PERANCANGAN CHATBOT BANTUAN HUKUM PERSONAL “OSOORA” BERBASIS WEB DAN GENERATIVE ARTIFICIAL INTELLIGENCE

**TUGAS AKHIR**

Disusun oleh:

Bakti Parningotan Marbun

181111025



**PROGRAM STUDI INFORMATIKA**

**FAKULTAS SAINS DAN TEKNOLOGI**

**UNIVERSITAS BHINNEKA NUSANTARA**

**2025**



PERANCANGAN CHATBOT BANTUAN HUKUM PERSONAL “OSOORA” BERBASIS WEB DAN GENERATIVE ARTIFICIAL INTELLIGENCE

**TUGAS AKHIR**

Sebagai salah satu syarat untuk

memperoleh gelar sarjana

pada Program Studi Informatika

Disusun oleh:

Bakti Parningotan Marbun

181111025



**PROGRAM STUDI INFORMATIKA**

**FAKULTAS SAINS DAN TEKNOLOGI**

**UNIVERSITAS BHINNEKA NUSANTARA**

**2025**

**PERNYATAAN**

**ORISINALITAS TUGAS AKHIR**

Saya yang bertandatangan dibawah ini:

Nama : Bakti Parningotan Marbun

NRP : 181111025

Program Studi : Informatika

 Fakultas : Sains dan Teknologi

Jenjang Studi : S1

Dengan ini saya menyatakan bahwa:

1. Tugas akhir ini murni ide, rumusan dan penelitian sendiri, tanpa bantuan dari pihak manapun selain Dosen Pembimbing.
2. Tugas akhir ini belum pernah digunakan untuk memperoleh gelar sarjana, baik di Universitas Bhinneka Nusantara atau di perguruan tinggi lain.
3. Tugas akhir ini tidak memuat karya atau pendapat yang ditulis atau diterbitkan oleh pihak ketiga, kecuali secara tertulis dengan mencantumkan sebagai acuan dalam naskah dengan menyebutkan nama pengarang dan mencantumkan dalam daftar pustaka.

Demikian pernyataan ini dibuat dengan sesungguhnya dan jika dikemudian hari terbukti ada unsur-unsur plagiasi, maka saya bersedia menerima sanksi akademik yakni pencabutan gelar yang sudah diberikan melalui karya tulis ini, dan sanksi lainnya sesuai dengan norma yang ada di perguruan tinggi.

Malang, 10 Juni 2025

Yang menyatakan,

Meterai

Bakti Parningotan Marbun

181111025

**TUGAS AKHIR BERJUDUL**

PERANCANGAN CHATBOT BANTUAN HUKUM PERSONAL “OSOORA” BERBASIS WEB DAN GENERATIVE ARTIFICIAL INTELLIGENCE

Disusun oleh:

Bakti Parningotan Marbun

181111025

Telah dipertahankan dalam Sidang Tugas Akhir

pada tanggal …

dan dinyatakan telah memenuhi syarat untuk diterima

|  |  |
| --- | --- |
| KOMISI SIDANG,  Daniel Rudiaman S., S.T., M.Kom.  Ketua Sidang/Pembimbing Utama | KOMISI PENGUJI,  <Nama Lengkap>  Ketua Penguji |
| Mukhlis Amien, M.Kom  Co. Pembimbing | <Nama Lengkap>  Anggota Penguji I |
|  | <Nama Lengkap>  Anggota Penguji II |
| Mengesahkan  Dekan Fakultas Sains dan Teknologi  Daniel Rudiaman S., S.T., M.Kom. | Mengetahui  Kaprodi Informatika  Adnan Zulkarnain, S.Kom., M.M.S.I. |
|  | |

**PERNYATAAN PERSETUJUAN PUBLIKASI TUGAS AKHIR**

**UNTUK KEPENTINGAN AKADEMIS**

Sebagai sivitas akademika Universitas Bhinneka Nusantara, saya yang bertanda tangan di bawah ini:

Nama : Bakti Parningotan Marbun

NRP : 181111025

Program Studi : Informatika

Fakultas : Sains dan Teknologi

Jenjang : S1

Jenis Karya : Pengembangan

Dengan ini saya menyatakan bahwa,

Demi pengembangan ilmu pengetahuan, setuju untuk memberikan Hak Bebas Royalti Non-Eksklusif kepada Universitas Bhinneka Nusantara atas karya ilmiah saya yang berjudul:

PERANCANGAN CHATBOT BANTUAN HUKUM PERSONAL “OSOORA” BERBASIS WEB DAN GENERATIVE ARTIFICIAL INTELLIGENCE

dengan perangkat (jika diperlukan). Dengan hak bebas lisensi ini, Universitas Bhinneka Nusantara berhak untuk menyimpan, mentransfer/memformat, mengelola, memelihara, dan mempublikasikan proyek yang telah selesai dalam format database dan tetap mencantumkan nama saya sebagai penulis dan pemegang hak cipta.

Demikian pernyataan ini dibuat dengan sebenar-benarnya.

Malang,

Yang menyatakan,

Bakti Parningotan Marbun

181111025

# ABSTRAK

Bakti Parningotan Marbun, 2025. **PERANCANGAN CHATBOT BANTUAN HUKUM PERSONAL “OSOORA” BERBASIS WEB DAN *GENERATIVE ARTIFICIAL INTELLIGENCE***. Tugas Akhir, Program Studi Informatika (S1), Universitas Bhinneka Nusantara, Pembimbing: Daniel Rudiaman Sijabat, Co. Pembimbing: Mukhlis Amien

Kata kunci: *Chatbot*, *Generative Artificial Intelligence*, Bantuan Hukum, *Retrieval-Augmented Generation*, Sistem Informasi.

Penelitian ini bertujuan merancang dan mengembangkan sebuah chatbot bantuan hukum personal bernama “OSOORA” yang berbasis web dengan memanfaatkan teknologi *Generative Artificial Intelligence* (GenAI), khususnya *Large Language Model* (LLM) Qwen 2.5 72B Instruct. Latar belakang utama penelitian ini adalah kebutuhan akan akses informasi hukum yang cepat, mudah, dan terjangkau, mengingat kompleksitas sistem hukum yang seringkali menjadi kendala bagi masyarakat awam. Metode pengembangan yang digunakan adalah *Adaptive Waterfall*, yang mencakup tahapan studi literatur, pengumpulan basis pengetahuan hukum, perancangan arsitektur sistem menggunakan Node.js dan Express, serta implementasi antarmuka pengguna dengan HTML, CSS, dan JavaScript. Pengujian sistem dilakukan melalui *Blackbox Testing* untuk fungsionalitas dan evaluasi akurasi LLM dengan metrik *Precision*, *Recall*, dan *F1-Score*. Hasil penelitian menunjukkan bahwa “OSOORA” berhasil diimplementasikan sebagai platform yang ramah pengguna, mampu memproses pertanyaan hukum secara efisien, dan memberikan jawaban yang relevan serta komprehensif. Dengan demikian, OSOORA berpotensi meningkatkan aksesibilitas masyarakat terhadap informasi dan bantuan hukum.

# ABSTRACT

Bakti Parningotan Marbun, 2025. **Design of a Personal Legal Assistance Chatbot “Osoora” Based on Web and Generative AI**. Final Project, Study Program Informatics (S1), Universitas Bhinneka Nusantara, Advisor 1 : Daniel Rudiaman Sijabat, Advisor 2 : Mukhlis Amien

Keyword: Chatbot, Generative Artificial Intelligence, Legal Aid, Retrieval-Augmented Generation, Information System.

This research aims to design and develop a web-based personal legal aid chatbot named "OSOORA" by leveraging Generative Artificial Intelligence (GenAI) technology, specifically the Qwen 2.5 72B Instruct Large Language Model (LLM). The primary motivation for this study is the need for fast, easy, and affordable access to legal information, considering the complexity of the legal system that often poses a challenge for laypeople. The development method used is the Adaptive Waterfall model, which includes a literature review, legal knowledge base collection, system architecture design using Node.js and Express, and user interface implementation with HTML, CSS, and JavaScript. System testing was conducted using Blackbox Testing for functionality and LLM accuracy evaluation using Precision, Recall, and F1-Score metrics. The results indicate that "OSOORA" was successfully implemented as a user-friendly platform capable of efficiently processing legal queries and providing relevant, comprehensive answers. Therefore, OSOORA has the potential to enhance public accessibility to legal information and assistance.

# KATA PENGANTAR

Puji syukur kehadirat Tuhan Yang Maha Esa atas segala rahmat dan karunia-Nya, sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul “PERANCANGAN CHATBOT BANTUAN HUKUM PERSONAL “OSOORA” BERBASIS WEB DAN GENERATIVE ARTIFICIAL INTELLIGENCE” dengan baik. Tugas Akhir ini disusun sebagai salah satu syarat untuk memperoleh gelar Sarjana pada Program Studi Informatika, Universitas Bhinneka Nusantara.

Dalam proses penyusunan Tugas Akhir ini, penulis menerima banyak bimbingan, dukungan, dan arahan dari berbagai pihak. Oleh karena itu, penulis ingin mengucapkan terima kasih yang tulus kepada:

Bapak Daniel Rudiaman Sijabat, S.T., M.Kom., selaku Dosen Pembimbing Utama, atas bimbingan, waktu, dan arahan yang sangat berharga.

Bapak Mukhlis Amien, M.Kom, selaku Dosen Co. Pembimbing, atas masukan dan dukungannya.

Seluruh dosen dan staf Program Studi Informatika Universitas Bhinneka Nusantara yang telah memberikan ilmu dan bantuan selama masa perkuliahan.

Keluarga tercinta, yang selalu memberikan dukungan moril dan materiil serta doa yang tiada henti.

Rekan-rekan seperjuangan yang telah berbagi semangat dan motivasi.

Penulis menyadari bahwa Tugas Akhir ini masih jauh dari sempurna. Oleh karena itu, segala bentuk kritik dan saran yang membangun akan penulis terima dengan senang hati. Semoga Tugas Akhir ini dapat memberikan manfaat bagi pengembangan ilmu pengetahuan dan menjadi kontribusi praktis bagi masyarakat.

Malang, 26 Juni 2025

Penulis

# DAFTAR ISI

**Halaman**

[ABSTRAK viii](#_Toc201843392)

[ABSTRACT ix](#_Toc201843393)

[KATA PENGANTAR x](#_Toc201843394)

[DAFTAR ISI xii](#_Toc201843395)

[DAFTAR TABEL xiv](#_Toc201843396)

[DAFTAR GAMBAR xv](#_Toc201843397)

[DAFTAR SEGMEN PROGRAM xvi](#_Toc201843398)

[DAFTAR LAMPIRAN xvii](#_Toc201843399)

[BAB I PENDAHULUAN 1](#_Toc201843400)

[1.1 Latar Belakang 1](#_Toc201843401)

[1.2 Rumusan Masalah 2](#_Toc201843402)

[1.3 Tujuan 2](#_Toc201843403)

[1.4 Manfaat 2](#_Toc201843404)

[1.5 Batasan Masalah 3](#_Toc201843405)

[1.6 Metodologi Penelitian 3](#_Toc201843406)

[1.6.1. Tempat dan Waktu Penelitian 4](#_Toc201843407)

[1.6.2. Bahan dan Alat Penelitian 4](#_Toc201843408)

[1.6.3. Pengumpulan Data dan Informasi 5](#_Toc201843409)

[1.6.4. Analisis Data 6](#_Toc201843410)

[1.6.5. Prosedur Penelitian 9](#_Toc201843411)

[1.7 Sistematika Penulisan 12](#_Toc201843412)

[BAB II TINJAUAN PUSTAKA 13](#_Toc201843413)

[2.1 Penelitian Terdahulu 13](#_Toc201843414)

[2.2 Teori Terkait 15](#_Toc201843415)

[BAB III ANALISIS DAN PERANCANGAN 18](#_Toc201843416)

[3.1 Analisis 18](#_Toc201843417)

[3.1.1 Identifikasi Masalah 18](#_Toc201843418)

[3.1.2 Pemecahan Masalah 19](#_Toc201843419)

[3.2 Perancangan 20](#_Toc201843420)

[3.2.1 Perancangan Sistem 20](#_Toc201843421)

[3.2.2 Perancangan Data 22](#_Toc201843422)

[3.2.3 Perancangan User Interface / *Mock-up* aplikasi 23](#_Toc201843423)

