interaksi antara pengguna, sistem AI, dan basis data dalam menjalankan sesi percakapan cerdas. Proses dimulai ketika pengguna memilih menu *AI Smart Searching* pada antarmuka aplikasi. Sistem AI kemudian memeriksa apakah terdapat *Conversation ID* yang aktif. Apabila belum ada, sistem akan membuat *Conversation ID* baru dan menyimpannya ke dalam basis data, sedangkan jika sudah ada, sistem menarik kembali *Conversation ID* yang tersimpan. Setelah *Conversation ID* tersedia, sistem mengambil riwayat memori percakapan dari basis data untuk memastikan konteks obrolan tetap terjaga.

Selanjutnya, pengguna mengirimkan prompt atau pertanyaan ke dalam sesi tersebut, yang kemudian diproses oleh mesin *Large Language Model* (LLM) dan teknologi AI lainnya. Hasil pemrosesan LLM berupa respons kemudian dihasilkan oleh sistem. Sebelum mengirimkan respons kepada pengguna, sistem memperbarui memori percakapan dengan mencatat input pengguna dan respons AI, serta menghitung dan menyimpan jarak vektor kontekstual dari pencarian semantik pada *vector database* menggunakan ChromaDB. Akhirnya, respons yang dihasilkan dikirimkan kembali ke pengguna, menutup satu iterasi alur aktivitas dan menyiapkan sistem untuk pertanyaan lanjutan berikutnya.

2) Pengujian dan Output

Pengujian terhadap pengembangan fitur *AI Smart Searching* pada Platform *AI Mastering Document* telah dilengkapi dengan kemampuan pencarian semantik dan tidak berdasarkan kata kunci atau *metadata*. Pada Tabel 3.15 menyajikan deskripsi pengujian, hasil yang diharapkan, hasil pengujian, dan kesimpulan secara lebih lengkap dan komprehensif.

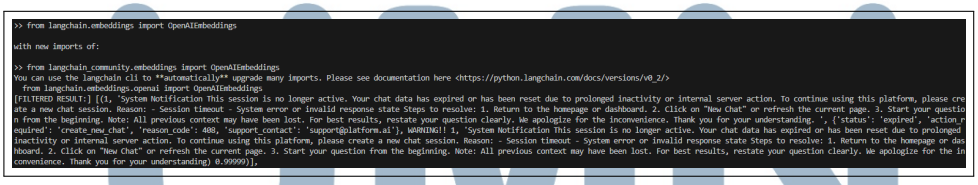
Tabel 3.15. Hasil pengujian dan evaluasi fitur *AI Smart Searching*

No.	Pengujian	Skenario	Hasil Pengujian	Kesimpulan
1	Memeriksa kemampuan pencarian semantik	Pengguna dapat melakukan pencarian semantik	Pencarian semantik sesuai dan relevan dengan pertanyaan pengguna	Sesuai dan Berhasil
Lanjut pada halaman berikutnya				

Lanjutan Tabel 3.15

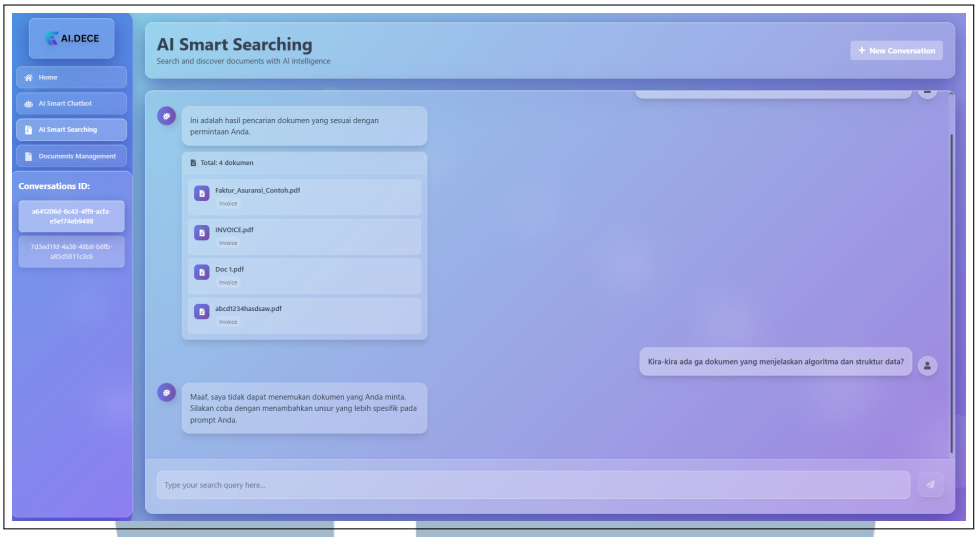
No.	Pengujian	Skenario	Hasil Pengujian	Kesimpulan
2	Memastikan tampilan menu *AI Smart Searching* sesuai	Pengguna dapat mengakses menu *AI Smart Searching* sesuai dengan desain	Tampilan menu *AI Smart Searching* sesuai dengan desain	Sesuai dan Berhasil
3	Memastikan adanya *error messages*	Pengguna dapat melihat informasi *error messages*	Tampilan dan informasi *error messages* muncul	Sesuai dan Berhasil
4	Memeriksa konsistensi tampilan yang responsif	Pengguna dapat mengakses menu *AI Smart Searching* dengan tampilan responsif	Tampilan menu *AI Smart Searching* responsif untuk semua ukuran *device*	Sesuai dan Berhasil

