Bilkent University
Department of Computer Science

CS 492 - Senior Design Project II
Detailed Design Report

Agreemind (T2516)

Group Members:
Ata Oğuz - 22202453
Ata Soykal - 22202290
Can Polat Bülbül - 22203369
Edip Emre Dönger - 22201531
Emir Görgülü - 22202834

Supervisor: Hamdi Dibeklioğlu
Instructors: Mert Bıçakçı, İlker Burak Kurt

1. Introduction

1.1. Purpose of the System

People regularly enter into agreements of all kinds: rental contracts, subscription services, insurance policies, warranty policies, loan documents, employment terms, freelance contracts, and many others. These documents form the basis of how individuals interact with companies, service providers, landlords, financial institutions, and digital platforms. Although these agreements shape important aspects of people's financial, personal, and professional lives, they are often difficult to understand. The language is typically dense and technical, key details are buried deep within long clauses, and the implications of certain terms are not always obvious to a non-expert reader. As a result, individuals may unknowingly accept conditions that impose restrictive obligations, and they may overlook important deadlines or rights.

The challenge does not end once a contract is signed. Many agreements are revised over time, sometimes without proper notification or clear explanations of how the changes affect the user. Companies frequently update Terms of Service or privacy policies, and users have no practical way to see what has been added or removed. Managing agreements becomes even more difficult when people juggle multiple subscriptions, insurance plans, and recurring services across different platforms. Users struggle not only to interpret the agreements they encounter but also to keep track of them and maintain awareness of their rights and responsibilities throughout the lifecycle of the contract.

Agreemind is designed to address these challenges by serving as an intelligent companion for understanding, tracking, and acting upon personal agreements. It analyzes documents in natural language, translates complex clauses into clear explanations, highlights terms that may be risky or unusually strict, and identifies obligations or deadlines the user must keep in mind. It stores processed agreements in a secure personal vault, making it possible to revisit them, search across them, or compare different versions or drafts. When companies update their terms, Agreemind identifies and explains the changes, ensuring that users stay informed.

Beyond interpretation, the system also helps users act on their rights by identifying possible actions and generating drafts or templates tailored to the relevant clause. By centralizing

documents, offering meaningful analysis, and providing practical tools, Agreemind enables individuals to make informed decisions, avoid unfavorable conditions, and maintain control over their responsibilities and protections throughout the life of an agreement.

1.2. Design Goals

1.2.1. Usability

- The user interface shall present summaries, clause flags, and risks in clear and readable formats understandable by non-experts.
- Color-coded indicators for risk levels shall follow accessibility guidelines.
- The UI for the mobile app should be intuitive and easy to use.

1.2.2. Reliability

- The system shall maintain high availability.
- The vault and document records shall not be lost due to server errors; periodic backups must be maintained.
- Notification services shall reliably trigger reminders before deadlines.

1.2.3. Portability

- The system shall run on major modern browsers and mobile operating systems.
- The backend shall be deployable on major cloud platforms without major modification.
- The browser extension shall support Chromium-based browsers and Firefox, subject to platform API limits.

1.2.4. Privacy

- Users shall retain full ownership of uploaded contracts and analysis results.
- No contract text or user-generated data shall be used for model training or external sharing without explicit opt-in.
- The system shall provide mechanisms for deleting individual documents from the vault and deleting the entire user account and all associated data.
- The system shall comply with applicable data protection laws.

1.2.5. Scalability

- The analysis pipeline shall scale horizontally to handle multiple simultaneous document uploads.
- The system shall support growth in the number of users and stored documents without significant performance degradation.
- The vector store and search mechanisms shall support large embedding collections efficiently.

1.2.6. Maintainability

- The system shall use a modular architecture so that core analysis components (clause classification, risk detection, obligation extraction) are independent from domain-specific logic.
- New domains, rule sets, or knowledge bases shall be addable as separate, self-contained modules without modifying core code.
- Regulatory or industry-specific rules shall be stored in external, versioned configuration files so they can be updated or expanded easily.
- The system shall expose clear internal interfaces that define how domain modules interact with the core engine, enabling low coupling and simple future extension.
- Updating or replacing domain modules, rules, or knowledge bases shall not require system downtime.
- Code shall be consistently structured and documented to support long-term maintainability.

1.2.7. Legal and Ethics

- The system shall clearly communicate that it does not provide legal advice and should be used as an informational tool.
- Explanations, warnings, and summaries shall be generated in a neutral, non-directive tone.
- The system shall ensure transparency regarding data usage, automated decision-making, and third-party integrations.

1.2.8. Modularity

- The system architecture shall separate the frontend, backend services, and data storage layers to allow independent development and testing.
- The document processing pipeline shall be divided into modular components such as document ingestion, clause segmentation, summarization, risk detection, and deadline extraction.
- Each analysis capability (e.g., clause classification, risk detection, obligation extraction) shall operate as an independent module.
- The Personal Vault, notification system, document comparison service, and rights-enforcement service shall be implemented as separate backend components.
- Individual modules shall be replaceable or upgradable without requiring changes to the rest of the system.

1.2.9. Performance

- The system shall process typical agreement analyses within a reasonable time to maintain a responsive user experience.
- Backend API response times for common actions (e.g., retrieving reports, searching the vault) shall be minimized to ensure smooth interaction.
- Document uploads and preprocessing shall be optimized to prevent excessive waiting times for users.
- Long-running analysis tasks shall be executed asynchronously to prevent blocking the user interface.
- The system shall support concurrent processing of multiple document analyses without significant performance degradation.

1.2.10. Flexibility

- The system shall support multiple document input formats such as PDF, DOCX, raw text, and HTML content.
- Users shall be able to submit agreements through multiple entry points including the web interface, mobile application, and browser extension.

- The platform shall support analysis of different agreement types such as Terms of Service, privacy policies, and consumer contracts.
- The user interface shall allow different workflows such as one-time analysis or long-term storage in the Personal Vault.
- The system architecture shall support deployment on different cloud platforms.

1.2.11. Extendibility

- The system architecture shall allow new AI models or analysis modules to be added without affecting existing components.
- Additional legal rule sets or regulatory knowledge bases shall be easily integrated into the analysis pipeline.
- New features such as additional comparison tools, notification mechanisms, or document analysis capabilities shall be added as separate modules.
- Backend services shall expose well-defined interfaces to allow integration of new components.
- The modular architecture shall support future expansion of the platform's functionality.

1.2.12. Marketability

- Agreemind shall target individual users who need assistance understanding online agreements and legal documents.
- The system shall provide a unified platform combining document analysis, risk detection, deadline tracking, and contract management.
- Multi-platform availability (web, mobile, and browser extension) shall increase accessibility and usability.
- The Personal Vault feature shall provide long-term value by allowing users to manage agreements throughout their lifecycle.
- The platform shall differentiate itself from enterprise legal tools by focusing on simplicity and accessibility for everyday users.

1.3. Definitions, Acronyms, Abbreviations

Definitions of Key Terms

● **Agreement Analysis:** The automated process of analyzing legal documents to extract summaries, risks, obligations, and deadlines.

● **Clause Segmentation:** The process of dividing a legal document into logical sections or clauses for structured analysis.

● **Personal Vault:** A secure storage system where users can save and manage previously analyzed agreements.

● **Risk Detection:** Identification of potentially harmful or restrictive contract clauses such as forced arbitration or automatic renewal.

● **Obligation Extraction:** The process of identifying actions that the user must perform according to the agreement.

● **Deadline Extraction:** Detection of time-sensitive events such as cancellation deadlines or renewal periods within contracts.

● **Version Tracking:** Monitoring changes between different versions of the same agreement over time.

● **Rights Enforcer:** A feature that identifies actionable rights within agreements and generates templates to exercise those rights.

● **Agreement Comparison:** The process of identifying differences between two agreements or versions of the same contract.

Acronyms and Abbreviations

● **AI (Artificial Intelligence):** Technology that enables machines to perform tasks that normally require human intelligence.

● **API (Application Programming Interface):** A set of rules that allows software components to communicate with each other.