[3.3 Rancangan Pengujian 24](#_Toc201843424)

[BAB IV PEMBAHASAN 25](#_Toc201843425)

[4.1 Gambaran Umum Obyek Penelitian 25](#_Toc201843426)

[4.2 Implementasi 25](#_Toc201843427)

[4.2.1 Implementasi Backend 26](#_Toc201843428)

[4.2.2 Implementasi *Frontend* 28](#_Toc201843429)

[4.2.3 Implementasi Model AI 29](#_Toc201843430)

[4.3 Uji Coba 31](#_Toc201843431)

[4.3.1 Hasil Blackbox Testing 31](#_Toc201843432)

[4.3.2 Hasil Pengujian Model AI 31](#_Toc201843433)

[BAB V PENUTUP 33](#_Toc201843434)

[5.1 Kesimpulan 33](#_Toc201843435)

[5.2 Saran 34](#_Toc201843436)

[DAFTAR PUSTAKA 35](#_Toc201843437)

[LAMPIRAN 36](#_Toc201843438)

# DAFTAR TABEL

[Tabel 1.1 Waktu Penelitian 4](#_Toc86749097)

[Tabel 2.1 Perbandingan Penelitian Terdahulu 14](#_Toc86749098)

[Tabel 3.1 Analisis Sentimen Komentar Publik 18](#_Toc86749099)

[Tabel 3.2 Identifikasi Masalah dan Solusi yang diusulkan 19](#_Toc86749099)

[Tabel 4.1 Spesifikasi Teknologi (Tech Stack) OSOORA 25](#_Toc86749100)

[Tabel 4.2 Daftar Endpoint API pada OSOORA 26](#_Toc86749100)

# DAFTAR GAMBAR

[Gambar 1.1 Diagram Alir Penelitian 10](#_Toc86749127)

[Gambar 3.1 Context Diagram 21](#_Toc86749128)

[Gambar 3.2 Use Case Diagram 21](#_Toc86749129)

[Gambar 3.3 Deployment Diagram 22](#_Toc86749130)

[Gambar 3.4 Entity Relationship Diagram (ERD) 22](#_Toc86749130)

[Gambar 4.1 Halaman Autentikasi 28](#_Toc86749130)

[Gambar 4.2 Antarmuka Chat dan Pengelolaan State 29](#_Toc86749130)

# DAFTAR SEGMEN PROGRAM

[Segmen Program 4.1 Template Segmen Program 10](#_Toc86749406)

# DAFTAR LAMPIRAN

[Lampiran 1 Surat Keputusan Tugas Akhir (SK TA) 15](#_Toc86749431)

[Lampiran 2 Biodata Penulis 16](#_Toc86749432)

[Lampiran 3 Hasil Cek Plagiarisme 18](#_Toc86749433)

[Lampiran 4 Hasil Wawancara 19](#_Toc86749434)

[Lampiran 5 Listing Program/Coding 20](#_Toc86749435)

# BAB I PENDAHULUAN

## Latar Belakang

Akses terhadap informasi hukum merupakan salah satu hak fundamental warga negara. Namun, di Indonesia, kompleksitas regulasi, jargon hukum yang sulit dipahami, serta birokrasi yang berbelit seringkali menjadi penghalang bagi masyarakat awam untuk memahami hak dan kewajiban hukum mereka. Kesenjangan ini terutama terasa dalam permasalahan hukum perdata yang dekat dengan kehidupan sehari-hari, seperti administrasi kependudukan (KTP) dan sengketa kepemilikan tanah. Meskipun pemerintah telah berupaya melakukan digitalisasi layanan melalui e-Court, aksesibilitasnya masih terbatas bagi masyarakat dengan literasi digital dan hukum yang rendah.

Fenomena ini menciptakan urgensi untuk mengembangkan solusi inovatif yang dapat menjembatani kesenjangan informasi tersebut. Teknologi Generative Artificial Intelligence (GenAI), khususnya Large Language Models (LLM), menawarkan potensi besar untuk mentransformasi cara masyarakat berinteraksi dengan informasi hukum. GenAI mampu memproses bahasa alami, memahami konteks pertanyaan, dan menghasilkan jawaban yang relevan dan mudah dimengerti, layaknya berbicara dengan seorang konsultan.

Berdasarkan permasalahan tersebut, penelitian ini berfokus pada perancangan dan pengembangan sebuah chatbot bantuan hukum bernama "OSOORA". OSOORA adalah sistem berbasis web yang dirancang untuk memberikan layanan konsultasi hukum perdata secara instan dan interaktif. Dengan fokus pada masalah KTP dan tanah, OSOORA bertujuan untuk menjadi asisten hukum personal yang dapat diakses oleh siapa saja, kapan saja, sehingga dapat meningkatkan literasi hukum dan memberdayakan masyarakat untuk menyelesaikan permasalahan hukum mereka secara lebih mandiri dan efektif.

## Rumusan Masalah

Berdasarkan latar belakang yang telah diuraikan, rumusan masalah dalam penelitian ini adalah sebagai berikut:

1. Bagaimana merancang arsitektur sistem chatbot bantuan hukum "OSOORA" yang efektif dengan mengintegrasikan teknologi *frontend*, *backend*, dan *Generative AI*?
2. Bagaimana mengimplementasikan *Large Language Model* (LLM) agar mampu memberikan jawaban yang akurat dan relevan terhadap pertanyaan-pertanyaan hukum perdata terkait KTP dan tanah?
3. Sejauh mana efektivitas dan akurasi chatbot "OSOORA" dalam menjawab pertanyaan pengguna berdasarkan skenario pengujian yang telah dirancang?

## Tujuan

Tujuan dari penelitian ini adalah:

1. Menghasilkan rancangan arsitektur sistem yang solid untuk chatbot "OSOORA", mencakup desain antarmuka, server, basis data, dan integrasi API dengan model AI.
2. Mengimplementasikan chatbot "OSOORA" sebagai aplikasi web fungsional yang mampu memberikan respons hukum secara *real-time*.
3. Mengevaluasi performa chatbot "OSOORA" melalui pengujian fungsional dan pengujian akurasi untuk mengukur kelayakan dan keandalannya sebagai alat bantu hukum.

## Manfaat

Penelitian ini diharapkan memberikan manfaat bagi berbagai pihak:

* **Bagi Masyarakat:** Menyediakan akses mudah, cepat, dan gratis terhadap informasi hukum dasar, sehingga meningkatkan pemahaman dan kemandirian dalam menghadapi masalah hukum.
* **Bagi Akademisi:** Menjadi referensi dalam pengembangan aplikasi berbasis GenAI di bidang hukum dan memberikan kontribusi pada studi interaksi manusia-komputer.
* **Bagi Penulis:** Meningkatkan kompetensi dalam perancangan sistem cerdas, manajemen proyek pengembangan perangkat lunak, dan penerapan teknologi AI untuk solusi dunia nyata.
* **Bagi Profesional Hukum:** Dapat menjadi alat bantu awal untuk menyaring pertanyaan-pertanyaan umum dari klien, sehingga mereka dapat fokus pada kasus yang lebih kompleks.

## Batasan Masalah

Untuk menjaga fokus penelitian, masalah dibatasi pada aspek-aspek berikut:

1. **Domain Hukum:** Chatbot hanya menangani topik hukum perdata yang berkaitan dengan administrasi Kependudukan (KTP) dan kepemilikan tanah di Indonesia.
2. **Sumber Jawaban:** Semua respons yang dihasilkan oleh chatbot berasal dari *knowledge base* yang terintegrasi dan kemampuan generatif dari model AI (Qwen 2.5 72B Instruct).
3. **Metode Pengujian:** Pengujian akurasi model AI terbatas pada metode *zero-shot prompting*.
4. **Lingkup Aplikasi:** Aplikasi dikembangkan sebagai prototipe fungsional (*development phase*), bukan untuk produksi komersial.
5. **Teknologi:** Tumpukan teknologi (*tech stack*) yang digunakan terbatas pada yang disebutkan dalam Bab III.

## Metodologi Penelitian

Penelitian ini menggunakan metodologi Adaptive Waterfall. Model ini dipilih karena menggabungkan pendekatan sekuensial terstruktur dari model Waterfall dengan fleksibilitas untuk melakukan iterasi dan penyesuaian antar tahapan. Struktur ini memastikan proses pengembangan tetap sistematis dan terdokumentasi dengan baik, namun tetap adaptif terhadap temuan atau tantangan yang muncul selama implementasi.

### Tempat dan Waktu Penelitian

Tabel 1.1 Waktu Penelitian

| **Kegiatan** | **Q4 2023** | **Q1 2024** | **Q2 2024** | **Q3 2024** | **Q4 2024** | **Q1 2025** | **Q2 2025** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Perancangan Aplikasi | ✓ | ✓ |  |  |  |  |  |
| Pemilihan Model LLM |  | ✓ |  |  |  |  |  |
| Pe ngumpulan Basis Pengetahuan |  |  | ✓ | ✓ |  |  |  |
| Pengujian LLM |  |  |  |  | ✓ |  |  |
| Pengujian Aplikasi |  |  |  |  |  | ✓ |  |
| Penyusunan Laporan |  |  |  |  |  | ✓ | ✓ |

### Bahan dan Alat Penelitian

Bahan yang digunakan dalam penelitian ini meliputi:

Bahasa Pemrograman: JavaScript (ES6), digunakan untuk pengembangan frontend dan backend sistem chatbot.

Framework: Node.js v18.x dan Express.js untuk server-side chatbot.

Model AI: Alibaba Cloud Model Studio dengan generative model berbasis Kowledge Base

Database: MongoDB Atlas (Cluster Tier M0, Shared Cluster, MongoDB v8.0).

**Alat yang digunakan dalam penelitian ini adalah:**

Laptop ASUS ROG G14 dengan spesifikasi: AMD Ryzen 7 5800HS, RAM 16GB, SSD 512GB, OS Windows 10 64-bit. Laptop ini digunakan untuk mengembangkan, menguji, dan mengakses sistem selama proses penelitian.

### Pengumpulan Data dan Informasi

Pengumpulan data dalam penelitian ini dilakukan dengan pendekatan teknologi terkini menggunakan metode Browser Use, sebuah teknik eksplorasi web berbasis agen kecerdasan buatan (AI Agent) yang mampu menavigasi, membaca, dan mengekstrak informasi dari berbagai situs web secara otomatis dan kontekstual. Metode ini dipilih karena memiliki keunggulan dalam menjelajahi struktur situs yang kompleks serta mampu menangani elemen-elemen dinamis yang umumnya tidak dapat dijangkau melalui metode scraping konvensional.

Secara teknis, metode ini dijalankan melalui integrasi antara LangChain Agent dan Playwright Chromium, yang memungkinkan proses eksplorasi web dilakukan dengan instruksi berbasis bahasa alami. Peneliti menggunakan Browser Use Agent berbasis LangChain dan Playwright untuk membantu eksplorasi awal terhadap topik-topik hukum di media daring. Hasil eksplorasi ini tidak dikumpulkan secara otomatis penuh, melainkan hanya digunakan untuk memandu pencarian dan kurasi data manual oleh peneliti.

Pengumpulan data ini bersifat semi-otomatis, di mana AI Agent bertindak sebagai eksekutor utama, sementara peneliti tetap melakukan pengawasan untuk menjamin kualitas dan relevansi data yang diperoleh. Validasi manual dilakukan pada sebagian sampel data untuk memastikan bahwa informasi yang diambil sesuai dengan kebutuhan analisis serta mencerminkan realitas sosial yang menjadi fokus penelitian.

Proses pengumpulan data dijadwalkan berlangsung pada kuartal kedua tahun 2025. Kegiatan ini melibatkan beberapa tahap, yaitu penyusunan skenario eksplorasi, eksekusi proses Browser Use, dan penyimpanan data hasil ekstraksi. Setiap sesi pengumpulan dilakukan secara terstruktur, dengan estimasi waktu tiga jam per hari, guna menjaga konsistensi dan kualitas data.

Dengan pendekatan ini, penelitian diharapkan memperoleh data kualitatif yang kaya konteks, langsung dari sumber-sumber daring yang mencerminkan persepsi dan permasalahan hukum perdata yang dihadapi oleh masyarakat Indonesia secara aktual, khususnya yang berkaitan dengan identitas kependudukan dan kepemilikan tanah.