Adapun contoh *output* yang dapat dilihat adalah pada Gambar 3.55 yang berisikan tampilan dengan kemampuan pencarian berdasarkan pencarian semantik. Selain itu, Gambar 3.54 merupakan tampilan dengan mekanisme yang masih berdasarkan kata kunci atau *metadata*.



Gambar 3.54. Tampilan sebelum pengembangan pada fitur *AI Smart Searching*

UNIVERSITAS MULTIMEDIA NUSANTARA



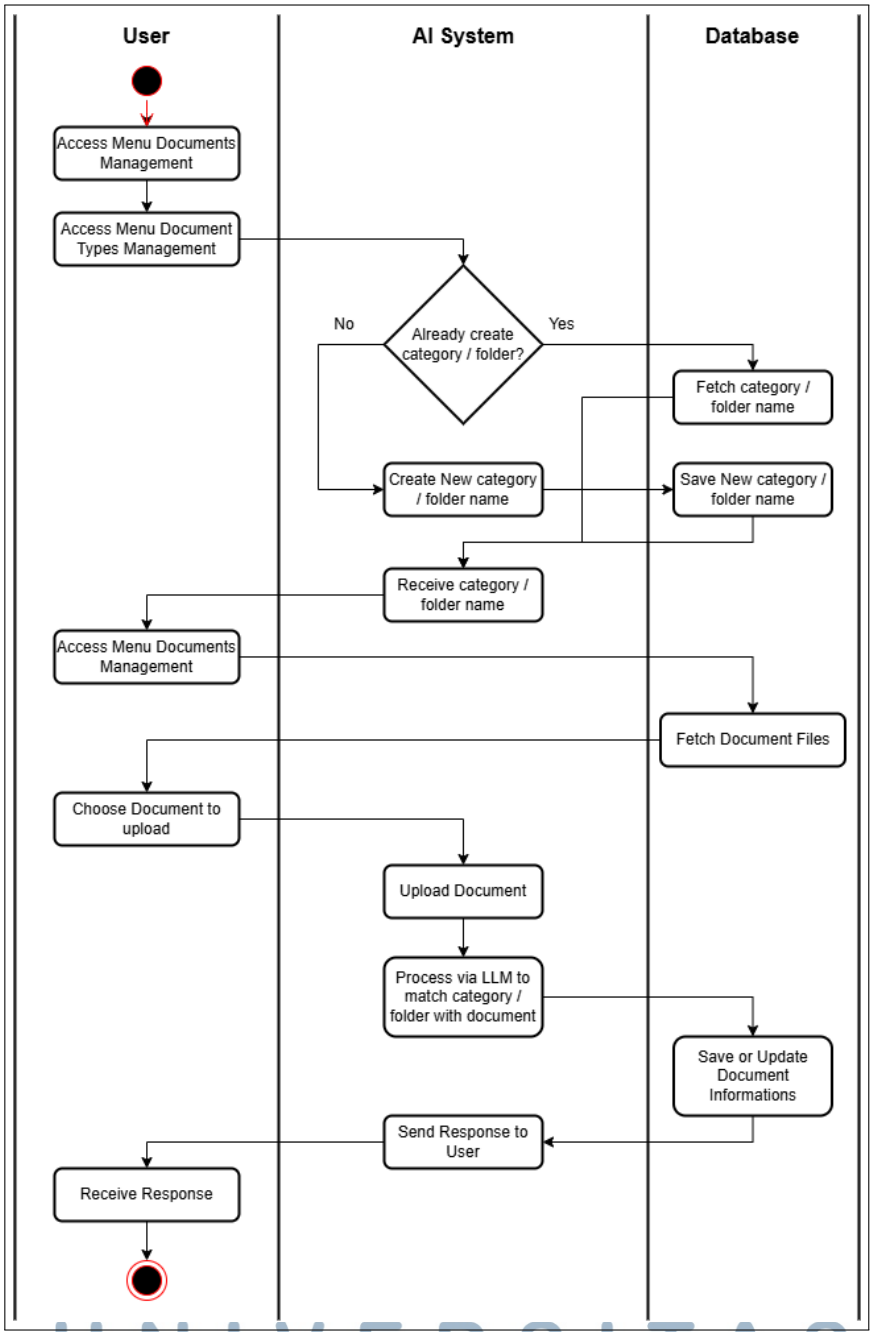
Gambar 3.55. Tampilan sesudah pengembangan pada fitur *AI Smart Searching*