● **DB (Database):** A structured system used for storing and managing application data.

● **LLM (Large Language Model):** A machine learning model capable of understanding and generating natural language text.

● **NLP (Natural Language Processing):** A field of AI focused on enabling computers to understand and process human language.

● **OCR (Optical Character Recognition):** Technology used to convert images or scanned documents into machine-readable text.

● **UI (User Interface):** The visual interface through which users interact with the system.

● **UX (User Experience):** The overall experience of users when interacting with the system.

● **RAG (Retrieval-Augmented Generation):** A system architecture combining information retrieval with generative AI models.

1.4. Overview

Agreemind is an AI-powered platform designed to help users understand and manage legal agreements such as Terms of Service, privacy policies, and consumer contracts. Many individuals accept agreements without fully understanding the legal implications due to the complexity and length of legal language. Agreemind addresses this challenge by automatically analyzing agreements and presenting key insights in clear and accessible language.

The system enables users to upload or share agreements through multiple entry points, including a web interface, mobile application, and browser extension. After ingestion, documents are processed through an analysis pipeline that performs clause segmentation, summarization, risk detection, and extraction of obligations and deadlines. The results are presented in an interactive report that highlights critical clauses and explains their potential impact.

Users may store agreements in a secure Personal Vault, allowing them to track deadlines, compare different versions of contracts, and query stored agreements through natural language search. Additional features such as reminders and the Rights Enforcer help users actively manage agreements rather than passively accepting them.

The architecture of Agreemind is designed to be modular and scalable, enabling the integration of new analysis modules, legal knowledge bases, and additional features in the future.

2. Current Software Architecture

The current legal technology market is predominantly focused on enterprise solutions, serving law firms and large corporations with complex contract lifecycle management tools. While these tools are powerful, they leave a significant gap in the consumer market. Individuals are frequently exposed to complex agreements, ranging from software Terms of Service to rental leases and freelance contracts, but lack accessible tools to understand or manage them.

Agreemind addresses this gap by positioning itself as a "Personal Legal Companion" rather than a corporate tool. Unlike existing solutions that focus on drafting or B2B compliance, Agreemind focuses on the reception of contracts, empowering the individual signer. Below is an analysis of similar tools and how Agreemind distinguishes itself:

● **Enterprise Legal AI (Ironclad, Kira Systems, Luminance):** These platforms utilize advanced machine learning for clause detection and contract review. They are designed for corporate legal teams to streamline workflows and ensure compliance [1], [2], [3].

Differentiation: These tools are cost-prohibitive and overly complex for individual users. Agreemind is built for the "non-expert reader," prioritizing the translation of "legalese" into plain language and offering a user-centric interface rather than a corporate dashboard.

● **Crowdsourced Transparency Projects (ToS;DR, Open Terms Archive):** These initiatives rely on community contributions to grade and summarize the Terms of Service for popular websites [4], [5].

Differentiation: These platforms suffer from limited coverage and cannot handle arbitrary personal documents (e.g., a specific landlord's lease or a freelance NDA). Agreemind utilizes AI to analyze any document uploaded by the user, providing immediate, personalized analysis rather than relying on a pre-existing database.

● **Real-time ToS Detectors (Termzy AI):** These tools often exist as browser extensions that analyze terms while a user browses the web [6].

Differentiation: While helpful for web browsing, these tools lack lifecycle management. Agreemind differentiates itself through its "Personal Vault" and "Rights Enforcer," which allow users to store agreements, track changes over time, and actively generate legal request drafts long after the document is signed.

In summary, while the market contains high-end corporate tools and basic web-summarizers, Agreemind stands out by combining the analytical power of enterprise AI with the accessibility required for consumer use. It shifts the user's role from passive acceptance to active management.

3. Proposed Software Architecture

3.1. Overview

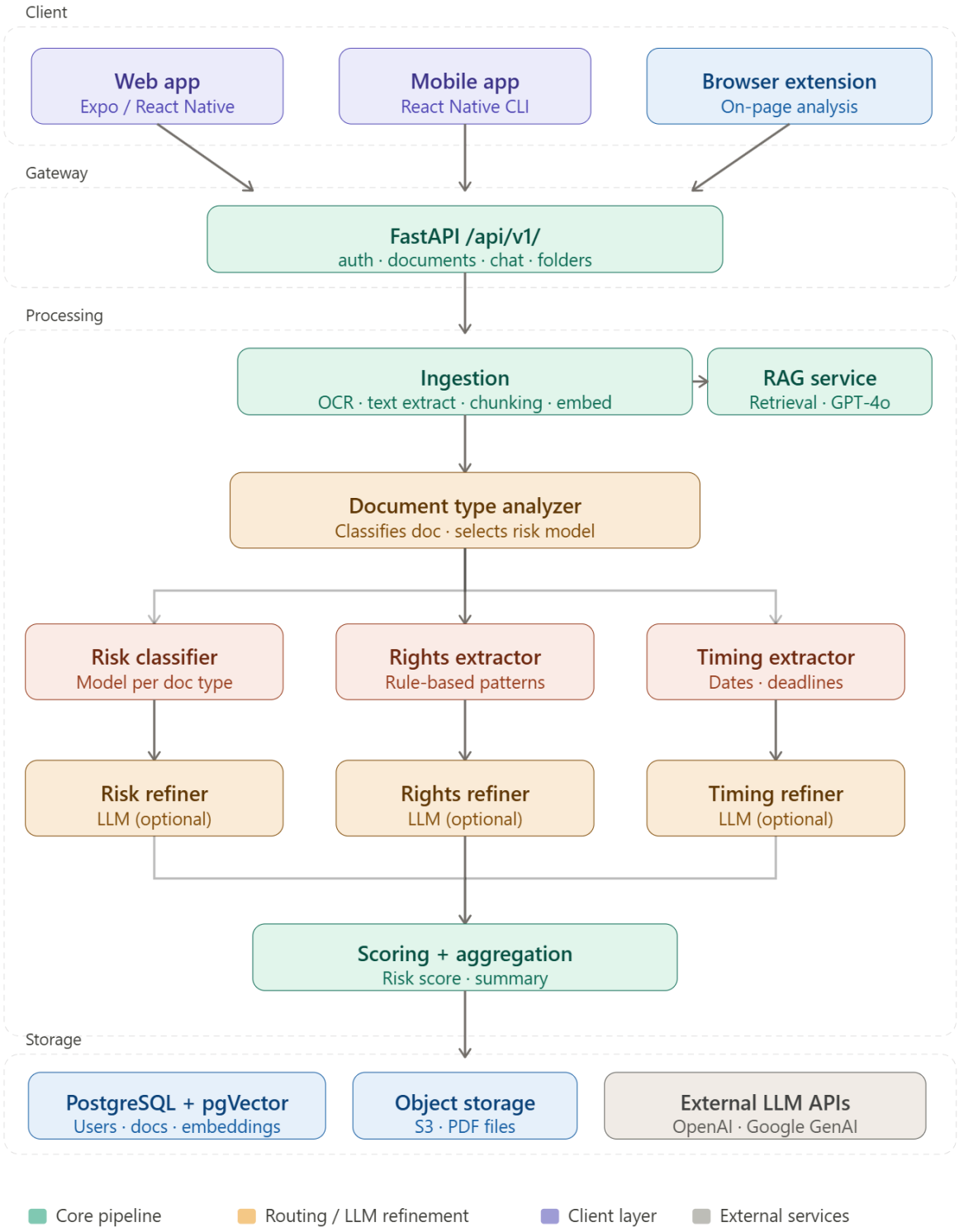
Agreemind is built on a layered, modular architecture that separates client presentation, API routing, document processing, and data persistence into distinct concerns. The system is designed to accept legal documents from multiple client surfaces, process them through an AI-powered analysis pipeline, and present the results in an interactive and queryable interface.