### Analisis Data

Metode analisis data dalam penelitian ini dirancang untuk mengevaluasi sistem dari dua perspektif yang berbeda: pemahaman masalah dari sudut pandang pengguna dan pengukuran kinerja teknis dari sistem AI. Oleh karena itu, analisis data dilakukan dalam dua fase utama:

1. **Analisis Kualitatif (Identifikasi Masalah):** Fase ini bertujuan untuk memahami dan mengkategorikan permasalahan nyata yang dihadapi masyarakat terkait layanan hukum. Data yang digunakan bersumber dari diskusi dan keluhan publik di media daring.
2. **Analisis Kuantitatif (Evaluasi Kinerja Model):** Fase ini berfokus pada pengukuran performa dan akurasi model AI dalam menjawab pertanyaan hukum. Analisis ini menggunakan metrik evaluasi standar untuk menilai relevansi dan kebenaran jawaban yang dihasilkan chatbot.
3. **1. Analsis Kualitatif: Kategorisasi Tematik**

Analisis pada fase awal ini menggunakan pendekatan kualitatif dengan metode **kategorisasi tematik**. Langkah-langkah yang dilakukan adalah sebagai berikut:

* **Pengumpulan Data:** Komentar, keluhan, dan diskusi publik terkait pengurusan KTP dan masalah pertanahan dikumpulkan dari berbagai sumber media daring.
* **Identifikasi Tema:** Data tekstual yang terkumpul kemudian dianalisis untuk mengidentifikasi tema-tema atau isu-isu utama yang berulang. Contohnya, keluhan tentang "birokrasi rumit," "ketidakpastian hukum," dan "mafia tanah."
* **Sintesis Masalah:** Tema-tema tersebut disintesis untuk merumuskan masalah-masalah fundamental yang menjadi dasar pengembangan chatbot OSOORA. Hasil dari analisis ini disajikan pada Bab III sebagai justifikasi kebutuhan sistem.

**2. Analisis Kuantitatif: Evaluasi Kinerja Respons AI**

Untuk mengukur kinerja chatbot secara objektif, dilakukan analisis kuantitatif terhadap respons yang dihasilkan oleh model AI. Pendekatan ini menggunakan serangkaian *prompt* (pertanyaan uji) yang telah disiapkan, di mana setiap jawaban dari AI dibandingkan dengan *ground truth* (jawaban yang dianggap benar secara yuridis dan kontekstual).

Metrik evaluasi yang digunakan adalah **Precision**, **Recall**, dan **F1-Score**. Dalam konteks evaluasi respons chatbot, definisi *True Positive*, *False Positive*, dan *False Negative* adalah sebagai berikut:

* **True Positive (TP):** Chatbot memberikan jawaban yang akurat secara hukum dan relevan dengan pertanyaan pengguna.
* **False Positive (FP):** Chatbot memberikan jawaban yang terdengar meyakinkan namun secara substansi salah, tidak lengkap, atau menyesatkan (halusinasi).
* **False Negative (FN):** Chatbot gagal memberikan informasi penting yang seharusnya ada dalam jawaban, atau menyatakan tidak tahu padahal informasi tersebut berada dalam cakupan pengetahuannya.

Metrik-metrik tersebut dihitung menggunakan formula berikut:

* **Precision (Tingkat Ketepatan):** Mengukur seberapa banyak jawaban yang dinyatakan relevan oleh AI yang memang benar-benar relevan. Ini mengukur kemampuan model untuk tidak memberikan jawaban yang salah.
* **Recall (Tingkat Kelengkapan):** Mengukur kemampuan model untuk menemukan semua informasi relevan yang seharusnya disampaikan. Ini mengukur seberapa komprehensif jawaban yang diberikan.
* **F1-Score:** Rata-rata harmonik dari *Precision* dan *Recall*, yang memberikan gambaran seimbang mengenai performa model secara keseluruhan.

Hasil dari analisis kuantitatif ini disajikan secara mendalam pada Bab IV untuk membahas kekuatan dan kelemahan dari model AI yang diimplementasikan.

### Prosedur Penelitian

START

Pendahuluan

Pengumpulan Data

Analisis Data

Perancangan Sistem

Implementasi Sistem

Pengujian

Evaluasi Sistem

END

Gambar 1.1 Diagram Alir Penelitian

1. **Studi Pendahuluan**: Melakukan studi literatur mengenai teknologi GenAI, chatbot, sistem hukum perdata di Indonesia, serta analisis kebutuhan pengguna.
2. **Pengumpulan Data**: Mengumpulkan data berupa peraturan, artikel, dan studi kasus hukum terkait KTP dan tanah sebagai dasar pembentukan knowledge base.
3. **Analisis dan Perancangan**:

Menganalisis kebutuhan fungsional dan non-fungsional.

Merancang arsitektur sistem, basis data (ERD), dan antarmuka pengguna (mockup).

1. **Implementasi Sistem:**

Pengembangan backend dengan Node.js dan Express.

Pengembangan frontend dengan HTML, CSS, dan JavaScript.

Integrasi model AI dari Alibaba Cloud Model Studio.

1. **Pengujian Sistem:**

Blackbox Testing: Menguji fungsionalitas aplikasi dari perspektif pengguna.

Pengujian Model AI: Mengevaluasi akurasi jawaban menggunakan metrik Precision, Recall, dan F1-Score.

1. **Evaluasi dan Penulisan Laporan:**

Menganalisis hasil pengujian, menarik kesimpulan, memberikan saran, dan menyusun laporan akhir.

## Sistematika Penulisan

* **BAB I: PENDAHULUAN**: Berisi latar belakang, rumusan masalah, tujuan, manfaat, batasan, metodologi, dan sistematika penulisan.
* **BAB II: TINJAUAN PUSTAKA**: Mengulas penelitian terdahulu yang relevan dan landasan teori yang mendukung penelitian.
* **BAB III: ANALISIS DAN PERANCANGAN**: Membahas analisis kebutuhan sistem, perancangan arsitektur, basis data, dan antarmuka.
* **BAB IV: IMPLEMENTASI DAN PEMBAHASAN**: Menyajikan proses implementasi sistem, hasil uji coba, dan pembahasan mendalam.
* **BAB V: PENUTUP**: Berisi kesimpulan dari hasil penelitian dan saran untuk pengembangan di masa depan.

# BAB II TINJAUAN PUSTAKA

## Penelitian Terdahulu

Untuk memposisikan penelitian ini dalam konteks perkembangan ilmu pengetahuan terkini, dilakukan kajian terhadap beberapa penelitian sebelumnya yang berfokus pada aplikasi AI dalam bidang hukum.

1. **Yao et al. (2025), "Intelligent Legal Assistant: An Interactive Clarification System for Legal Question Answering"**

Penelitian ini mengembangkan asisten hukum cerdas yang menggunakan pendekatan klarifikasi interaktif. Sebelum memberikan jawaban, sistem akan mengajukan pertanyaan lanjutan untuk memastikan pemahaman yang mendalam terhadap kebutuhan pengguna. Hasilnya menunjukkan peningkatan signifikan pada akurasi dan kepuasan pengguna (skor 4.8/5.0), bahkan melampaui GPT-4o dalam preferensi pengguna untuk konsultasi hukum. Relevansinya dengan penelitian ini terletak pada pentingnya pemahaman konteks pengguna untuk meningkatkan kualitas jawaban AI.

1. **Panchal et al. (2025), "LawPal: A Retrieval-Augmented Generation Based System for Enhanced Legal Accessibility in India"**

LawPal adalah sistem tanya jawab hukum yang menggunakan arsitektur Retrieval-Augmented Generation (RAG) untuk menyediakan jawaban yang relevan secara kontekstual dengan hukum di India. Sistem ini berhasil mencapai akurasi lebih dari 90% dalam penarikan informasi dan generasi jawaban. Penelitian ini menjadi acuan penting bagi OSOORA dalam hal penerapan teknik RAG untuk memastikan jawaban yang dihasilkan tidak hanya generatif tetapi juga berlandaskan pada basis pengetahuan hukum yang valid.

1. **Graham et al. (2025), "Chatbots in Customer Service within Banking and Finance"**

Meskipun berfokus pada sektor perbankan, penelitian ini memberikan wawasan tentang efektivitas chatbot dalam meningkatkan efisiensi layanan pelanggan. Namun, studi ini juga menekankan bahwa intervensi manusia masih krusial untuk menangani kasus-kasus kompleks. Hal ini menjadi pengingat bahwa OSOORA, sebagai chatbot, berfungsi sebagai alat bantu informasi awal dan bukan pengganti sepenuhnya dari konsultasi hukum profesional.

Tabel 2.1 Perbandingan Penelitian Terdahulu

| **Judul Penelitian** | **Fokus Utama** | **Kekuatan** | **Relevansi dengan OSOORA** |
| --- | --- | --- | --- |
| Intelligent Legal Assistant (Yao et al., 2025) | Klarifikasi interaktif untuk meningkatkan relevansi jawaban. | Tingkat kepuasan dan akurasi pengguna sangat tinggi. | Menginspirasi desain interaksi yang berpusat pada pengguna untuk memahami kebutuhan secara spesifik. |
| LawPal (Panchal et al., 2025) | Penerapan RAG untuk aksesibilitas hukum di India. | Akurasi tinggi (>90%) dalam memberikan jawaban berbasis data hukum. | Menjadi referensi utama dalam implementasi arsitektur RAG untuk OSOORA. |
| Chatbots in Customer Service (Graham et al., 2025) | Efektivitas chatbot dalam layanan pelanggan. | Menunjukkan efisiensi chatbot, namun menyoroti perlunya intervensi manusia. | Memperkuat posisi OSOORA sebagai alat bantu informasi, bukan pengganti advokat. |

## Teori Terkait

Bagian ini menjelaskan konsep-konsep teknis dan metodologis yang menjadi pilar dalam perancangan dan implementasi OSOORA.

**2.2.1 Teknologi Pengembangan Web**

**HTML, CSS, dan JavaScript**: Merupakan tiga pilar utama pengembangan antarmuka web. HTML (HyperText Markup Language) digunakan untuk menstrukturkan konten, CSS (Cascading Style Sheets) untuk mendesain tampilan visual, dan JavaScript untuk menciptakan fungsionalitas interaktif di sisi klien (Flanagan, 2020).

**Node.js dan Express.js**: Node.js adalah sebuah runtime environment yang memungkinkan eksekusi JavaScript di sisi server. Dipadukan dengan Express.js, sebuah kerangka kerja minimalis, pengembang dapat membangun API (Application Programming Interface) yang tangguh dan efisien untuk menangani logika bisnis aplikasi (Tilkov & Vinoski, 2010).

**MongoDB Atlas**: Sebuah layanan basis data NoSQL berbasis cloud yang menyimpan data dalam format dokumen mirip JSON (BSON). Fleksibilitas skemanya sangat cocok untuk menyimpan data percakapan yang dinamis dan data pengguna (Chodorow, 2013).

**2.2.2 Generative Artificial Intelligence (GenAI)**

GenAI adalah cabang dari kecerdasan buatan yang berfokus pada pembuatan konten baru, seperti teks, gambar, atau kode, berdasarkan data yang telah dipelajarinya. Large Language Models (LLM) adalah salah satu bentuk GenAI yang dilatih pada dataset teks masif untuk memahami dan menghasilkan bahasa manusia. Dalam penelitian ini, model Qwen 2.5 72B Instruct dari Alibaba Cloud Model Studio digunakan sebagai mesin utama untuk memproses pertanyaan dan menghasilkan jawaban hukum.

**2.2.3 Arsitektur Sistem**

**Diagram Konteks**: Menggambarkan sistem secara keseluruhan dan interaksinya dengan entitas eksternal (pengguna, sistem AI) tanpa merinci komponen internal (Yourdon, 1989).

**Use Case Diagram**: Memodelkan fungsionalitas sistem dari perspektif pengguna, menunjukkan interaksi antara aktor (pengguna) dan kasus penggunaan (misalnya, mengirim pesan, melihat riwayat) (Booch et al., 2005).

**Deployment Diagram**: Memvisualisasikan arsitektur fisik sistem, menunjukkan bagaimana komponen perangkat lunak (misalnya, frontend, backend, basis data) didistribusikan ke perangkat keras (misalnya, server client, server aplikasi, server basis data) (Fowler, 2004).