D Fitur AI Grouping

1) Activity Diagram AI Grouping

Selain *Activity Diagram* yang tersedia pada fitur *AI Smart Chatbot* dan *AI Smart Searching*, tentu pada fitur *AI Grouping* juga memiliki *Activity Diagram*-nya sendiri yaitu sebagai berikut.

UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA



Gambar 3.56. *Activity Diagram* pada fitur *AI Grouping*

UNIVERSITAS MULTIMEDIA NUSANTARA

Diagram aktivitas fitur *AI Grouping* pada Platform *AI Mastering Document* ini dibagi menjadi tiga bagian utama, yaitu *User*, *AI System*, dan *Database*. Alur aktivitas dimulai pada bagian *User*, dimana pengguna pertama-tama memilih menu *Documents Management*, kemudian memasuki submenu *Document Types Management*. Setelah itu, input pengguna diteruskan ke sistem AI untuk memeriksa apakah kategori atau *folder* yang diinginkan sudah pernah dibuat sebelumnya.

Pada bagian *AI System*, terdapat sebuah keputusan (decision) dengan label "Already create category / folder?". Jika jawabannya "No", sistem akan menjalankan aktivitas *Create New category / folder name*, kemudian meneruskan nama baru tersebut ke bagian *database* untuk disimpan melalui aktivitas *Save New category / folder name*. Sebaliknya, jika jawabannya "Yes", sistem cukup mengambil nama kategori atau *folder* yang sudah ada melalui aktivitas *Fetch category / folder name* di *database*. Setelah nama kategori atau *folder* didapatkan (baik dari pembuatan baru maupun pengambilan lama), sistem AI mengirimkannya kembali ke pengguna.

Setelah pengguna menerima nama kategori atau *folder*, alur kembali ke bagian *User* untuk memilih menu *Documents Management* dan menentukan dokumen yang akan diunggah. Dokumen tersebut kemudian diunggah ke sistem AI (*Upload Document*), yang selanjutnya memproses dokumen menggunakan LLM untuk mencocokkan konten dokumen dengan kategori atau *folder* yang telah ditentukan (*Process via LLM to match category / folder with document*). Hasil pemrosesan ini kemudian disimpan atau diperbarui di database melalui aktivitas *Save or Update Document Informations*. Terakhir, sistem AI mengirimkan respons berisi hasil pengelompokan kembali ke pengguna, dan pengguna menutup alur dengan menerima respons tersebut.

2) Pengujian dan Output

Pengujian terhadap pengembangan fitur *AI Grouping* pada Platform *AI Mastering Document* telah dilengkapi dengan kemampuan mengkategorikan isi dokumen dengan *folder* yang telah dibuat. Pada Tabel 3.16 menyajikan deskripsi pengujian, hasil yang diharapkan, hasil pengujian, dan kesimpulan secara lebih lengkap dan komprehensif.