At the top of the stack, two client applications, a web application and a mobile application, share a single backend API. The backend is a FastAPI application that acts as the central, handling authentication, routing requests, triggering document ingestion, and managing all communication with the AI processing pipeline and the database. The core processing layer is where Agreemind's distinguishing functionality lives: a two-stage contract analysis pipeline that first applies deterministic NLP engines to extract risk signals, user rights, and timing constraints from clause-level text, then optionally refines those findings using a large language model. Documents are also chunked and embedded at ingestion time, enabling a retrieval-augmented generation (RAG) chat interface that allows users to ask free-form questions across all their stored agreements.

The entire stack is containerized using Docker, which orchestrates four services: the PostgreSQL database, the FastAPI backend, the Expo web frontend, and a pgAdmin management interface, under a unified local environment.

3.2. Subsystem decomposition

The system is organized into four layers: the Client Layer, the API Gateway Layer, the Core Processing Layer, and the Storage Layer. Figure 1 illustrates the subsystem decomposition and data flow between these layers.


Client
Web app
Expo / React Native
Mobile app
React Native CLI
Browser extension
On-page analysis
Gateway
FastAPI /api/v1/
auth · documents · chat · folders
Processing
Ingestion
OCR · text extract · chunking · embed
RAG service
Retrieval · GPT-4o
Document type analyzer
Classifies doc · selects risk model
Risk classifier
Model per doc type
Rights extractor
Rule-based patterns
Timing extractor
Dates · deadlines
Risk refiner
LLM (optional)
Rights refiner
LLM (optional)
Timing refiner
LLM (optional)
Scoring + aggregation
Risk score · summary
Storage
PostgreSQL + pgVector
Users · docs · embeddings
Object storage
S3 · PDF files
External LLM APIs
OpenAI · Google GenAI
Core pipeline
Routing / LLM refinement
Client layer
External services

3.2.1. Client Layer

The client layer consists of two applications that expose the full feature set of Agreemind to end users.

The web application is built with React Native and Expo Router, running in the browser via react-native-web. It is structured around a file-based navigation system with ten main screens: homepage, login, dashboard, documents, document history, document chat, compare, vault, alerts, and rights. The documents screen is the most complex, managing file upload, analysis result display, version history, and in-browser PDF viewing.

The mobile application is a standalone React Native CLI app targeting Android and iOS. It mirrors the web application's feature set using a tab-based navigation structure with five tabs Home, Documents, Vault, Alerts, and More where the More section houses the Compare, Rights, and Settings screens.

Both clients communicate with the backend exclusively through HTTP requests, attaching JWT bearer tokens to every protected request. Tokens are stored using expo-secure-store on native platforms and localStorage on the web through a unified storage interface.

Data flow: Client → Backend: authentication, document upload, analysis retrieval, chat queries, folder management. Backend → Client: JWT tokens, document metadata, analysis results, chat responses.

3.2.2. API Gateway Layer

All client requests pass through a single FastAPI application running on port 8000. It serves as the unified entry point for the system and mounts four versioned routers under the /api/v1/ prefix:

- /auth — user registration, login, and profile management
- /documents — document upload, ingestion, analysis retrieval, versioning, and comparison
- /chat — RAG-based natural language querying over stored documents

- /folders — hierarchical folder creation and management

Every protected endpoint uses a shared authentication dependency that decodes the incoming JWT and resolves the corresponding user record before any business logic executes. This layer also applies CORS middleware and request logging.

Data flow: Client → Gateway: all API requests. Gateway → Processing Layer: triggers document analysis and embedding. Gateway → Storage Layer: reads and writes structured data.

3.2.3. Core Processing Layer

This layer contains the system's primary AI functionality and is divided into two pipelines.

The document analysis pipeline is orchestrated by the ContractAnalysisService. When a document is uploaded, text is extracted using PyMuPDF and pytesseract (for OCR on scanned files). The extracted text is passed to the ContractSegmenter, which breaks it into anchors (logical sections) and atoms (individual clauses). Three deterministic engines then run in parallel over every atom: the RiskClassifier, which applies a fine-tuned multi-label transformer model to detect risky clause categories; the RightsExtractor, which uses rule-based and dependency-parse patterns to identify user obligations and opt-out rights; and the TimingExtractor, which detects dates, deadlines, and renewal windows. If LLM refinement is enabled, a second stage passes the flagged candidates to RiskRefiner, RightsRefiner, and TimingRefiner, which process them in batches through OpenAI or Google GenAI models. A final scoring step aggregates the refined findings into an overall risk level and a per-category breakdown.

The RAG pipeline runs in parallel with analysis at ingestion time. The document text is split into overlapping chunks (1000-character chunks with 200-character overlap), and each chunk is embedded using OpenAI's text-embedding-3-small model (1536 dimensions) and stored in PostgreSQL via the pgVector extension. At query time, the user's question is embedded with the same model, and a hybrid retrieval step combines vector similarity search with full-text search using Reciprocal Rank Fusion. The top results are expanded with neighboring chunks and passed to GPT-4o to generate a cited answer.

Data flow: Gateway → Processing Layer: raw file and user context. Processing Layer → Storage Layer: analysis results, document chunks, and embeddings. Processing Layer → External APIs: embedding and LLM refinement requests to OpenAI and Google GenAI.

3.2.4. Storage Layer

The storage layer holds all persistent system data across two mechanisms. PostgreSQL stores structured records for users, documents, analysis results, document chunks with their vector embeddings, and folders. PDF files are stored in a cloud object storage bucket (AWS S3 or compatible), with each file addressed by a UUID-prefixed key. The Document.file_path field stores the object key rather than a local path, and pre-signed URLs are generated on demand for serving PDFs to the client. On deletion, the object is removed from the bucket alongside the database records and vector chunks.

Data flow: Processing Layer → PostgreSQL: writes analysis results, chunk embeddings, and document metadata. Gateway → PostgreSQL: reads user, document, folder, and analysis data in response to client requests. Gateway → Filesystem: reads uploaded files for PDF serving.

3.3. Persistent data management

Agreemind must store and efficiently retrieve several categories of data per user: document metadata, full AI analysis results, raw uploaded files, and dense vector embeddings for semantic search. The combination of these data types particularly the 1536-dimensional embedding vectors stored for every text chunk requires a storage design that supports both structured relational queries and fast approximate nearest-neighbor search.

To address this, the team integrated the pgVector extension directly into PostgreSQL rather than deploying a separate vector database. This allows the system to perform cosine similarity search on the DocumentChunk table's embedding column using standard SQL queries, alongside the relational filtering needed to scope results to the current user and document. The

DocumentChunk table therefore serves a dual role: it is both a relational record (linked by foreign key to Document and User) and a vector index entry. For a typical document, ingestion produces between 20 and 100 chunks depending on length, each carrying a 1536-float embedding vector. The hybrid retrieval query combines a pgVector cosine search with a PostgreSQL full-text search on the text column, merging the two ranked lists via Reciprocal Rank Fusion to improve retrieval quality.

PDF files are stored in object storage rather than the application filesystem, keeping the backend stateless and allowing horizontal scaling without shared-volume concerns. The Document table stores the object key, and the serving layer generates short-lived pre-signed URLs so the client can stream PDFs directly from the bucket without proxying through the backend.

3.4. Access control and security

The access control and security design of Agreemind is centered on two goals: verifying user identity on every request, and ensuring that users can only access their own documents and data.

User passwords are never stored in plain text. At registration, each password is hashed using bcrypt with a per-user random salt before being written to the database. On login, the submitted password is verified against the stored hash using a constant-time comparison to prevent timing attacks. Upon successful authentication, the server issues a signed JWT access token using the HS256 algorithm. The token encodes the user's email address as the subject claim and includes an expiry timestamp. All subsequent requests from the client must include this token in the Authorization: Bearer header.

On the backend, every protected endpoint uses the get_current_user FastAPI dependency. This dependency decodes and verifies the JWT on each request, resolves the corresponding user record from the database, and injects it into the endpoint handler. If the token is missing, expired, or invalid, a 401 response is returned immediately. A secondary variant of this dependency, get_current_user_flexible, additionally accepts the token as a URL query parameter to support the in-app PDF viewer, which must function inside an iframe where header injection is not possible.