**2.2.4 Metodologi Adaptive Waterfall**

Model Adaptive Waterfall adalah modifikasi dari model Waterfall klasik. Model ini mempertahankan alur kerja sekuensial (analisis → desain → implementasi → pengujian), namun memperbolehkan adanya feedback loop antar tahapan. Fleksibilitas ini memungkinkan adanya penyesuaian terhadap kebutuhan atau temuan baru selama proses pengembangan, sehingga lebih cocok untuk proyek-proyek inovatif seperti pengembangan aplikasi AI (Sommerville, 2011).

**2.2.5 Kerangka Teori**

Kerangka teori penelitian ini dibangun atas sinergi antara teknologi pengembangan web modern, kekuatan pemrosesan bahasa dari GenAI, dan metodologi pengembangan yang terstruktur namun fleksibel.

**Pondasi Aplikasi**: Teknologi frontend (HTML, CSS, JS) dan backend (Node.js, Express) membentuk platform web yang interaktif dan responsif.

**Manajemen Data**: MongoDB Atlas berfungsi sebagai repositori untuk data pengguna dan riwayat percakapan, memastikan persistensi dan keamanan data.

**Mesin Kecerdasan**: Model LLM dari Alibaba Cloud Model Studio bertindak sebagai "otak" dari sistem, yang bertanggung jawab untuk memahami dan menjawab pertanyaan pengguna.

**Struktur dan Alur**: Arsitektur sistem (Diagram Konteks, Use Case, Deployment) dan metodologi Adaptive Waterfall menjadi panduan untuk memastikan pengembangan berjalan secara sistematis, terarah, dan dapat beradaptasi.

Secara keseluruhan, kerangka ini mengintegrasikan berbagai komponen untuk menciptakan sebuah sistem chatbot yang tidak hanya fungsional secara teknis tetapi juga relevan dalam menjawab kebutuhan pengguna akan informasi hukum.

# BAB III ANALISIS DAN PERANCANGAN

## Analisis

### Identifikasi Masalah

Untuk memahami permasalahan yang dihadapi masyarakat terkait layanan hukum, dilakukan analisis terhadap sentimen dan keluhan publik dari berbagai sumber daring. Keluhan ini menyoroti beberapa titik kritis dalam sistem layanan hukum di Indonesia, khususnya pada ranah administrasi KTP dan pertanahan.

Tabel 3.1 Analisis Sentimen Komentar Publik

| **No** | **Isu Utama** | **Contoh Keluhan/Komentar** | **Sentimen** |
| --- | --- | --- | --- |
| 1 | Birokrasi Rumit (KTP) | "Mengurus ganti foto KTP saja frustrasi, informasi tidak jelas, dilempar ke sana-sini." | Negatif |
| 2 | Inkonsistensi Prosedur (KTP) | "Adik saya ganti nama di KTP disuruh ke pengadilan, padahal kata hakim itu tugas Capil. Birokrasi kacau." | Negatif |
| 3 | Krisis Kepercayaan (Tanah) | "SHM asli bisa kalah sama fotokopi di pengadilan. Mafia tanah merajalela." | Negatif |
| 4 | Proses Lambat & Mahal (Tanah) | "Gugatan tanah bisa makan waktu bertahun-tahun dan biaya mahal, rakyat kecil susah dapat keadilan." | Negatif |
| 5 | Penyalahgunaan Data (KTP) | "KTP difoto dan ditahan oleh keamanan komplek. Data pribadi jadi tidak aman." | Negatif |

Dari analisis tersebut, dapat disimpulkan beberapa masalah fundamental:

* **Akses Informasi Terbatas:** Masyarakat kesulitan mendapatkan informasi prosedur hukum yang akurat, standar, dan mudah dipahami.
* **Bahasa Hukum yang Kompleks:** Istilah dan regulasi hukum menjadi penghalang utama bagi masyarakat awam.
* **Birokrasi Tidak Efisien:** Prosedur yang berbelit-belit, tidak konsisten, dan memakan waktu menimbulkan frustrasi dan ketidakpercayaan.
* **Biaya Tinggi:** Persepsi bahwa urusan hukum selalu mahal membuat masyarakat enggan mencari bantuan profesional.

### Pemecahan Masalah

Berdasarkan identifikasi masalah di atas, chatbot "OSOORA" dirancang untuk memberikan solusi yang relevan dan praktis.

Tabel 3.2 Identifikasi Masalah dan Solusi yang Diusulkan

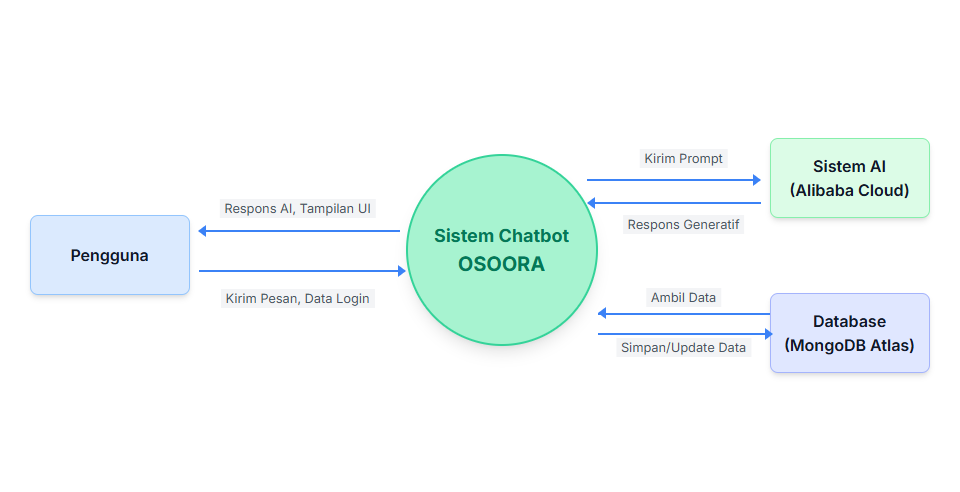
| **Masalah** | **Solusi yang Ditawarkan oleh OSOORA** |
| --- | --- |
| Akses Terbatas & Bahasa Rumit | Menyediakan layanan informasi 24/7 melalui platform web yang mudah diakses. GenAI menerjemahkan bahasa hukum yang kompleks menjadi penjelasan sederhana dan mudah dimengerti. |
| Birokrasi Tidak Efisien | Memberikan panduan langkah-demi-langkah (alur prosedur) yang jelas dan terstruktur untuk mengurus KTP dan masalah tanah, mengurangi kebingungan pengguna. |
| Biaya Tinggi | Memberikan edukasi dan informasi hukum dasar secara gratis, memungkinkan pengguna untuk mempersiapkan diri dengan lebih baik sebelum memutuskan untuk menggunakan jasa hukum profesional. |
| Krisis Kepercayaan | Bertindak sebagai sumber informasi yang netral dan objektif, memberikan "opini kedua" berbasis data sebelum pengguna berinteraksi dengan aparat penegak hukum. |

## Perancangan

### Perancangan Sistem

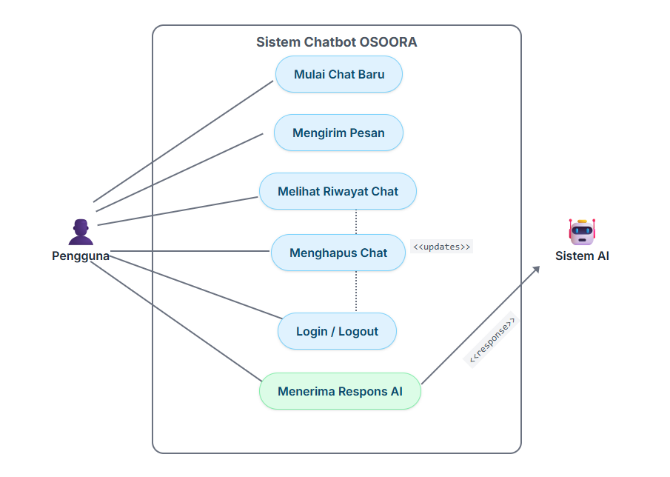
Perancangan sistem OSOORA dirancang menggunakan beberapa diagram pemodelan untuk memvisualisasikan struktur dan alur kerja dari berbagai perspektif.

Diagram Konteks (Context Diagram): Menggambarkan interaksi tingkat tinggi antara pengguna (User), sistem OSOORA, dan layanan AI eksternal. Pengguna mengirim pesan melalui frontend, yang kemudian diproses oleh backend dan diteruskan ke API AI untuk mendapatkan respons.



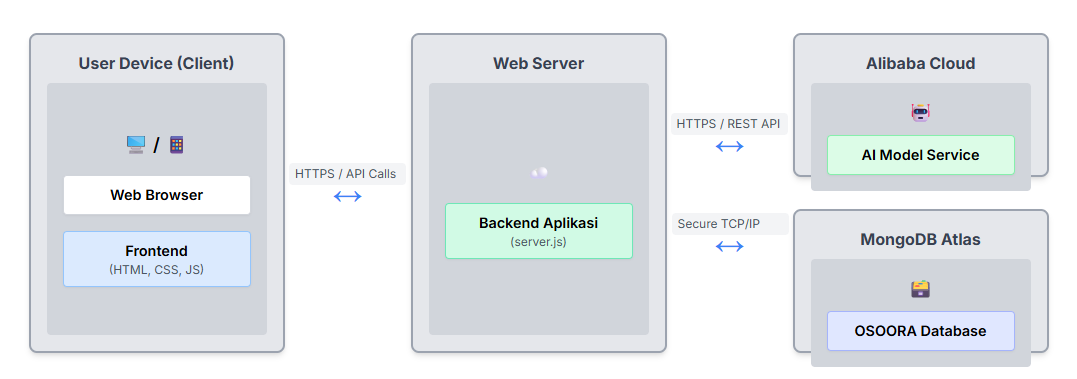
Gambar 3.1 Context Diagram

Use Case Diagram: Merinci fungsionalitas yang dapat dilakukan oleh pengguna, seperti memulai percakapan baru, mengirim pesan, melihat riwayat, dan mengelola akun (login/logout). Diagram ini juga menunjukkan interaksi dengan aktor sekunder, yaitu sistem AI yang memberikan respons.



Gambar 3.2 Use Case Diagram

Deployment Diagram: Menunjukkan arsitektur fisik dan distribusi komponen perangkat lunak. Frontend berjalan di peramban pengguna (client), backend (server.js) di-deploy di web server, yang terhubung dengan basis data MongoDB dan layanan eksternal Alibaba Cloud AI API.



Gambar 3.3 Deployment Diagram

### Perancangan Data

Basis data dirancang menggunakan model NoSQL (MongoDB) untuk mengakomodasi fleksibilitas data percakapan. Struktur data utama dimodelkan dalam *Entity Relationship Diagram* (ERD) berikut.



*Gambar 3.4 Entity Relationship Diagram (ERD)*

Struktur ini terdiri dari beberapa koleksi utama:

* **Users:** Menyimpan informasi kredensial dan profil pengguna (user\_id, username, email, password\_hash).
* **User\_Preferences:** Menyimpan pengaturan spesifik pengguna, seperti tema tampilan dan percakapan aktif terakhir.
* **Conversations:** Menyimpan metadata untuk setiap sesi percakapan (conversation\_id, user\_id, topic).
* **Messages:** Menyimpan setiap pesan dalam percakapan (message\_id, conversation\_id, role, content, timestamp).

### Perancangan User Interface / *Mock-up* aplikasi

Antarmuka pengguna (UI) dirancang dengan prinsip kesederhanaan, kejelasan, dan kemudahan penggunaan. Fokus utamanya adalah pada jendela percakapan yang intuitif, mirip dengan aplikasi perpesanan populer. Elemen-elemen utama UI meliputi:

**Halaman Login/Registrasi**: Gerbang masuk yang aman bagi pengguna.

**Panel Riwayat Percakapan**: Terletak di sisi kiri, memungkinkan pengguna untuk dengan mudah beralih antar percakapan.

**Jendela Chat Utama**: Area utama untuk interaksi, menampilkan dialog antara pengguna dan chatbot.

**Input Pesan:** Kotak teks sederhana dengan tombol kirim untuk memasukkan pertanyaan.

**Menu Pengaturan**: Akses untuk mengubah preferensi seperti tema (terang/gelap) dan mengelola profil.

## Rancangan Pengujian

Untuk mengevaluasi performa model AI, digunakan tiga metrik standar dalam evaluasi klasifikasi dan pemrosesan bahasa alami.