UNIVERSITAS MULTIMEDIA NUSANTARA

Tabel 3.16. Hasil pengujian dan evaluasi fitur *AI Grouping*

No.	Pengujian	Skenario	Hasil Pengujian	Kesimpulan
1	Memeriksa *grouping* secara otomatis	Pengguna dapat melakukan pengunggahan dokumen dan langsung otomatis terkategorikan	Penyimpanan dokumen sesuai dengan kategori atau *folder* secara otomatis dan menyimpan di Cloudinary	Sesuai dan Berhasil
2	Memastikan pembuatan kategori atau *folder*	Pengguna dapat membuat kategori atau *folder baru*	Pembuatan kategori atau *folder* berjalan dan menyimpan di Cloudinary	Sesuai dan Berhasil
3	Memastikan kondisi harus membuat kategori atau *folder* terlebih dahulu sebelum mengunggah dokumen	Pengguna harus membuat kategori atau *folder* terlebih dahulu sebelum mengunggah dokumen	Pembuatan kondisi berhasil	Sesuai dan Berhasil
4	Memastikan tampilan menu *Document Types Management* sesuai yang terdapat fitur *AI Grouping*	Pengguna dapat mengakses menu *Document Types Management* sesuai dengan desain	Tampilan menu *Document Types Management* sesuai dengan desain	Sesuai dan Berhasil

Lanjut pada halaman berikutnya

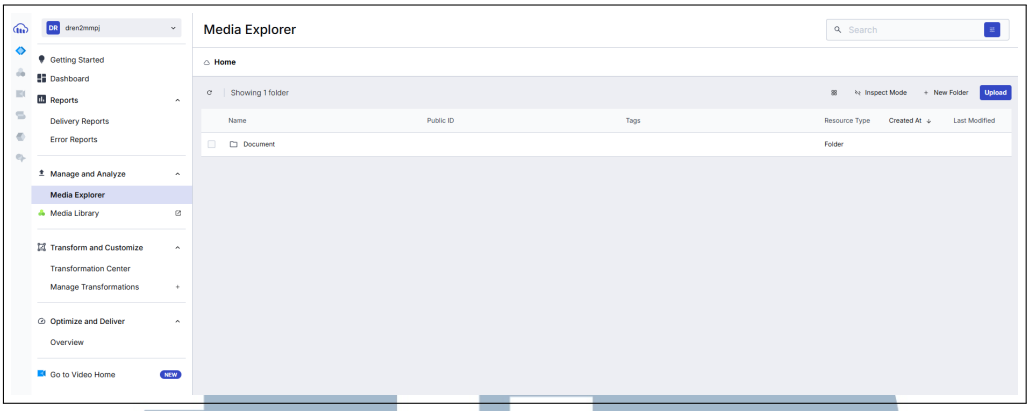
UNIVERSITAS MULTIMEDIA NUSANTARA

Lanjutan Tabel 3.16

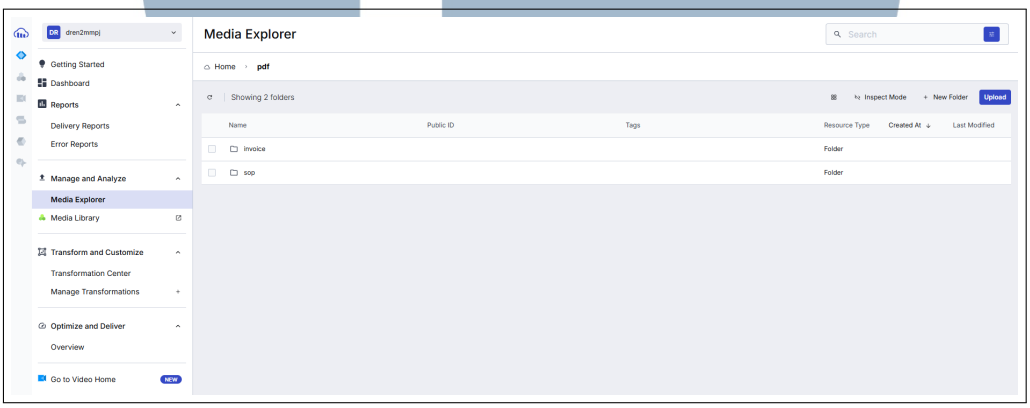
No.	Pengujian	Skenario	Hasil Pengujian	Kesimpulan
5	Memastikan tampilan menu *Documents Management* sesuai yang terdapat fitur *AI Grouping*	Pengguna dapat mengakses menu *Documents Management* sesuai dengan desain	Tampilan menu *Documents Management* sesuai dengan desain	Sesuai dan Berhasil
6	Memastikan adanya *error messages*	Pengguna dapat melihat informasi *error messages*	Tampilan dan informasi *error messages* muncul	Sesuai dan Berhasil
7	Memeriksa konsistensi tampilan yang responsif	Pengguna dapat mengakses menu *Document Types Management* dengan tampilan responsif	Tampilan menu *Document Types Management* responsif untuk semua ukuran *device*	Sesuai dan Berhasil
8	Memeriksa konsistensi tampilan yang responsif	Pengguna dapat mengakses menu *Documents Management* dengan tampilan responsif	Tampilan menu *Documents Management* responsif untuk semua ukuran *device*	Sesuai dan Berhasil

