Implementation Plan and Role Division: Terms Demystifier

Project Team

October 22, 2025

Abstract

The Universal Terms Demystifier is a browser extension designed to simplify complex legal documents (Terms of Service, Privacy Policies) using Generative AI. This document outlines the technical implementation plan, focusing exclusively on the AI backend, data management, and client-side logic, along with a three-role split to ensure clear separation of responsibilities and efficient development.

---

1. Three-Role Split (Excluding Frontend/Design)

The project's complexity requires distinct technical roles focused on the core innovation (AI) and the infrastructure needed to support it.

1. AI/ML Engineer (Prompt Engineering & Core Logic)

Focus: Crafting and optimizing the prompts for the `gemini-1.5-flash-latest` model.

Key Tasks: Developing system instructions to achieve accurate extraction, risk assessment (**HIGH, MEDIUM, LOW, NEUTRAL**), categorization (Privacy, Fees, Cancellation), and simple language explanation.

Responsibilities: Defining the JSON output schema for the AI response; implementing long-text chunking strategies if needed; continuous prompt refinement based on feedback.

2. Backend/API Engineer (Server, Database & Local Setup)

Focus: Building the secure Node.js/Express API and managing data persistence with MongoDB.

Key Tasks: Developing the Express web service; securing the Gemini API key using environment variables; setting up the **MongoDB** database for storing user analyses.

Responsibilities: Implementing CORS middleware for extension communication; managing data queries.

3. Extension Developer (Client-Side Logic & Integration)

Focus: The browser-specific logic, text extraction, and communication layers.

Key Tasks: Implementing the DOM parsing logic to reliably identify and extract legal text from any webpage; handling the `fetch` requests to the Backend API (including authentication tokens).

Responsibilities: Managing the flow between the Content Script and the Background Script; receiving the JSON results from the backend and preparing them for display (e.g., summary popup, dashboard cards); handling user authentication via **Auth0** within the extension using `chrome.identity`.

---

2. Implementation Plan

The implementation is divided into four stages, emphasizing a Minimum Viable Product (MVP) in Phase 2.

Phase 1: Foundation and Proof of Concept (POC)

- **AI PoC:** The AI/ML Engineer crafts initial prompts for basic clause extraction and simplification, mandating a structured JSON output.
- **Backend Setup:** The Backend Engineer builds the basic `/api/analyze` Express endpoint, successfully receiving raw text and making a successful, authenticated call to the Gemini API using the correct model.
- **Extension Shell:** The Extension Developer creates the basic browser extension manifest and the mechanism to extract text and send a request to the local backend.

Phase 2: Core Functionality (MVP)

- **AI Refinement:** Prompts are refined to include three key components: Extraction of original snippet, Simple Explanation, and a Risk Level (**HIGH, MEDIUM, LOW, NEUTRAL**). Implement long-text chunking if needed.
- **Backend Integration:** The Backend Engineer integrates the finalized AI logic, enforces the JSON schema check, and ensures secure API key management via `.env` files. CORS is correctly configured for local development. **MongoDB** connection and saving logic is implemented. `/api/dashboard` endpoint is created.
- **Client-Side Core:** The Extension Developer implements the full communication loop (extract -> get **Auth0** token -> API call -> receive result). The content script uses the received data to display the summary popup. The frontend dashboard fetches and displays analysis cards.

Phase 3: Data Persistence and Advanced Features

- **Data Model Refinement:** The Backend Engineer refines the **MongoDB** model as needed (e.g., adding timestamps, ensuring `userId` indexing). Implement logic to *update* existing analyses instead of creating duplicates (`findOneAndUpdate`).

- **User Management:** Secure user authentication is implemented using **Auth0** on both the frontend website and within the extension background script (`chrome.identity`). Backend endpoints are secured using Auth0 JWT validation middleware.
- **Extension Features:** The Extension Developer implements token storage/refresh logic (`chrome.storage.local`) and potentially features like Export/Share.

Phase 4: Scaling, Monitoring, and QA

- **Monitoring & Logging:** The Backend Engineer sets up better logging within the Node.js application to track API calls, response times, and errors more effectively during local testing.
- **QA/Testing:** Systematic testing is conducted across various common legal documents and websites to identify edge cases in text extraction and prompt failures. Test authentication flows thoroughly.
- **Feedback Loop:** The Extension Developer could build an in-app user feedback mechanism, and the AI/ML Engineer uses this data for continuous prompt and risk assessment calibration.