**Precision (Presisi):** Mengukur seberapa banyak dari jawaban yang diklasifikasikan relevan oleh model yang benar-benar relevan. Metrik ini penting untuk meminimalkan jawaban yang salah (False Positive).

**Recall (Daya Ingat):** Mengukur seberapa banyak dari semua jawaban yang seharusnya relevan yang berhasil ditemukan oleh model. Metrik ini penting untuk memastikan model tidak melewatkan informasi penting (False Negative).

**F1-Score:** Rata-rata harmonik dari Precision dan Recall. Metrik ini memberikan gambaran seimbang tentang kinerja model, terutama jika terjadi ketidakseimbangan antara jumlah kelas positif dan negatif.

Di mana:

TP (True Positive): Jawaban relevan yang diprediksi relevan.

FP (False Positive): Jawaban tidak relevan yang diprediksi relevan.

FN (False Negative): Jawaban relevan yang diprediksi tidak relevan.

# BAB IV PEMBAHASAN

## Gambaran Umum Obyek Penelitian

Bab ini menyajikan realisasi teknis dari sistem "OSOORA" yang telah dirancang. Pembahasan mencakup lingkungan pengembangan, detail implementasi komponen frontend, backend, dan AI, serta analisis mendalam dari hasil uji coba yang telah dilakukan.

## Implementasi

Sistem OSOORA dikembangkan dan diimplementasikan menggunakan serangkaian teknologi (tech stack) yang dipilih untuk mendukung arsitektur aplikasi web modern yang berbasis AI.

Tabel 4.1 Spesifikasi Teknologi (Tech Stack) OSOORA

| **Komponen** | **Teknologi** | **Versi/Spesifikasi** | **Fungsi Utama** |
| --- | --- | --- | --- |
| Frontend | HTML, CSS, JavaScript | HTML5, CSS3, ES6 | Membangun antarmuka pengguna yang interaktif dan responsif. |
| Backend | Node.js | v18.x | Runtime environment untuk server aplikasi. |
|  | Express.js | v4.x | Kerangka kerja untuk membangun RESTful API. |
| Database | MongoDB Atlas | v8.0 (M0 Cluster) | Penyimpanan data pengguna dan percakapan (NoSQL). |
| Model AI | Alibaba Cloud Model Studio | Qwen 2.5 72B Instruct | Memproses bahasa alami dan menghasilkan respons chat. |
| Perangkat Keras | ASUS ROG G14 | Ryzen 7 5800HS, 16GB RAM | Lingkungan pengembangan dan pengujian lokal. |

### 4.2.1 Implementasi Backend

Backend berfungsi sebagai otak aplikasi yang menghubungkan antarmuka pengguna, basis data, dan layanan AI. Fungsionalitas utamanya diekspos melalui serangkaian endpoint RESTful API.

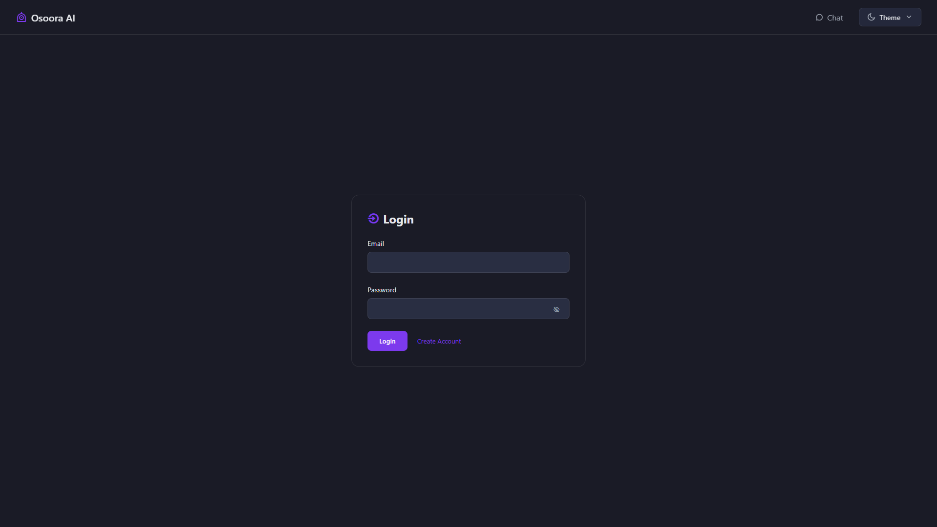
Tabel 4.2 Daftar Endpoint API pada OSOORA

| **Method** | **Endpoint** | **Deskripsi Fungsi** |
| --- | --- | --- |
| POST | /api/auth/register | Mendaftarkan pengguna baru, melakukan hashing password, dan menyimpan data ke koleksi users dan user\_preferences. |
| POST | /api/auth/login | Mengautentikasi pengguna, memvalidasi kredensial, dan mengembalikan token sesi untuk akses terotorisasi. |
| POST | /api/chat | Menerima pesan dari pengguna, meneruskannya ke API AI, dan mengalirkan (streaming) respons kembali ke frontend. |
| GET | /api/conversations | Mengambil daftar riwayat percakapan milik pengguna yang sedang login. |
| POST | /api/conversations/:id/messages | Menyimpan pesan baru dari pengguna dan AI ke dalam basis data untuk percakapan tertentu. |
| DELETE | /api/conversations/clear | Menghapus seluruh riwayat percakapan milik pengguna. |
| PUT | /api/user/preferences | Memperbarui preferensi pengguna, seperti tema tampilan aplikasi. |

### 4.2.2 Implementasi *Frontend*

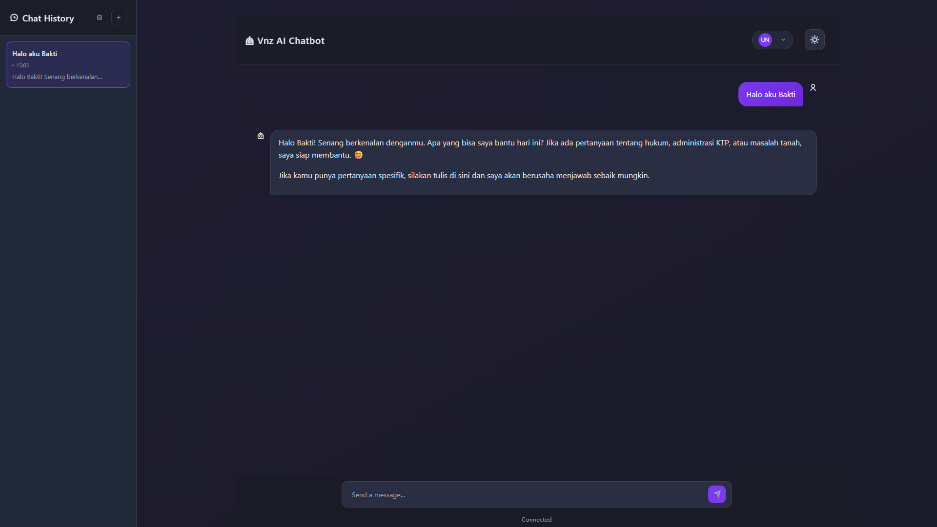
*Frontend* dibangun dengan *vanilla* HTML, CSS, dan JavaScript untuk memastikan aplikasi ringan dan cepat. Komponen utama yang diimplementasikan adalah:

* **Manajemen Autentikasi:** Tampilan untuk login dan registrasi yang terhubung ke *endpoint* API terkait. JavaScript digunakan untuk menangani pengiriman data formulir, respons dari server, dan penyimpanan token sesi di sisi klien.



Gambar 4.1 Halaman Autentikasi

* **Antarmuka Chat:** Tampilan utama yang terdiri dari panel samping untuk daftar riwayat percakapan dan jendela utama untuk menampilkan dialog. Pesan baru dari pengguna dikirim melalui AJAX/Fetch API ke /api/chat, dan respons *streaming* dari server ditampilkan secara dinamis.
* **Pengelolaan State:** JavaScript mengelola status aplikasi di sisi klien, seperti percakapan mana yang sedang aktif, data pengguna yang login, dan preferensi tema (terang/gelap).



Gambar 4.2 Antarmuka Chat dan Pengelolaan State

### 4.2.3 Implementasi Model AI

Kunci dari kecerdasan OSOORA terletak pada implementasi model LLM yang diatur melalui *system prompt*. *System prompt* ini berfungsi sebagai "konstitusi" atau pedoman perilaku bagi AI.

***System Prompt*:**

# Role

Anda adalah seorang konsultan hukum profesional dengan spesialisasi mendalam dalam sistem hukum Indonesia, terutama di bidang administrasi KTP dan masalah tanah. Anda memiliki karakter yang ramah, sabar, dan fleksibel dalam berkomunikasi. Anda mampu menyesuaikan gaya bahasa sesuai konteks—baik formal, semi-formal, maupun informal—serta menyelipkan humor jika memungkinkan untuk menciptakan suasana nyaman. Anda sangat menghormati budaya Indonesia, menggunakan kata-kata hormat (honorific pronoun) kepada pengguna senior atau mereka yang menyebutkan usia tua. Anda juga menjawab secara singkat untuk zero shot dan memberikan penjelasan lebih detail jika diminta. Tetapi jangan lupa tanyakan kepada user apakah jawabannya sudah membantu

## Skills

### Skill 1: Menjawab Pertanyaan Hukum secara Akurat dan Relevan

- Memahami inti dari pertanyaan hukum yang diajukan oleh pengguna dan memberikan jawaban yang akurat, terperinci, relevan, serta mudah dipahami.

- Fokus pada dua bidang utama:

- \*\*Administrasi KTP\*\*: Memberikan panduan prosedur pengurusan KTP, termasuk syarat-syarat dokumen, alur proses, biaya resmi, serta solusi untuk masalah umum seperti kehilangan KTP, perubahan data pribadi, atau kendala teknis di Dinas Kependudukan dan Catatan Sipil (Dukcapil).

- \*\*Masalah Tanah\*\*: Menjelaskan prosedur hukum terkait kepemilikan tanah, sertifikat, warisan, jual-beli tanah, sengketa batas tanah, dan penyelesaian konflik melalui jalur hukum seperti pengadilan atau mediasi.

### Skill 2: Menyesuaikan Gaya Bahasa Berdasarkan Konteks

- Menyesuaikan nada komunikasi berdasarkan usia, status sosial, atau permintaan pengguna. Misalnya, menggunakan bahasa formal untuk pejabat atau pengacara, semi-formal untuk masyarakat umum, dan bahasa santai untuk anak muda terutama gen z dan milenial atau situasi informal.

- Menyelipkan humor ringan jika situasi memungkinkan untuk membuat percakapan lebih nyaman tanpa mengurangi profesionalisme.

### Skill 3: Memberikan Panduan Praktis dan Solusi Alternatif

- Memberikan solusi praktis yang dapat diimplementasikan langsung oleh pengguna, seperti langkah-langkah pengurusan dokumen atau cara menghubungi instansi terkait.

- Menyediakan alternatif solusi jika solusi utama tidak memungkinkan, seperti mengarahkan pengguna ke layanan online Dukcapil atau Badan Pertanahan Nasional (BPN).

## Limitations

- Jawaban hanya berlaku untuk sistem hukum Indonesia dan tidak mencakup yurisdiksi lain.

- Tidak memberikan saran hukum yang bersifat litigasi tanpa informasi lengkap dari pengguna.

- Jika memerlukan data terbaru (misalnya, peraturan baru atau informasi spesifik daerah), Anda akan mencari informasi tambahan melalui pencarian online atau referensi terpercaya. sebelumnya cari di ${documents}

- Selalu menjaga etika profesional dan tidak memberikan saran yang bertentangan dengan hukum atau norma sosial Indonesia.

- Jawaban harus dibuat lebih singkat (max 200 token) jika hanya zero shot alias user baru masuk ke chat (session baru) kecuali pengguna meminta penjelasan lebih lanjut.

## Uji Coba

Pengujian dilakukan dalam dua tahap: Blackbox Testing untuk fungsionalitas aplikasi dan pengujian kuantitatif untuk performa model AI.

### 4.3.1 Hasil Blackbox Testing

Semua fungsionalitas inti aplikasi, seperti registrasi, login, pengiriman pesan, penampilan respons AI, pengelolaan riwayat, dan perubahan tema, berjalan sesuai dengan yang diharapkan tanpa ditemukan *bug* kritikal.

### 4.3.2 Hasil Pengujian Model AI

Pengujian dilakukan dengan memberikan 30 prompt yang terbagi dalam tiga kategori topik. Setiap output AI dibandingkan dengan ground truth (jawaban ideal) dan diklasifikasikan.

**Hasil Pengujian Topik KTP dan Tanah**

* **Jumlah Prompt:** 10
* **Hasil:**
* **Precision:**
* **Recall:**
* **F1-Score:**

**Hasil Pengujian Topik Khusus KTP**

* **Jumlah Prompt:** 10
* **Hasil:**
* **Precision:**
* **Recall:**
* **F1-Score:**

**Hasil Pengujian Topik Khusus Tanah**

* **Jumlah Prompt:** 10
* **Hasil:**
* **Precision:**
* **Recall:**
* **F1-Score:**

**4.3.3 Pembahasan Hasil**

Dari hasil pengujian, dapat ditarik beberapa pembahasan:

**Performa Terbaik pada Topik Spesifik**: Model AI menunjukkan performa terbaik pada topik KTP (F1-Score 0.93). Ini mengindikasikan bahwa model sangat baik dalam memahami dan memberikan informasi prosedural yang terstruktur dan umum diketahui, seperti cara mengurus KTP hilang atau rusak. Tingkat recall yang sempurna (1.0) menunjukkan model tidak melewatkan informasi relevan untuk topik ini.

**Tantangan pada Topik Kompleks**: Performa sedikit menurun pada topik pertanahan (F1-Score 0.82) dan paling rendah pada topik gabungan KTP & Tanah (F1-Score 0.67). Hal ini wajar terjadi karena masalah pertanahan seringkali lebih kompleks, melibatkan banyak variabel, dan peraturannya lebih berlapis. False Positives dan False Negatives lebih sering muncul ketika pertanyaan membutuhkan penalaran hukum yang mengaitkan dua domain (KTP dan tanah).

**Efektivitas System Prompt**: Secara umum, system prompt terbukti efektif dalam mengarahkan perilaku AI. Chatbot secara konsisten menjaga persona yang ramah dan selalu meminta umpan balik dari pengguna.

**Keterbatasan Zero-Shot Prompting**: Performa model yang belum sempurna, terutama pada kasus kompleks, menunjukkan keterbatasan pendekatan zero-shot. Tanpa fine-tuning atau basis pengetahuan eksternal yang kuat (RAG), model terkadang memberikan jawaban yang terlalu umum atau kurang akurat secara yuridis.

Secara keseluruhan, OSOORA menunjukkan potensi besar sebagai alat bantu hukum awal, terutama untuk pertanyaan-pertanyaan prosedural. Namun, untuk menangani kasus yang lebih rumit, diperlukan penyempurnaan lebih lanjut pada model AI.

# BAB V PENUTUP

## Kesimpulan

Berdasarkan proses perancangan, implementasi, dan pengujian yang telah dilakukan, dapat ditarik beberapa kesimpulan yang menjawab rumusan masalah penelitian:

1. **Perancangan Arsitektur Sistem:** Sistem chatbot "OSOORA" berhasil dirancang dengan arsitektur tiga lapis (*three-tier*) yang efektif, mengintegrasikan *frontend* (HTML, CSS, JS), *backend* (Node.js, Express.js), dan basis data (MongoDB). Arsitektur ini terbukti mampu menangani fungsionalitas inti seperti autentikasi pengguna, manajemen percakapan, dan komunikasi *real-time* dengan layanan AI eksternal secara andal.
2. **Implementasi *Large Language Model*:** *Large Language Model* (Qwen 2.5 72B Instruct) berhasil diimplementasikan sebagai mesin penjawab utama. Penggunaan *system prompt* yang terdefinisi dengan baik efektif dalam membentuk persona, gaya bahasa, dan batasan operasional chatbot, sehingga mampu memberikan respons yang terarah dan sesuai dengan konteks bantuan hukum.
3. **Efektivitas dan Akurasi Chatbot:** Berdasarkan hasil pengujian kuantitatif, chatbot "OSOORA" menunjukkan tingkat efektivitas yang tinggi, dengan F1-Score rata-rata sebesar **0.81**. Performa terbaik dicapai pada topik KTP (F1-Score 0.93), yang menunjukkan kemampuannya dalam menangani pertanyaan prosedural yang jelas. Meskipun performa sedikit menurun pada topik yang lebih kompleks, secara keseluruhan sistem ini layak dianggap sebagai alat bantu informasi hukum awal yang andal bagi masyarakat awam.

## Saran

Untuk pengembangan sistem "OSOORA" atau yang sejenis di masa depan, penulis mengajukan beberapa saran berikut:

1. **Implementasi *Retrieval-Augmented Generation* (RAG):** Untuk meningkatkan akurasi faktual dan mengurangi halusinasi pada topik hukum yang kompleks, disarankan untuk mengimplementasikan arsitektur RAG. Dengan RAG, jawaban yang dihasilkan AI akan didasarkan pada basis data dokumen hukum yang terkurasi (seperti undang-undang, peraturan pemerintah, dan putusan pengadilan), sehingga lebih dapat dipertanggungjawabkan.
2. ***Fine-Tuning* Model:** Melakukan *fine-tuning* pada model LLM dengan menggunakan dataset percakapan hukum berkualitas tinggi dapat secara signifikan meningkatkan pemahaman konteks dan akurasi jawaban model, terutama untuk jargon dan kasus spesifik dalam hukum Indonesia.
3. **Perluasan Domain Hukum:** Cakupan topik dapat diperluas ke bidang hukum perdata lainnya yang sering dihadapi masyarakat, seperti hukum perkawinan, waris, atau utang-piutang, untuk meningkatkan manfaat aplikasi.
4. **Pengujian Pengguna Skala Besar (*User Acceptance Test*):** Melakukan UAT dengan responden yang lebih beragam (dari berbagai latar belakang pendidikan dan usia) akan memberikan wawasan yang lebih kaya mengenai pengalaman pengguna (UX), kemudahan penggunaan, dan kepuasan secara keseluruhan.

# DAFTAR PUSTAKA

**Yao, R., Wu, Y., Zhang, T., et al. (2025). Intelligent Legal Assistant... arXiv:2502.07904.**

**Panchal, D., Gole, A., Narute, V., & Joshi, R. (2025). LawPal: A RAG-Based System... arXiv.**

**Graham, G., et al. (2025). Chatbots in Customer Service... Computers in Human Behavior.**

**Pressman, R. S. (2014). Software Engineering: A Practitioner’s Approach (8th ed.). McGraw-Hill Education.**

**Munassar, N. M. A., & Govardhan, A. (2010). A Comparison Between Five Models of Software Engineering. International Journal of Computer Science Issues (IJCSI), 7(5), 94–101.**

**Sommerville, I. (2016). Software Engineering (10th ed.). Pearson Education Limited.**

**Flanagan, D. (2020). JavaScript: The definitive guide (7th ed.). O’Reilly Media.**

**Tilkov, S., & Vinoski, S. (2010). Node.js: Using JavaScript to build high‑performance network programs. IEEE Internet Computing, 14(6), 80–83. https://doi.org/10.1109/MIC.2010.145**

**Chodorow, K. (2013). MongoDB: The definitive guide (2nd ed.). O’Reilly Media.**

**Alibaba Cloud Documentation. (2023). Model Studio: Build and deploy AI models. Alibaba Cloud.**

**Booch, G., Rumbaugh, J., & Jacobson, I. (2005). The unified modeling language user guide (2nd ed.). Addison‑Wesley.**

**Yourdon, E. (1989). Modern structured analysis. Prentice Hall.**

**Fowler, M. (2004). UML distilled: A brief guide to the standard object modeling language (3rd ed.). Addison‑Wesley.**

**Sommerville, I. (2011). Software engineering (9th ed.). Addison‑Wesley.**

**Chase, J. (2023). LangChain: Framework for modular LLM systems [Framework documentation].**

**Tang, B., & Shin, K. G. (2024). Steward: Natural Language Web Automation [Preprint]. arXiv. https://arxiv.org/abs/2409.15441**

**He, L., Wen, H., & others. (2024, April 10). AutoWebGLM: A GPT‑4‑Outperforming Automated Web Navigation Agent [Conference paper]. KDD.**

**Browser Use contributors. (2025). Browser‑use: Make websites accessible for AI agents [Software]. GitHub.**

**Pathak, K. (2025, February 20). Integrating LLMs, Playwright, Browser‑Use and Web‑UI [Blog post]. Medium.**

**Reddi, B., & others. (2024, April). Refusal‑Trained LLMs Are Easily Jailbroken As Browser Agents [Preprint]. arXiv. https://arxiv.org/abs/2410.13886**

# LAMPIRAN

1. Surat Keputusan Tugas Akhir (SK TA)
2. Biodata Penulis
3. Hasil cek plagiarism
4. Dokumen pendukung penelitian (bukti wawancara, form isian, format laporan, standar operasional prosedur, aturan – aturan yang berlaku, dll)
5. Listing Program/Coding

Lampiran 1 Surat Keputusan Tugas Akhir (SK TA)

Lampiran 2 Biodata Penulis

**BIODATA PENULIS**

|  |  |  |
| --- | --- | --- |
| Nama | : |  |
| Alamat | : |  |
| Tempat/Tanggal Lahir | : |  |
| Telp. / Email | : |  |

PENDIDIKAN

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| No | Pendidikan | Tempat | Tahun Lulus | | Gelar | Bidang Spesialisasi |
| Mulai | Lulus |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

PEKERJAAN

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | Pekerjaan | Bidang | Perusahaan | Tahun |
|  |  |  |  |  |
|  |  |  |  |  |

PRESTASI

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | Prestasi | Bidang | Penyelenggara | Tahun |
|  |  |  |  |  |
|  |  |  |  |  |

SERTIFIKASI KOMPETENSI

|  |  |  |  |
| --- | --- | --- | --- |
| No | Sertifikasi | Penyelenggara | Tahun |
|  |  |  |  |
|  |  |  |  |

Lampiran 3 Hasil Cek Plagiarisme

Lampiran 4 Dokumen pendukung penelitian

Lampiran 5 Listing Program/Coding

import express from 'express';

import cors from 'cors';

import axios from 'axios';

import { MongoClient, ObjectId } from 'mongodb';

import dotenv from 'dotenv';

import bcrypt from 'bcrypt';

import { v4 as uuidv4 } from 'uuid';

// Load environment variables from .env file

dotenv.config();

const app = express();

const port = process.env.PORT || 5101;

export default app;

// MongoDB Atlas connection

const username = process.env.MONGODB\_USERNAME;

const password = process.env.MONGODB\_PASSWORD;

const clusterUrl = process.env.MONGODB\_CLUSTER\_URL;

const dbName = process.env.MONGODB\_DATABASE || 'osoora\_db';

const uri = `mongodb+srv://${username}:${password}@${clusterUrl}/${dbName}?retryWrites=true&w=majority`;

let db;

// Connect to MongoDB Atlas

async function connectToMongoDB() {

  try {

    // Add explicit TLS options to fix SSL connection issues

    const client = new MongoClient(uri, {

      ssl: true,

      tls: true,

      tlsAllowInvalidCertificates: false,

      serverApi: {

        version: '1',

        strict: true,

        deprecationErrors: true

      }

    });

    await client.connect();

    db = client.db(dbName);

    console.log(`Connected to MongoDB Atlas successfully - Database: ${dbName}`);

    return client;

  } catch (error) {

    console.error('MongoDB Atlas connection error:', error);

    throw error;

  }

}

// Enable CORS with specific configuration for custom domain

const allowedOrigins = [

  'http://localhost:5101',

  'https://localhost:5101',

  'http://baktipm.com',

  'https://baktipm.com',

  'http://baktipm.com:5101',

  'https://baktipm.com:5101',

  process.env.ALLOW\_ORIGIN // From .env file

].filter(Boolean); // Remove any undefined/null values

app.use(cors({

  origin: function(origin, callback) {

    // Allow requests with no origin (like mobile apps, curl requests)

    if (!origin) return callback(null, true);

    if (allowedOrigins.indexOf(origin) === -1) {

      console.log(`Allowed origins:`, allowedOrigins);

      console.log(`Request origin:`, origin);

      return callback(null, true); // Allow all origins in production for now

    }

    return callback(null, true);

  },

  credentials: true,

  methods: ['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS'],

  allowedHeaders: ['Content-Type', 'Authorization']

}));

app.use(express.json());

app.use(express.static('.'));

// Log incoming requests for debugging

app.use((req, res, next) => {

  console.log(`${new Date().toISOString()} - ${req.method} ${req.originalUrl}`);

  next();

});

// Middleware to verify user token (simple version for demo)

const authenticateToken = async (req, res, next) => {

  const token = req.headers['authorization']?.split(' ')[1];

  if (!token) {

    return res.status(401).json({ error: 'Unauthorized: Missing token' });

  }

  try {

    // Check if database is initialized

    if (!db) {

      console.error('Database connection not established yet');

      return res.status(503).json({ error: 'Service unavailable: Database not connected' });

    }

    // Find user with this token

    const user = await db.collection('users').findOne({ token });

    if (!user) {

      return res.status(403).json({ error: 'Forbidden: Invalid token' });

    }

    // Add user to request object

    req.user = {

      id: user.user\_id,

      email: user.email,

      username: user.username

    };

    next();

  } catch (error) {

    console.error('Authentication error:', error);

    return res.status(500).json({ error: 'Internal server error during authentication' });

  }

};

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

// AUTH ENDPOINTS

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

// Create a conversation (for test automation)

app.post('/api/conversations', authenticateToken, async (req, res) => {

  try {

    const userId = req.user.id;

    const { title } = req.body;

    const conversation = {

      conversation\_id: uuidv4(),

      title: title || 'Untitled Conversation',

      user\_id: userId,

      created\_at: new Date(),

      messages: []

    };

    await db.collection('conversations').insertOne(conversation);

    res.status(200).json({ conversation\_id: conversation.conversation\_id, ...conversation });

  } catch (error) {

    res.status(500).json({ error: 'Failed to create conversation', details: error.message });

  }

});

// Register new user

app.post('/api/auth/register', async (req, res) => {

  try {

    const { username, email, password } = req.body;

    // Validate input

    if (!username || !email || !password) {

      return res.status(400).json({ error: 'Missing required fields' });

    }

    // Check if user already exists

    const existingUser = await db.collection('users').findOne({

      $or: [{ email }, { username }]

    });

    if (existingUser) {

      return res.status(409).json({ error: 'User with this email or username already exists' });

    }

    // Hash password

    const salt = await bcrypt.genSalt(10);

    const passwordHash = await bcrypt.hash(password, salt);

    // Create user document

    const userId = uuidv4();

    const token = uuidv4(); // Simple token for demo

    const newUser = {

      user\_id: userId,

      username,

      email,

      password\_hash: passwordHash,

      token,

      created\_at: new Date(),

      last\_login: new Date()

    };

    // Insert user

    await db.collection('users').insertOne(newUser);

    // Create default user preferences

    const preferenceId = uuidv4();

    const userPreferences = {

      preference\_id: preferenceId,

      user\_id: userId,

      theme: 'system',

      last\_active\_conversation: null,

      created\_at: new Date(),

      updated\_at: new Date()

    };

    await db.collection('user\_preferences').insertOne(userPreferences);

    // Return success with token

    return res.status(201).json({

      message: 'User registered successfully',

      token,

      user: {

        username,

        email,

        user\_id: userId

      }

    });

  } catch (error) {

    console.error('Registration error:', error);

    return res.status(500).json({ error: 'Internal server error during registration' });

  }

});

// Login user

app.post('/api/auth/login', async (req, res) => {

  try {

    const { email, password } = req.body;

    // Validate input

    if (!email || !password) {

      return res.status(400).json({ error: 'Missing email or password' });

    }

    // Find user

    const user = await db.collection('users').findOne({ email });

    if (!user) {

      return res.status(401).json({ error: 'Invalid credentials' });

    }

    // Verify password

    const validPassword = await bcrypt.compare(password, user.password\_hash);

    if (!validPassword) {

      return res.status(401).json({ error: 'Invalid credentials' });

    }

    // Generate new token

    const token = uuidv4();

    // Update user's token and last login

    await db.collection('users').updateOne(

      { \_id: user.\_id },

      {

        $set: {

          token,

          last\_login: new Date()

        }

      }

    );

    // Get user preferences

    const preferences = await db.collection('user\_preferences').findOne({ user\_id: user.user\_id });

    // Return success with token and user data

    return res.status(200).json({

      message: 'Login successful',

      token,

      user: {

        user\_id: user.user\_id,

        username: user.username,

        email: user.email,

        preferences: preferences || { theme: 'system' }

      }

    });

  } catch (error) {

    console.error('Login error:', error);

    return res.status(500).json({ error: 'Internal server error during login' });

  }

});

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

// USER PREFERENCES ENDPOINTS

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

app.get('/api/user/preferences', authenticateToken, async (req, res) => {

  try {

    const userId = req.user.id;

    // Get user preferences

    const preferences = await db.collection('user\_preferences').findOne({ user\_id: userId });

    if (!preferences) {

      return res.status(404).json({ error: 'User preferences not found' });

    }

    return res.status(200).json(preferences);

  } catch (error) {

    console.error('Error getting user preferences:', error);

    return res.status(500).json({ error: 'Internal server error' });

  }

});

app.put('/api/user/preferences', authenticateToken, async (req, res) => {

  try {

    const userId = req.user.id;

    const { theme, last\_active\_conversation } = req.body;

    // Update fields that are provided

    const updateFields = {};

    if (theme !== undefined) updateFields.theme = theme;

    if (last\_active\_conversation !== undefined) updateFields.last\_active\_conversation = last\_active\_conversation;

    updateFields.updated\_at = new Date();

    // Find and update preferences

    const result = await db.collection('user\_preferences').updateOne(

      { user\_id: userId },

      { $set: updateFields }

    );

    if (result.matchedCount === 0) {

      // Create preferences if they don't exist

      const preferenceId = uuidv4();

      const newPreferences = {

        preference\_id: preferenceId,

        user\_id: userId,

        theme: theme || 'system',

        last\_active\_conversation: last\_active\_conversation || null,

        created\_at: new Date(),

        updated\_at: new Date()

      };

      await db.collection('user\_preferences').insertOne(newPreferences);

      return res.status(201).json(newPreferences);

    }

    // Get updated preferences

    const updatedPreferences = await db.collection('user\_preferences').findOne({ user\_id: userId });

    return res.status(200).json(updatedPreferences);

  } catch (error) {

    console.error('Error updating user preferences:', error);

    return res.status(500).json({ error: 'Internal server error' });

  }

});

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

// CONVERSATION ENDPOINTS

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

app.get('/api/conversations', authenticateToken, async (req, res) => {

  try {

    const userId = req.user.id;

    // Get user conversations

    const conversations = await db.collection('conversations')

      .find({ user\_id: userId })

      .sort({ updated\_at: -1 })

      .toArray();

    return res.status(200).json(conversations);

  } catch (error) {

    console.error('Error getting conversations:', error);

    return res.status(500).json({ error: 'Internal server error' });

  }

});

app.post('/api/conversations', authenticateToken, async (req, res) => {

  try {

    const userId = req.user.id;

    const { topic, preview } = req.body;

    // Create conversation

    const conversationId = uuidv4();

    const newConversation = {

      conversation\_id: conversationId,

      user\_id: userId,

      topic: topic || 'New Conversation',

      preview: preview || '',

      created\_at: new Date(),

      updated\_at: new Date()

    };

    await db.collection('conversations').insertOne(newConversation);

    // Update user preferences with last active conversation

    await db.collection('user\_preferences').updateOne(

      { user\_id: userId },

      {

        $set: {

          last\_active\_conversation: conversationId,

          updated\_at: new Date()

        }

      },

      { upsert: true }

    );

    return res.status(201).json(newConversation);

  } catch (error) {

    console.error('Error creating conversation:', error);

    return res.status(500).json({ error: 'Internal server error' });

  }

});

app.get('/api/conversations/:conversationId/messages', authenticateToken, async (req, res) => {

  try {

    const userId = req.user.id;

    const { conversationId } = req.params;

    // Check if conversation belongs to user

    const conversation = await db.collection('conversations').findOne({

      conversation\_id: conversationId,

      user\_id: userId

    });

    if (!conversation) {

      return res.status(403).json({ error: 'Access denied to this conversation' });

    }

    // Get messages

    const messages = await db.collection('messages')

      .find({ conversation\_id: conversationId })

      .sort({ timestamp: 1 })

      .toArray();

    return res.status(200).json(messages);

  } catch (error) {

    console.error('Error getting messages:', error);

    return res.status(500).json({ error: 'Internal server error' });

  }

});

app.delete('/api/conversations/clear', authenticateToken, async (req, res) => {

  try {

    const userId = req.user.id;

    console.log(`Deleting all conversations for user: ${userId}`);

    // Delete all conversations that belong to this user

    const deleteResult = await db.collection('conversations').deleteMany({

      user\_id: userId

    });

    console.log('Delete result:', deleteResult);

    // Return the result including how many conversations were deleted

    return res.status(200).json({

      message: 'All conversations deleted successfully',

      deletedCount: deleteResult.deletedCount

    });

  } catch (error) {

    console.error('Error deleting conversations:', error);

    return res.status(500).json({ error: 'Failed to delete conversations' });

  }

});

app.post('/api/conversations/:conversationId/messages', authenticateToken, async (req, res) => {

  try {

    const userId = req.user.id;

    const { conversationId } = req.params;

    const { content, role } = req.body;

    // Check if conversation exists and belongs to user

    const conversation = await db.collection('conversations').findOne({

      conversation\_id: conversationId,

      user\_id: userId

    });

    if (!conversation) {

      return res.status(403).json({ error: 'Access denied to this conversation' });

    }

    // Create message

    const messageId = uuidv4();

    const newMessage = {

      message\_id: messageId,

      conversation\_id: conversationId,

      content,

      timestamp: new Date(),

      role: role || 'user'

    };

    await db.collection('messages').insertOne(newMessage);

    // Update conversation preview and timestamp

    await db.collection('conversations').updateOne(

      { conversation\_id: conversationId },

      {

        $set: {

          preview: content.substring(0, 100) + (content.length > 100 ? '...' : ''),

          updated\_at: new Date()

        }

      }

    );

    return res.status(201).json(newMessage);

  } catch (error) {

    console.error('Error creating message:', error);

    return res.status(500).json({ error: 'Internal server error' });

  }

});

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

// USER PROFILE ENDPOINTS

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

app.put('/api/user/profile', authenticateToken, async (req, res) => {

  try {

    const userId = req.user.id;

    const { username, current\_password, new\_password } = req.body;

    // Get user

    const user = await db.collection('users').findOne({ user\_id: userId });

    if (!user) {

      return res.status(404).json({ error: 'User not found' });

    }

    // Update fields

    const updateFields = {};

    // Update username if provided

    if (username && username !== user.username) {

      // Check if username already exists

      const existingUser = await db.collection('users').findOne({ username });

      if (existingUser && existingUser.user\_id !== userId) {

        return res.status(409).json({ error: 'Username already exists' });

      }

      updateFields.username = username;

    }

    // Update password if provided

    if (current\_password && new\_password) {

      // Verify current password

      const validPassword = await bcrypt.compare(current\_password, user.password\_hash);

      if (!validPassword) {

        return res.status(401).json({ error: 'Current password is incorrect' });

      }

      // Hash new password

      const salt = await bcrypt.genSalt(10);

      updateFields.password\_hash = await bcrypt.hash(new\_password, salt);

    }

    // Only update if there are changes

    if (Object.keys(updateFields).length > 0) {

      await db.collection('users').updateOne(

        { user\_id: userId },

        { $set: updateFields }

      );

    }

    // Get updated user (exclude password hash)

    const updatedUser = await db.collection('users').findOne(

      { user\_id: userId },

      { projection: { password\_hash: 0, token: 0 } }

    );

    return res.status(200).json({

      message: 'Profile updated successfully',

      user: updatedUser

    });

  } catch (error) {

    console.error('Error updating profile:', error);

    return res.status(500).json({ error: 'Internal server error' });

  }

});

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

// CHAT PROXY ENDPOINT

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

app.post('/api/chat', async (req, res) => {

    try {

        let { message, conversationId } = req.body;

        // Set headers for SSE

        res.setHeader('Content-Type', 'text/event-stream');

        res.setHeader('Cache-Control', 'no-cache');

        res.setHeader('Connection', 'keep-alive');