Adapun contoh *output* yang dapat dilihat adalah pada Gambar 3.58 yang berisikan tampilan penyimpanan dokumen sesuai dengan kategori atau *folder* yang telah dibuat. Selain itu, Gambar 3.57 merupakan tampilan penyimpanan dokumen hanya pada satu *folder* biasa tanpa ada kategori. Kedua *output* tersebut dapat dilihat tampilannya melalui Cloudinary.

UNIVERSITAS MULTIMEDIA NUSANTARA



Gambar 3.57. Tampilan sebelum pengembangan pada fitur *AI Grouping*



Gambar 3.58. Tampilan sesudah pengembangan pada fitur *AI Grouping*

Pengujian terhadap ketiga fitur utama pada Platform *AI Mastering Document* dilakukan menggunakan pendekatan *White Box*. Metode ini melibatkan pemeriksaan secara langsung terhadap kode program untuk memastikan tidak adanya kesalahan logika maupun struktur dalam implementasi. Apabila keluaran (*output*) yang dihasilkan tidak sesuai dengan ekspektasi, maka kode program akan dianalisis kembali dan dikompilasi ulang hingga memperoleh hasil yang tepat. Dengan demikian, pengguna platform sudah memiliki pemahaman terhadap cara kerja sistem secara umum [86]. Seluruh hasil pengujian tersebut telah melalui proses verifikasi dan disetujui oleh supervisor, sebagaimana tercantum pada Lampiran 7.

UNIVERSITAS MULTIMEDIA NUSANTARA

3.6 Kendala dan Solusi yang Ditemukan

3.6.1 Kendala

Selama proses pelaksanaan magang di perusahaan AI.DECE, proses pengembangan yang dilakukan pasti menghadapi berbagai tantangan teknis dan non-teknis yang muncul dalam proses pengembangan Platform *AI Mastering Document*. Kendala-kendala tersebut menjadi bagian penting dalam evaluasi proses pengembangan karena berkaitan langsung dengan kapabilitas dan performa dari fitur-fitur utama yang diimplementasikan, yaitu *AI Smart Chatbot*, *AI Smart Searching*, dan *AI Grouping*, beserta dengan tampilan antarmuka pengguna. Pemahaman terhadap kendala ini memberikan gambaran nyata mengenai permasalahan yang terjadi di lapangan serta menjadi pijakan dalam menentukan strategi pengembangan selanjutnya. Adapun rincian kendala yang ditemukan adalah sebagai berikut.

1. Terbatasnya dokumentasi internal yang belum terstandarisasi, sehingga proses integrasi dan *onboarding* tim baru cenderung lambat.
2. Inkonsistensi kualitas respons model LLM di tengah variasi beban dan konteks permintaan.
3. Tantangan dalam manajemen versi dependensi pustaka AI seperti LangChain, Weaviate, atau ChromaDB yang terkadang tidak kompatibel.
4. Penggunaan sistem operasi yang berbeda antar tim dalam mengembangkan suatu proyek berbasis teknologi.

3.6.2 Solusi

Menindaklanjuti berbagai kendala yang muncul selama proses pengembangan, hal yang dapat dilakukan adalah merumuskan solusi yang tepat dan aplikatif guna mengatasi permasalahan yang ada. Solusi yang dirancang tidak hanya difokuskan pada perbaikan teknis, tetapi juga mempertimbangkan aspek keberlanjutan sistem serta kemudahan integrasi dengan ekosistem layanan yang telah berjalan. Dengan pendekatan ini, setiap permasalahan ditangani secara sistematis agar hasil pengembangan dapat berjalan optimal. Adapun solusi-solusi yang diterapkan dalam menjawab kendala-kendala tersebut untuk setiap poin adalah sebagai berikut.