Data isolation is enforced at the query level on every endpoint. All document reads, analysis fetches, version history lookups, folder operations, and deletions include an explicit filters in the database query. A user cannot access another user's documents even with knowledge of the document's integer ID. Documents also carry an is_private flag (defaulting to True) which lays the groundwork for future sharing functionality without exposing data by default.

4. Subsystem Services

4.1. Client Subsystem

The client subsystem of Agreemind consists of two parallel applications a web application built with Expo and React Native Web, and a native mobile application built with React Native CLI that share the same overall screen structure and communicate with the same backend API. Both applications require the user to authenticate before accessing any protected screen; unauthenticated users are redirected to the login screen, and all protected routes verify a valid JWT token before rendering.

Navigation in the web application is handled through a persistent sidebar navbar with six primary destinations: Dashboard, Documents, Vault (AI chat), Alerts, Compare, and Rights. The mobile application mirrors this structure through a bottom tab bar. A shared set of themed UI components cards, risk badges, PDF viewer, and a storage abstraction layer is used across both platforms to ensure consistent behaviour. The client subsystem is displayed in Figure 2.

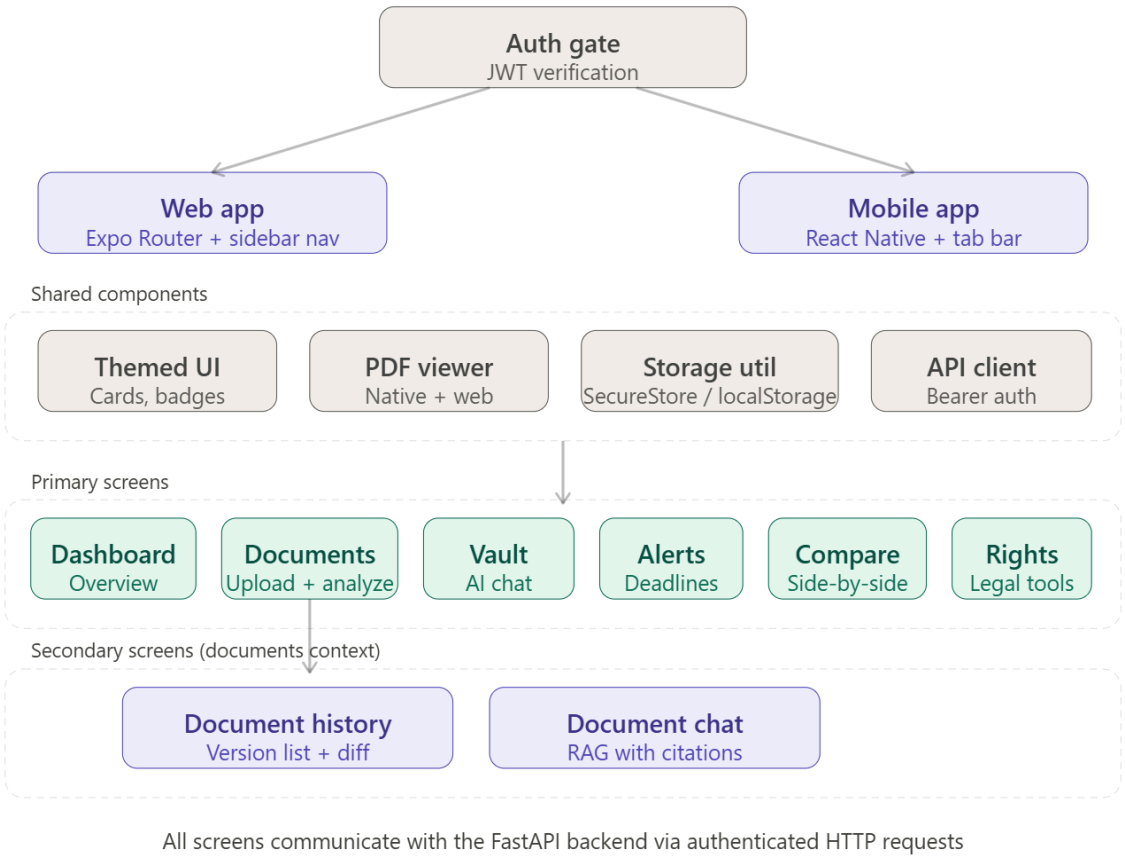


Figure 2: Client Subsystem Diagram

Document Upload and Analysis Flow

```
const handleUploadConfirm = async () => {
    setProcessing(true);
    const formData = new FormData();
    formData.append('file',        pendingFile);
    formData.append('name',        uploadName);
    formData.append('description', uploadDescription);
    formData.append('folder_id',   uploadFolderId ?? '');

    const res = await fetch(`${API_URL}/api/v1/documents/upload`, {
        method: 'POST',
        headers: { Authorization: `Bearer ${token}` },
        body: formData,
```

```
    });
    const doc = await res.json();

    // Poll until background ingestion + analysis completes
    let analysed = doc;
    while (analysed.ingestion_status !== 'completed') {
        await new Promise(r => setTimeout(r, 2000));
        const poll = await fetch(`${API_URL}/api/v1/documents/${doc.id}`, {
            headers: { Authorization: `Bearer ${token}` },
        });
        analysed = await poll.json();
    }
    setDocuments(prev => [analysed, ...prev]);
    setProcessing(false);
};
```

4.2. Backend Subsystem

The overall structure of the backend is displayed in Figure 3. All incoming requests first arrive at the FastAPI gateway, which applies CORS middleware, request logging, and JWT authentication before routing to one of four API routers: /auth, /documents, /chat, and /folders. The authentication service handles registration, login, and token verification using bcrypt password hashing and HS256 JWT tokens. Every protected endpoint uses a shared get_current_user dependency that decodes the token and injects the resolved user into the request handler.

The document service coordinates the full document lifecycle: it receives uploaded PDF files, uploads them to object storage and records the resulting object key, extracts text using PyMuPDF and pytesseract OCR, and then dispatches the extracted content to the contract analysis pipeline. The RAG service independently chunks the same document text, generates embeddings via the OpenAI embeddings API, and stores them in PostgreSQL with pgVector for later retrieval. The chat router uses the RAG service to answer user questions about their documents with source citations. All results are written to PostgreSQL, and document-level data isolation is enforced at the query level by filtering on the authenticated user's ID.

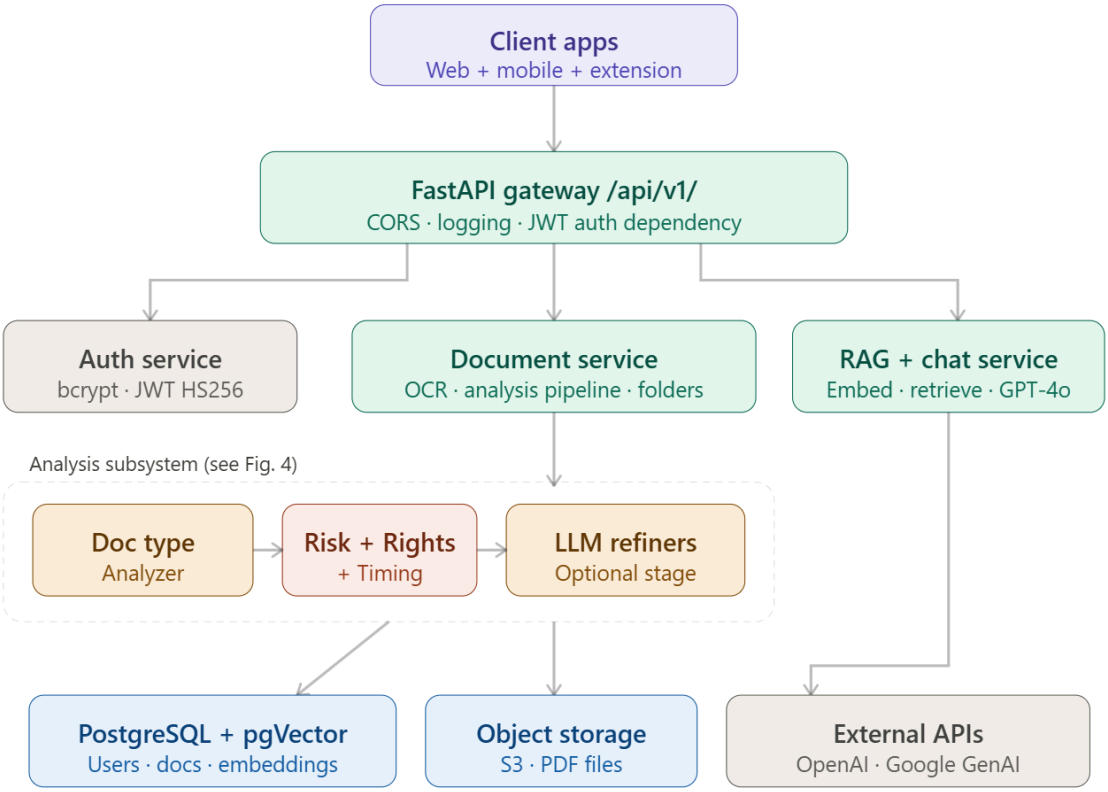


Figure 3: Backend Subsystem Diagram

Text Analysis Function

```
def analyze_text(self, raw_text: str) -> AnalysisResult:
    # ── Stage 1: segment into anchors / atoms ──────────────────────
    anchors = self.segmenter.process(raw_text)

    risk_candidates, rights_candidates, timing_candidates = [], [], []

    for anchor in anchors:
        for atom in anchor.atoms:
            if len(atom.text.strip()) < cfg.pipeline.min_atom_length:
                continue
            rights  = self.rights_engine.process(atom)
            timings = self.timing_engine.process(atom)
```

```
            rp      = self.risk_engine.predict_atom(atom)

            if self.risk_engine.is_risky(rp):
                risk_candidates.append((anchor.title, atom, rp))
            if rights:
                rights_candidates.append((anchor.title, atom, rights))
            if timings:
                timing_candidates.append((anchor.title, atom, timings))

    # ── Stage 2: optional LLM refinement (three independent lanes) ─
    if cfg.llm.enabled:
        refined_risks     = self.risk_refiner.refine(risk_candidates)
        refined_rights    = self.rights_refiner.refine(rights_candidates)
        refined_reminders = self.timing_refiner.refine(timing_candidates)
    else:
        refined_risks     = self.risk_engine.to_app_items(risk_candidates)
        refined_rights    = RightsExtractor.to_app_items(rights_candidates)
        refined_reminders = TimingExtractor.to_app_items(timing_candidates)

    refined_rights = self._merge_rights(refined_rights)
    score = compute_risk_score(refined_risks)

    return AnalysisResult(
        risk=refined_risks,   rights=refined_rights,
        reminders=refined_reminders,
        risk_score=score.overall, risk_level=score.level,
        risk_score_breakdown=score.by_category,
    )
```

4.3. Analysis Subsystem

The analysis subsystem is the core machine intelligence layer of Agreemind and is displayed in Figure 4. When a document is uploaded and its text extracted, it is first passed to the Document Type Analyzer, which classifies the document into one of the supported legal categories, such as rental agreements, employment contracts, terms of service, or insurance policies. This classification drives the selection of the appropriate Risk Classifier model, as separate fine-tuned transformer models have been trained for different document domains to improve precision.

Once the document type and risk model are determined, three parallel extraction engines process the document simultaneously. The Risk Classifier produces a multi-label risk assessment with per-category scores that are aggregated by the scoring module into an overall risk level of LOW, MEDIUM, or HIGH. The Rights Extractor uses rule-based patterns and spaCy dependency parsing to identify the user's legal rights and actionable entitlements within the document. The Timing Extractor uses regular expressions and NLP to surface key dates, deadlines, renewal windows, and payment obligations. An optional second stage passes each engine's output through a corresponding LLM refiner backed by OpenAI or Google GenAI which can improve extraction quality for ambiguous or complex documents. The final aggregated result is persisted to the database and returned to the client.

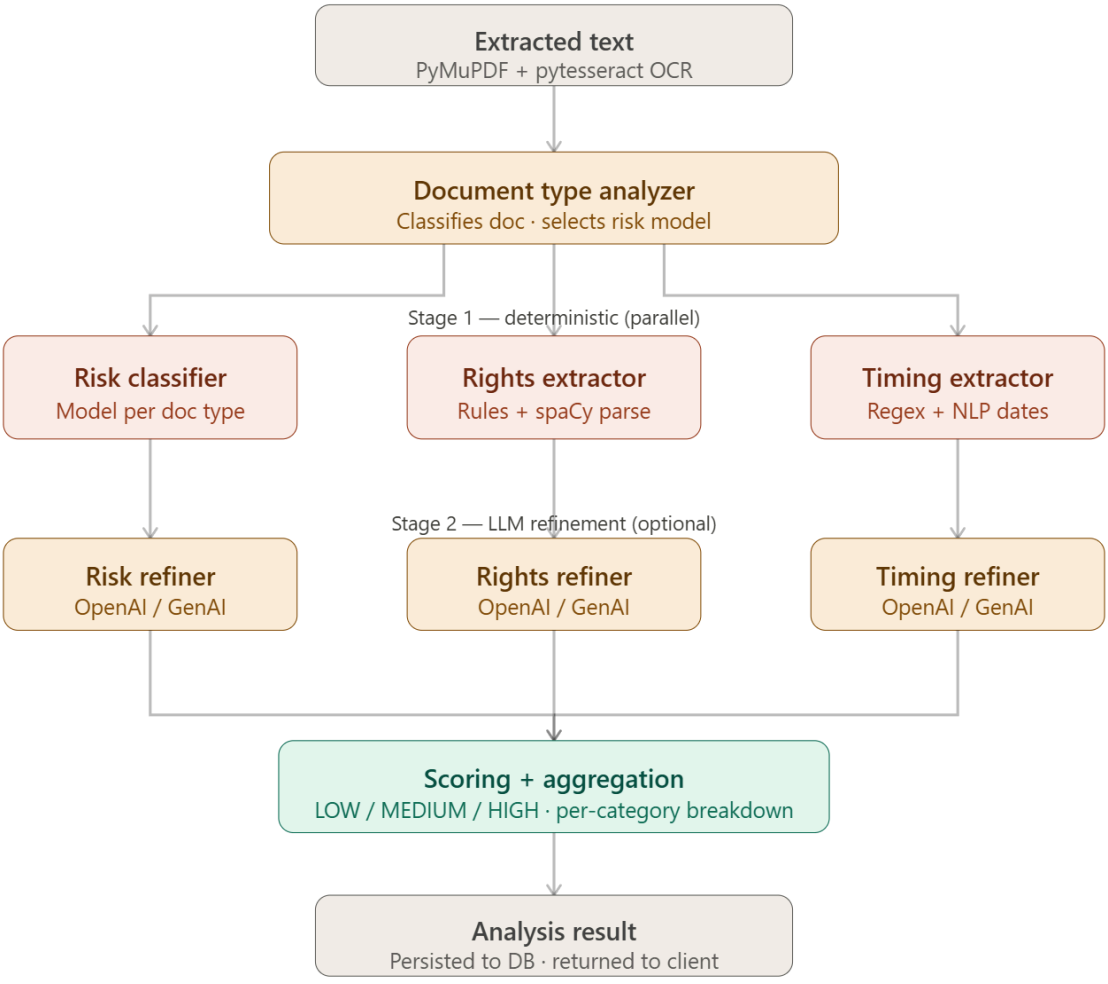


Figure 4: Analysis Subsystem Diagram

Risk Prediction Function

```
@torch.inference_mode()
def predict_atom(self, atom: Atom) -> RiskPrediction:
    enc = self.tokenizer(
        atom.text.strip(),
        truncation=True, max_length=self.max_length,
        padding=False,   return_tensors="pt",
    )
    enc = {k: v.to(self.device) for k, v in enc.items()}

    logits = self.model(**enc).logits.squeeze(0)  # shape: [num_labels]
    probs  = torch.sigmoid(logits).cpu().tolist()

    scores = {label: float(p) for label, p in zip(self.labels, probs)}
    flags  = [label for label, p in scores.items() if p >= self.threshold]

    return RiskPrediction(
        source_atom_id=atom.id,
        scores=scores,
        flags=flags,
        max_score=max(scores.values(), default=0.0),
    )
```

5. Test Cases

Test ID	TC-01	Category	Functional	Severity	
Objective	Verify that a registered user can log in successfully and access the main dashboard.				
Steps	1. Open the Agreemind web application. 2. Navigate to the login page. 3. Enter a valid registered email address and password. 4. Click the Log In button.				

Expected	1. The web application loads without errors and displays the landing/login interface. 2. The login form is displayed correctly with fields for email and password. 3. The system accepts the entered credentials without validation errors. 4. The user is authenticated successfully and redirected to the main dashboard or vault page.
Date	10.03.2026

Test ID	TC-02	Category	Functional	Severity	
Objective	Verify that a user can upload a PDF agreement and receive a summary and risk analysis.				
Steps	1. Log in to the Agreemind system. 2. Navigate to the agreement upload page. 3. Select a valid PDF agreement file from local storage. 4. Submit the file for analysis. 5. Wait for the system to complete processing.				
Expected	1. The user session is active and the system allows access to upload functionality. 2. The upload interface is visible and ready to accept a document. 3. The system accepts the PDF file and begins processing without file				

	format errors. 4. The uploaded agreement is sent to the backend successfully and an analysis task is created. 5. The system displays the generated plain-language summary together with detected risky clauses/categories.
Date	10.03.2026

Test ID	TC-03	Category	Functional	Severity	
Objective	Verify that risky clauses are highlighted correctly in the document viewer after analysis.				
Steps	1. Log in and upload or open an already analyzed agreement. 2. Wait until the analysis report page is displayed. 3. Click on one of the detected risk categories or flagged clauses. 4. Observe the corresponding location in the document viewer.				
Expected	1. The system opens the selected agreement and displays its associated analysis results. 2. The report page loads with a summary, risk list, and document viewer. 3. The selected risk item becomes active and triggers navigation to the related part of the agreement. 4. The relevant clause or text span is highlighted clearly in the document viewer, matching the selected risk item.				
Date	10.03.2026				

Test ID	TC-04	Category	Data Integrity	Severity	
Objective	Verify that an analyzed agreement can be saved to the vault and retrieved later through search.				
Steps	1. Log in to the Agreemind system. 2. Upload and analyze an agreement, or open an existing analyzed agreement. 3. Save the agreement to the personal vault. 4. Navigate to the vault page. 5. Search for the agreement by title or keyword.				
Expected	1. The user is authenticated and has access to vault operations. 2. The selected agreement is available in analyzed form and ready to be saved. 3. The system stores the agreement and confirms that it has been added to the personal vault. 4. The vault page displays saved agreements associated with the current user account. 5. The search returns the saved agreement correctly, and the user can reopen it without data loss.				
Date	10.03.2026				

Test ID	TC-05	Category	Functional	Severity	

Objective	Verify that two versions of the same agreement can be compared and differences are displayed correctly.
Steps	1. Log in to the Agreemind system. 2. Open the comparison feature from the dashboard or vault. 3. Select Version A and Version B of the same agreement. 4. Start the comparison process. 5. Review the comparison output.
Expected	1. The user is authenticated and can access the comparison functionality. 2. The comparison interface allows the user to choose two saved agreement versions. 3. Both versions are accepted by the system without compatibility or retrieval errors. 4. The system processes both versions and generates a comparison report. 5. The report clearly displays added, removed, or modified clauses, allowing the user to identify meaningful changes between the two versions.
Date	11.03.2026

Test ID	TC-06	Category	Functional	Severity	
Objective	Verify that the system rejects invalid login credentials and prevents unauthorized access.				
Steps	1. Open the Agreemind web application.				

	2. Navigate to the login page. 3. Enter an invalid email address or incorrect password. 4. Click the Log In button.
Expected	1. The application loads successfully and shows the login interface. 2. The login form is displayed correctly. 3. The system accepts the input format and submits the login request. 4. The system denies authentication, displays an appropriate error message, and does not grant access to the dashboard or vault.
Date	11.03.2026

Test ID	TC-07	Category	Functional	Severity	
Objective	Verify that the system supports agreement analysis through pasted text input.				
Steps	1. Log in to the Agreemind system. 2. Navigate to the agreement input page. 3. Paste a valid Terms of Service or contract text into the text input field. 4. Submit the text for analysis. 5. Wait for the result page to load.				
Expected	1. The user is authenticated and has access to the agreement input page. 2. The input area is visible and accepts pasted text. 3. The system stores the pasted text temporarily and validates that it is non-empty. 4. The system processes the pasted agreement text without requiring a				

	file upload. 5. The result page displays a summary and detected risky clauses/categories for the pasted text.
Date	11.03.2026

Test ID	TC-08	Category	Functional	Severity	
Objective	Verify that the system handles empty or invalid pasted input gracefully.				
Steps	1. Log in to the Agreemind system. 2. Navigate to the agreement input page. 3. Leave the text input field empty, or paste only whitespace. 4. Click the **Analyze** button.				
Expected	1. The user is logged in and the input page is accessible. 2. The text input field is displayed and editable. 3. The system accepts the button click and checks the provided input. 4. The system rejects the request, displays a validation message indicating that the agreement text cannot be empty, and does not start analysis.				
Date	11.03.2026				

Test ID	TC-09	Category	Functional	Severity	

Objective	Verify that unsupported or invalid file types are rejected during upload.
Steps	1. Log in to the Agreemind system. 2. Navigate to the agreement upload page. 3. Select a file in an unsupported format (for example, .exe or another invalid type). 4. Attempt to submit the file for analysis.
Expected	1. The user is authenticated and the upload page is accessible. 2. The upload interface is displayed correctly. 3. The system receives the selected file and checks its type before processing. 4. The system rejects the file, informs the user that the file format is unsupported, and does not create an analysis task.
Date	11.03.2026

Test ID	TC-10	Category	Functional	Severity	
Objective	Verify that the browser extension can send a webpage for analysis and open the result in Agreemind.				
Steps	1. Open a webpage containing Terms of Service or other agreement text in the browser. 2. Click the Agreemind browser extension icon. 3. Trigger the analysis action from the extension interface. 4. Wait for the extension to send the page content or URL to the				

	backend. 5. Open the generated result in the Agreemind web application.
Expected	1. The webpage loads successfully and the extension is available in the browser. 2. The extension interface opens without errors and recognizes the current page context. 3. The extension successfully starts the analysis request. 4. The system receives the page information and processes it without requiring manual copy-paste. 5. The analysis result opens in Agreemind and displays the expected summary and risk findings for the selected webpage.
Date	11.03.2026

Test ID	TC-11	Category	Functional	Severity	
Objective	Verify that a new user can create an account successfully through the sign-up flow.				
Steps	1. Open the Agreemind web application. 2. Navigate to the sign-up page. 3. Enter valid registration information, including email and password. 4. Submit the sign-up form. 5. Wait for account creation to complete.				

Expected	1. The application loads successfully and displays the authentication interface. 2. The sign-up form is accessible and fields are displayed correctly. 3. The system accepts the entered registration information without validation errors. 4. The system creates a new user account and confirms successful registration. 5. The user is redirected to the dashboard or prompted to log in with the new account.
Date	13.03.2026

Test ID	TC-12	Category	Functional	Severity	
Objective	Verify that a logged-in user can log out successfully and lose access to protected pages.				
Steps	1. Log in to the Agreemind system. 2. Navigate to the main dashboard or vault page. 3. Click the Log Out button. 4. Attempt to revisit a protected page such as the vault or dashboard.				
Expected	1. The user is authenticated successfully and the dashboard is accessible. 2. Protected pages are visible only to the authenticated user. 3. The system ends the user session and redirects the user to the login or landing page. 4. Access to protected pages is denied until the user logs in again.				

Date	13.03.2026

Test ID	TC-13	Category	Functional	Severity	
Objective	Verify that the vault preserves the correct association between agreements and their analysis results after multiple uploads.				
Steps	1. Log in to the Agreemind system. 2. Upload and analyze two different agreements. 3. Save both agreements to the personal vault. 4. Open each saved agreement from the vault one by one. 5. Compare the displayed summaries and risk findings with the originally generated results.				
Expected	1. Log in to the Agreemind system. 2. Upload and analyze two different agreements. 3. Save both agreements to the personal vault. 4. Open each saved agreement from the vault one by one. 5. Compare the displayed summaries and risk findings with the originally generated results.				
Date	13.03.2026				

Test ID	TC-14	Category	Functional	Severity	
Objective	Verify that searching the vault with a non-matching keyword is handled				

	clearly and without errors.
Steps	1. Log in to the Agreemind system. 2. Navigate to the personal vault. 3. Enter a keyword that does not match any saved agreement title or content. 4. Submit the search query. 5. Observe the search results area.
Expected	1. The user is authenticated and the vault page loads correctly. 2. The vault interface displays the search bar and previously saved agreements. 3. The system accepts the keyword and processes the search request. 4. The system completes the search without crashing or freezing. 5. The user is shown a clear “no results found” state instead of an empty or broken interface.
Date	13.03.2026

Test ID	TC-15	Category	Functional	Severity	
Objective	Verify that the system responds within an acceptable time when analyzing a typical agreement document.				
Steps	1. Log in to the Agreemind system. 2. Upload a valid agreement document of typical size. 3. Submit the document for analysis. 4. Measure the elapsed time until the summary and risk results are				

	displayed.
Expected	1. The user session is active and upload functionality is available. 2. The system accepts the document without errors. 3. The backend begins processing the uploaded document and provides visible feedback that analysis is in progress. 4. The system completes analysis within an acceptable response time for a normal document and displays the results without timeout or failure.
Date	13.03.2026

Test ID	TC-16	Category	Functional	Severity	
Objective	Verify that the sign-up process rejects registration when an already registered email address is used.				
Steps	1. Open the Agreemind web application. 2. Navigate to the sign-up page. 3. Enter an email address that is already associated with an existing account. 4. Fill in the remaining required fields with valid values. 5. Submit the sign-up form.				
Expected	1. The application loads correctly and the sign-up page is accessible. 2. The sign-up form is displayed properly. 3. The system accepts the entered email format and other inputs. 4. The system checks whether the email is already registered. 5. The system rejects the registration request, informs the user that the				

	email is already in use, and does not create a duplicate account.
Date	13.03.2026

Test ID	TC-17	Category	Functional	Severity	
Objective	Verify that the system handles corrupted or unreadable PDF files gracefully during upload.				
Steps	1. Log in to the Agreemind system. 2. Navigate to the agreement upload page. 3. Select a corrupted or unreadable PDF file. 4. Submit the file for analysis. 5. Observe the system response.				
Expected	1. The user is authenticated and can access the upload page. 2. The upload interface is available and accepts the selected file. 3. The system receives the file and attempts to parse it. 4. The system detects that the PDF cannot be processed correctly. 5. The system reports the failure clearly to the user and does not generate a misleading summary or analysis result.				
Date	13.03.2026				

Test ID	TC-18	Category	Functional	Severity	

Objective	Verify that the comparison feature handles invalid version selections safely.
Steps	1. Log in to the Agreemind system. 2. Open the comparison page. 3. Select only one agreement version, or select an invalid/missing version combination. 4. Attempt to start the comparison process.
Expected	1. The user is authenticated and the comparison page loads correctly. 2. The comparison interface is displayed and allows version selection. 3. The system accepts the user interaction and validates the selected versions before comparison starts. 4. The system prevents the comparison request from proceeding, informs the user that two valid versions are required, and avoids generating an invalid comparison report.
Date	13.03.2026

Test ID	TC-19	Category	Functional	Severity	
Objective	Verify that a user cannot access another user’s saved agreements or analysis results.				
Steps	1. Log in as User A and save at least one analyzed agreement in the vault. 2. Log out from User A’s account. 3. Log in as User B with a different account. 4. Navigate to the vault and attempt to access User A’s saved agreement				

	directly or indirectly.
Expected	1. User A can authenticate and store agreements in their own vault. 2. User A's session is terminated successfully after logout. 3. User B is authenticated successfully and sees only their own account context. 4. The system prevents User B from viewing, opening, or retrieving any agreement or analysis result that belongs to User A.
Date	13.03.2026

Test ID	TC-20	Category	Functional	Severity	
Objective	Verify that the browser extension handles unsupported or non-analyzable pages without crashing.				
Steps	1. Open a webpage where agreement text cannot be extracted properly (for example, a highly dynamic or unsupported page). 2. Click the Agreemind browser extension icon. 3. Attempt to start the analysis from the extension. 4. Observe the response shown by the extension or connected web app.				
Expected	1. The webpage loads and the extension remains available in the browser. 2. The extension interface opens correctly. 3. The system attempts to process the page content or URL and detects that extraction is not possible or insufficient.				

	4. The extension or web app informs the user clearly that analysis could not be completed for the current page and does not crash, freeze, or display invalid results.
Date	13.03.2026

6. Consideration of Various Factors in Engineering Design

6.1. Constraints

6.1.1. Public Health

Public health considerations had no direct impact on the design of Agreemind since the system is primarily a digital legal analysis platform rather than a healthcare or safety-critical system.

Impact Level: **0 / 10**

6.1.2. Safety

Safety considerations were evaluated primarily in the context of system reliability and preventing harmful misuse of information. While Agreemind does not directly affect physical safety, incorrect or misleading analysis of legal documents could potentially lead to poor user decisions. To mitigate this risk, the system presents analysis results as informational guidance rather than legal advice. The interface clearly links generated insights to the original contract text so users can verify interpretations.

Impact Level: **3 / 10**

6.1.3. Security

Security was one of the most influential factors in the design process. Agreemind processes potentially sensitive documents such as rental agreements, employment contracts, and terms of service. Protecting user data and maintaining confidentiality were therefore major priorities. The system design includes encrypted communication between system components, secure storage of documents in protected storage services, and logical separation of user data. Additionally, access control mechanisms ensure that only authorized users can access their stored agreements.

Impact Level: **9 / 10**

6.1.4. Welfare

User welfare was an important design consideration. Many individuals accept agreements without fully understanding their consequences due to complex legal language. Agreemind aims to empower users by providing simplified explanations, highlighting risky clauses, and identifying obligations and deadlines. By increasing transparency and awareness, the system helps users make more informed decisions about agreements that affect their daily lives.

Impact Level: **7 / 10**

6.1.5. Global Factors

Global factors were considered mainly in terms of deployment and accessibility. The system is designed to operate on widely used platforms such as web browsers and mobile devices, allowing users from different regions to access the service. Additionally, the modular architecture allows the integration of jurisdiction-specific rule sets and regulatory knowledge bases in the future, enabling the system to adapt to different legal environments.

Impact Level: **5 / 10**

6.1.6. Cultural Factors

Cultural factors were considered during the design of the system because legal language, contract structures, and expectations about agreements can vary across different cultures and regions. People from different cultural backgrounds may interpret legal terms differently or may have different levels of familiarity with legal documents. To address this, the system focuses on providing clear explanations and maintaining traceability between generated insights and the original document text. This helps users verify the analysis and reduces the risk of misinterpretation. However, the current system primarily assumes agreements written in English and common consumer contract formats. Supporting multiple languages and adapting explanations to different legal cultures may require further improvements in future versions of the system.

Impact Level: **2 / 10**

6.1.7. Social Factors

Social factors played a meaningful role in the design of Agreemind. Many individuals lack access to professional legal assistance due to cost or availability. By providing automated analysis of agreements, the system aims to reduce this accessibility gap and make legal information easier to understand for everyday users. The platform is therefore designed to prioritize usability and clarity for non-expert users.

Impact Level: **6 / 10**

6.1.8. Environmental Factors

Environmental considerations had limited influence on the system design. As a cloud-based software platform, Agreemind does not involve physical manufacturing or hardware components. However, efficient cloud resource usage and scalable architecture were considered to minimize unnecessary computational overhead and energy consumption during large-scale document analysis.

Impact Level: **2 / 10**

6.1.9. Economic Factors

Economic factors influenced several design decisions. The system is designed to provide an affordable alternative to traditional legal consultation for understanding agreements. The architecture emphasizes scalable cloud infrastructure and modular services so that operational costs can remain manageable while supporting a growing number of users. These considerations also support the long-term sustainability of the platform.

Impact Level: **5 / 10**

Constraint	Impact level	Effect
Public health	None	The project is not related to public health.
Public safety	Medium	The system must safely handle user data and prevent misuse.
Security	High	Strong data protection and secure storage are required for user documents.
Public welfare	High	Helps users better understand agreements and avoid harmful terms.
Global factors	Medium	Legal regulations vary across countries and may affect system expansion.
Cultural factors	Low	Legal interpretation and expectations vary across cultures.
Social factors	High	Improves access to legal understanding for non-experts.
Environmental factors	Low	The system has minimal environmental impact as a software platform.
Economic factors	Medium	Automated analysis can reduce time and cost for users.

Table 1: Table of Constraints

6.2. Standards

6.2.1. IEEE 830 - Software Requirements Specifications

The IEEE 830 standard is utilized to rigorously define the functional and non-functional requirements of the system, ensuring clarity and completeness in the specification document.

6.2.2. UML 2.5.1 - Unified Modeling Language

UML 2.5.1 is employed as the standard for visualizing the system's architecture, ensuring that the interactions between the diverse client applications and the backend are clearly mapped. Specifically, Sequence and Class diagrams will be used to model the modular architecture.

6.2.3. REST API Guidelines

The system adheres to REST architectural principles to ensure stateless and scalable communication between the backend and its various clients, including the web dashboard, mobile app, and browser extension. This standardization allows for efficient resource management.

7. Teamwork Details

We created an issue list and listed all the features we required. Each team member chose a set of tasks from the list that aligned with their previous experiences, technical skills.

7.1. Contributing and functioning effectively on the team

- We held biweekly sprints and meetings along with our supervisor in which we decided on the tasks for the upcoming sprint.
- Project reports were prepared with an equal and fair distribution of tasks.
- Frontend and the UI of the project was designed collaboratively, each member designed the UI of their respective functionality. Ata Soykal refined the overall user interface.
- Edip Emre Dönger and Ata Oğuz worked on the Android and IOS applications respectively.
- Backend functionalities were implemented by Ata Oğuz, Emir Görgülü, Edip Emre Dönger.
- Machine Learning models for the various legal fields were designed by Can Polat Bülbül and integrated into the app by Edip Emre Dönger.

7.2. Helping creating a collaborative and inclusive environment

- By regularly meeting and discussing the many matters involving the project we made all decisions unanimously. During these discussions each group member shared their vision of the app and their ideas.
- We used JIRA's issue trackers and Github to create a seamless collaboration environment where each member was able to actively monitor what the other members were working on.
- To ensure smooth communication outside of scheduled meetings, we used WhatsApp and Discord as our primary messaging channels, allowing team members to stay informed, share updates, and address blockers quickly regardless of time or location.

7.3. Taking lead role and sharing leadership on the team

Each member of the team held a distinct responsibility, and the lead role was shared flexibly across the project's phases, allowing every member to develop ownership over their area while supporting others.

- Ata Oğuz led the implementation of the core backend services, including the API architecture and database design. He coordinated the backend development effort and ensured that services were structured in a way that made integration straightforward for the rest of the team. He also implemented the RAG search functionalities.
- Ata Soykal led the UI design for both web and mobile clients, and worked on improving the user experience and also the aesthetics of the application.
- Can Polat Bülbül led the research and development of the machine learning models used for legal document analysis across different domains. He evaluated and selected the appropriate models for each document type and designed the analysis pipeline architecture.
- Emir Görgülü led the development of backend processing pipelines and the integration of external services. He was responsible for the document ingestion workflow and the retrieval-augmented generation system that powers the chat functionality.
- Edip Emre Dönger led the integration of the machine learning models into the application. He connected the ML layer and the backend, ensuring that model outputs were correctly consumed by the processing pipeline and surfaced accurately to users. He also implemented the analysis and refinement pipelines.

8. Glossary

Agreemind: The proposed consumer-facing legal-document assistant that summarizes agreements, highlights risks, and helps users track obligations without providing legal advice.

Agreement: A legal text the user uploads or shares (e.g., Terms of Service, Privacy Policy, rental/subscription contract).

Clause: A meaningful segment of an agreement (sentence/paragraph/section) that expresses a rule, right, limitation, or obligation.

Risk Flag: A detected clause category that may be unfavorable to the user (e.g., data sharing, auto-renewal, unilateral change, arbitration).

Plain-Language Summary: A simplified explanation of an agreement or clause written for non-expert users.

Obligation: An action the user must do (or avoid) according to the agreement (e.g., payment, notice submission, compliance requirement).

Deadline / Notice Period: A time constraint extracted from the agreement (e.g., cancellation window, renewal date, "within 30 days").

Personal Vault: A secure personal repository where a user's processed agreements, reports, and metadata are stored for later search and comparison.

Version Comparison: A feature that identifies and presents changes between two versions of the same agreement ("what changed?").

Analysis Pipeline: The backend processing steps applied to an agreement (ingestion → text extraction → chunking → retrieval/classification → summarization → report).

AnalysisJob: A backend job that represents one analysis request from a user and its processing state (queued/running/completed/failed).

ConsentRecord: A stored record that the user explicitly permitted an agreement to be processed (especially important if external APIs are used).

RAG (Retrieval-Augmented Generation): A method where the system retrieves relevant text passages and constrains the LLM to answer using that context.

Embedding: A numeric vector representation of text used to support semantic search and retrieval in the vault.

Vector Store: A database/index optimized for similarity search over embeddings (used for vault querying and context retrieval).

HNSW: A graph-based approximate nearest neighbor indexing method commonly used for fast vector similarity search.

LLM (Large Language Model): A model used to generate summaries/explanations; in your system it must be constrained to informational output (not legal advice).

Custom Model: A smaller model you train/fine-tune for a specific subtask (e.g., risk classification or date extraction) to reduce cost and dependency on external APIs.

NER (Named Entity Recognition): A technique to detect structured entities in text (e.g., dates, organizations, money amounts).

Share Sheet / Share Intent: Mobile OS functionality that lets the user share a webpage/text into Agreemind for on-demand analysis (instead of background monitoring).

Privacy by Design: Designing the system to minimize data collection, enforce access control, and prevent model training on user data without explicit opt-in.

GDPR / Right to be Forgotten: Data protection requirements that include user deletion/export rights and limits on data retention/processing.

9. References

[1] Ironclad – AI-powered Contract Lifecycle Management Software (2025). Retrieved from https://ironcladapp.com/product/ai-based-contract-management

[2] Kira Systems – AI-powered Contract Analysis Software (2025). Retrieved from https://kira.ai/solutions/legal-workflow

[3] Luminance – Legal-Grade AI Contract & Document Review Software (2025). Retrieved from https://luminance.com/solutions/legal/

[4] ToS;DR – Crowd-sourced ToS & Privacy Policy Ratings (2025). Retrieved from https://tosdr.org

[5] Open Terms Archive – Public Archive of Online Terms & Conditions (2025). Retrieved from https://opentermsarchive.org

[6] Termzy AI – Real-time ToS Detection Software (2025). Retrieved from https://www.termzyai.com/#features