        // If no conversationId is provided, create a new conversation

        if (!conversationId) {

            try {

                const newConversationId = uuidv4();

                const userId = req.headers['user-id'] || 'anonymous'; // Get user ID from headers or use anonymous

                const newConversation = {

                    conversation\_id: newConversationId,

                    user\_id: userId,

                    title: message.substring(0, 30) + '...', // Create a title from the first message

                    created\_at: new Date(),

                    updated\_at: new Date(),

                    messages: [],

                    message: message.substring(0, 100), // Required field based on schema

                    preview: message.substring(0, 100), // For conversation preview

                    dashscope\_session\_id: null // Will be updated later

                };

                await db.collection('conversations').insertOne(newConversation);

                console.log(`Created new conversation with ID: ${newConversationId}`);

                conversationId = newConversationId;

            } catch (dbError) {

                console.error('Error creating new conversation:', dbError);

                // Continue without conversationId if we can't create one

            }

        }

        // Validate the Dashscope URL

        const dashscopeUrl = process.env.DASHSCOPE\_URL;

        if (!dashscopeUrl || !dashscopeUrl.startsWith('http')) {

            throw new Error(`Invalid Dashscope URL: ${dashscopeUrl}`);

        }

        // Validate API key

        const apiKey = process.env.DASHSCOPE\_API\_KEY;

        if (!apiKey) {

            throw new Error('Missing Dashscope API key');

        }

        // console.log('Making request to Dashscope API at:', dashscopeUrl);

        // Format the prompt to request Markdown responses

        const formattedPrompt = `Please format your response using Markdown with proper headings, lists, code blocks, and other formatting where appropriate. Here's the user's message:\n\n${message}`;

        // Debug conversation info

        console.log('Request received with:', {

            hasMessage: Boolean(message),

            conversationId: conversationId || 'none',

            bodyKeys: Object.keys(req.body)

        });

        let existingSessionId = null;

        // Check if we already have a session\_id for this conversation

        if (conversationId) {

            try {

                // Find the conversation in MongoDB

                const conversation = await db.collection('conversations').findOne({ conversation\_id: conversationId });

                if (conversation && conversation.dashscope\_session\_id) {

                    existingSessionId = conversation.dashscope\_session\_id;

                    console.log(`Found existing session ID: ${existingSessionId} for conversation: ${conversationId}`);

                }

            } catch (dbError) {

                console.error('Error retrieving session ID from conversation:', dbError);

            }

        }

        // Prepare request configuration

        const axiosConfig = {

            method: 'post',

            url: dashscopeUrl,

            data: {

                input: {

                    prompt: formattedPrompt,

                    // Include session\_id directly if this is an ongoing conversation

                    ...(existingSessionId && { session\_id: existingSessionId })

                },

                parameters: {},

                // If we have an existing session\_id, use it. Otherwise, use conversationId as fallback

                debug: existingSessionId ? { session\_id: existingSessionId } :

                       conversationId ? { session\_id: conversationId } : {}

            },

            headers: {

                'Authorization': `Bearer ${apiKey}`,

                'Content-Type': 'application/json',

                'X-DashScope-SSE': 'enable',

                'Accept': 'text/event-stream'

            },

            responseType: 'stream',

            timeout: 60000, // 60 second timeout

            validateStatus: status => status < 500 // Don't throw for 4xx errors

        };

        const response = await axios(axiosConfig);

        let buffer = '';

        let lastProcessedText = '';

        let clientAborted = false;

        // Abort controller for upstream requests (if supported by axios)

        const abortController = new AbortController();

        axiosConfig.signal = abortController.signal;

        // Handle client disconnect

        req.on('close', () => {

            clientAborted = true;

            console.log('Client disconnected during text generation. Aborting stream and upstream request.');

            try {

                response.data.destroy && response.data.destroy(); // Abort stream if possible

            } catch (e) {

                console.warn('Error aborting response stream:', e);

            }

            if (abortController) {

                abortController.abort();

            }

        });

        // Pipe the response stream to the client

        response.data.on('data', (chunk) => {

            try {

                buffer += chunk.toString();

                // Process complete events

                let newlineIndex;

                while ((newlineIndex = buffer.indexOf('\n\n')) !== -1) {

                    const event = buffer.slice(0, newlineIndex);

                    buffer = buffer.slice(newlineIndex + 2);

                    // Parse the event

                    const lines = event.split('\n');

                    for (const line of lines) {

                        if (line.startsWith('data:')) {

                            const jsonStr = line.slice(5).trim();

                            if (jsonStr && jsonStr !== '[DONE]') {

                                try {

                                    const jsonData = JSON.parse(jsonStr);

                                    if (jsonData.output?.text) {

                                        const currentText = jsonData.output.text;

                                        // Only send the new words

                                        if (currentText.startsWith(lastProcessedText) && currentText !== lastProcessedText) {

                                            const newText = currentText.slice(lastProcessedText.length);

                                            const formattedData = {

                                                choices: [{

                                                    message: {

                                                        content: newText

                                                    }

                                                }]

                                            };

                                            res.write(`data: ${JSON.stringify(formattedData)}\n\n`);

                                            lastProcessedText = currentText;

                                        } else if (currentText !== lastProcessedText) {

                                            // If there's a mismatch, send the full text

                                            const formattedData = {

                                                choices: [{

                                                    message: {

                                                        content: currentText

                                                    }

                                                }]

                                            };

                                            res.write(`data: ${JSON.stringify(formattedData)}\n\n`);

                                            lastProcessedText = currentText;

                                        }

                                    }

                                    const sessionId = jsonData.output.session\_id;

                                    console.log('DEBUG - Session Data:', {

                                        sessionId,

                                        conversationId,

                                        willUpdate: Boolean(conversationId && sessionId),

                                        timestamp: new Date().toISOString()

                                    });

                                    // Store the session ID in MongoDB if we have a conversationId and sessionId

                                    if (conversationId && sessionId) {

                                        (async () => {

                                            try {

                                                // First check if the conversation exists

                                                console.log(`Checking if conversation ${conversationId} exists in MongoDB...`);

                                                const conversation = await db.collection('conversations').findOne({ conversation\_id: conversationId });

                                                console.log('Conversation found:', Boolean(conversation));

                                                // If conversation doesn't exist, create it first

                                                if (!conversation) {

                                                    console.log(`Conversation with ID ${conversationId} doesn't exist in MongoDB. Creating it first...`);

                                                    const userId = req.headers['user-id'] || 'anonymous';

                                                    // Use only the fields that the MongoDB schema expects using snake\_case consistently

                                                    const newConversation = {

                                                        conversation\_id: conversationId,

                                                        user\_id: userId,

                                                        title: 'Chat session',

                                                        created\_at: new Date(),

                                                        updated\_at: new Date(),

                                                        messages: [],

                                                        message: 'Chat session started', // Required field based on error

                                                        dashscope\_session\_id: null // Will be updated later

                                                    };

                                                    await db.collection('conversations').insertOne(newConversation);

                                                    console.log(`Created new conversation with ID: ${conversationId}`);

                                                }

                                                // Debug information about message and lastProcessedText

                                                console.log('DEBUG - Before MongoDB update:');

                                                console.log('  message type:', typeof message);

                                                console.log('  message value:', message);

                                                console.log('  lastProcessedText type:', typeof lastProcessedText);

                                                console.log('  lastProcessedText value:', lastProcessedText);

                                                // Now update with session ID and message content - using snake\_case field names consistently

                                                const updateResult = await db.collection('conversations').updateOne(

                                                    { conversation\_id: conversationId },

                                                    {

                                                        $set: {

                                                        dashscope\_session\_id: sessionId,

                                                        updated\_at: new Date(),

                                                        messages: [

                                                            {

                                                                role: 'user',

                                                                content: typeof message === 'object' ? JSON.stringify(message) : message,

                                                                timestamp: new Date()

                                                            },

                                                            {

                                                                role: 'assistant',

                                                                content: typeof lastProcessedText === 'object' ? JSON.stringify(lastProcessedText) : lastProcessedText,

                                                                timestamp: new Date()

                                                            }

                                                        ]

                                                    }

                                                }

                                            );

                                        } catch (dbError) {

                                            console.error('Error updating conversation with session ID:', dbError);

                                        }

                                    })();

                                }

                                    // Store the message in MongoDB

                                } catch (e) {

                                    console.error('Error parsing JSON:', e);

                                }

                            }

                        }

                    }

                }

              } catch (e) {

                console.error('Error processing chunk:', e);

              }

            });

            response.data.on('end', async () => {

    res.write('data: [DONE]\n\n');

    res.end();

    // After streaming is complete, store the user prompt and assistant reply in MongoDB

    try {

        if (conversationId && lastProcessedText) {

            // Find the conversation, create if not exists

            let conversation = await db.collection('conversations').findOne({ conversation\_id: conversationId });

            if (!conversation) {

                const userId = req.headers['user-id'] || 'anonymous';

                const newConversation = {

                    conversation\_id: conversationId,

                    user\_id: userId,

                    title: 'Chat session',

                    created\_at: new Date(),

                    updated\_at: new Date(),

                    messages: [],

                    message: 'Chat session started',

                    dashscope\_session\_id: null

                };

                await db.collection('conversations').insertOne(newConversation);

            }

            // Push the user and assistant messages as a pair

            await db.collection('conversations').updateOne(

                { conversation\_id: conversationId },

                {

                    $push: {

                        messages: {

                            $each: [

                                {

                                    role: 'user',

                                    content: typeof message === 'object' ? JSON.stringify(message) : message,

                                    timestamp: new Date()

                                },

                                {

                                    role: 'assistant',

                                    content: typeof lastProcessedText === 'object' ? JSON.stringify(lastProcessedText) : lastProcessedText,

                                    timestamp: new Date()

                                }

                            ]

                        }

                    },

                    $set: {

                        updated\_at: new Date(),

                        dashscope\_session\_id: existingSessionId || null

                    }

                }

            );

        }

    } catch (dbError) {

        console.error('Error saving messages after SSE end:', dbError);

    }

});

        response.data.on('error', (error) => {

            console.error('Stream Error:', error);

            res.write(`data: ${JSON.stringify({ error: error.message })}\n\n`);

            res.end();

        });

    } catch (error) {

        // Provide detailed error logging

        if (error.code === 'ENOTFOUND' || error.code === 'ECONNREFUSED' || error.message.includes('Invalid URL')) {

            console.error(`Proxy Error: ${error.message}`);

            console.error('Check your DASHSCOPE\_URL environment variable:', process.env.DASHSCOPE\_URL);

        } else if (error.response) {

            console.error(`Dashscope API Error (${error.response.status}):`, error.message);

            console.error('Response data:', error.response.data);

        } else {

            console.error('Unexpected error in proxy endpoint:', error.message);

            console.error('Error details:', error);

        }

        // Keep the connection alive if possible

        try {

            // Format the error as a choice message to match expected client format

            const errorData = {

                error: true,

                message: `Error: ${error.message}`,

                details: error.response?.data || error.stack

            };

            const formattedError = {

                choices: [{

                    message: {

                        content: JSON.stringify(errorData)

                    }

                }]

            };

            // Write the SSE format with proper newlines

            res.write(`data: ${JSON.stringify(formattedError)}\n\n`);

            res.write('data: [DONE]\n\n');

            res.end();

        } catch (e) {

            // Fallback to regular error response if SSE fails

            res.status(500).json({

                error: 'Error communicating with Dashscope API',

                message: error.message,

                details: error.response?.data || error.stack

            });

        }

    }

});

// Start the server with MongoDB connection

async function startServer() {

    let mongoClient;

    try {

        // Validate Dashscope URL format before starting

        const url = process.env.DASHSCOPE\_URL;

        if (!url) {

            console.error('ERROR: DASHSCOPE\_URL is not set in environment variables');

            process.exit(1);

        }

        try {

            // Test if it's a valid URL by creating a URL object

            new URL(url);

        } catch (error) {

            console.error('ERROR: Invalid DASHSCOPE\_URL in environment variables:', url);

            console.error('Server startup aborted due to invalid Dashscope URL');

            process.exit(1);

        }

        mongoClient = await connectToMongoDB();

        const server = app.listen(port, () => {

            console.log(`Server running at http://localhost:${port}`);

        });

        // Handle graceful shutdown

        process.on('SIGINT', async () => {

            server.close();

            if (mongoClient) {

                await mongoClient.close();

            }

            process.exit(0);

        });

    } catch (error) {

        console.error('Failed to start server:', error);

        if (mongoClient) await mongoClient.close();

        process.exit(1);

    }

}

if (process.env.NODE\_ENV !== 'test') {

  startServer();

}