1. Menyusun panduan teknis dan prosedur kerja terstruktur, mencakup tata kelola repositori, standar penulisan kode, dan dokumentasi API yang terpusat.
2. Melakukan evaluasi performa LLM berkala, menerapkan *prompt engineering*, serta menggunakan *model ensemble* atau strategi fallback untuk menjaga konsistensi respons.
3. Menerapkan *dependency locking* dan pengujian kompatibilitas dengan menetapkan versi yang sama untuk semua pengembang.
4. Menyesuaikan sistem operasi berupa terdapat pergantian sintaks yang digunakan pada saat pengembangan atau menggunakan Docker lebih mendalam.

3.7 Hasil Pelaksanaan Magang

Proses pelaksanaan magang di perusahaan AI.DECE dapat dikatakan telah berhasil diselesaikan dengan melengkapi serangkaian tugas teknis dan dokumentasi yang mendukung kelancaran pengembangan Platform *AI Mastering Document*. Meskipun mengalami beberapa hambatan yang telah diuraikan pada Subbab 3.6.1, pelaksanaan kerja magang ini berhasil mendapatkan hasil yang dinilai sangat bagus oleh pihak perusahaan.

Berbagai kegiatan teknis dan dokumentasi telah terselesaikan untuk mendukung pengembangan Platform *AI Mastering Document*. Perancangan antarmuka pengguna dilakukan dengan *wireframe* menggunakan Figma yang kemudian diperkaya dengan *mockup* prototipe di Adobe Photoshop. Desain mencakup tata letak menu *AI Smart Chatbot*, menu *AI Smart Searching*, menu *Documents Management* dengan fitur *AI Grouping*, dan menu *Home* sehingga siap dipresentasikan kepada klien untuk tahap integrasi kepada perusahaan klien.

Selain itu, dokumentasi API *endpoint* disusun secara komprehensif, meliputi spesifikasi metode HTTP, parameter *input*, struktur respons, dan contoh panggilan melalui *Postman*. *Endpoint* tersebut mencakupi empat pembagian yaitu *Conversation Chatbots* untuk percakapan pada fitur menu *AI Smart Chatbot*, *Conversation* untuk percakapan pada fitur menu *AI Smart Searching*, *Document Type* untuk bagian kategori atau *folder*, dan *Document* untuk bagian dokumen PDF. Rincian ini memudahkan untuk mengembangkan *frontend* dalam mengintegrasikan fitur sesuai standar yang telah ditetapkan.

Adapun proses implementasi modul teknis dijalankan secara bertahap dengan pendekatan eksplorasi dan *trial-and-error*. Studi konsep *Retrieval-Augmented Generation* (RAG), penerapan *semantic embeddings*, serta pemanfaatan pustaka LangChain dan ChromaDB atau Weaviate berhasil menghasilkan sistem *memory management* yang memenuhi kebutuhan fungsional. Optimasi performa dan pengujian unit turut dilakukan untuk memastikan stabilitas setiap modul.

Terlebih lagi, integrasi hasil pengujian yang sebelumnya hanya tampil di *output* terminal kini telah beralih ke antarmuka web yang interaktif. Fitur *AI Smart Chatbot*, *AI Smart Searching*, dan *AI Grouping* dapat diakses langsung melalui browser dengan alur kerja yang intuitif dan respons waktu nyata. Semua komponen *backend* dan *frontend* telah diharmonisasikan untuk memberikan pengalaman pengguna yang optimal.

Keseluruhan penyampaian termasuk desain *wireframe*, dokumentasi *API*, kode *backend*, serta tampilan *frontend* yang sudah dikemas ke dalam Platform *AI Mastering Document* telah disiapkan untuk diserahkan dan dipresentasikan kepada klien. Klien tersebut dapat memilih untuk mengintegrasi fitur apa saja yang terdapat pada Platform *AI Mastering Document* kepada perusahaan klien itu sendiri. Kesiapan ini membuka peluang integrasi fitur ke dalam lingkungan produksi perusahaan klien tanpa hambatan yang signifikan.

UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA