IT314 - Software Engineering
GROUP - 32 AdvocAI

202301011	Hiya Modi
202301247	Sharnam Shah
202301443	Vansh Patel
202301444	Divyesh Vagh
202301446	Jainesh Patel
202301450	Shubham Varmora
202301472	Parth Karena
202301473	Nilesh Mori
202301481	Kalp Shah
202301483	Dhanush Pillai
202301486	Maharshi Patel

Group Leader - Maharshi Patel

UNDER THE GUIDANCE OF:
PROF : *DR.SAURABH TIWARI*
MENTOR : *Jay Goyani*

Sprint 1

Users and Stakeholders

Users	Stakeholders
Individuals and General Consumers	All Users
Small Business Owners	Investors (who invested in the building of this platform)
Professionals in Non-Legal Fields(HR manager, Finance and Accounting Professionals, Real Estate Agents and Property Managers)	Development ,Testing and Product Team
Lawyers	Legal Professionals and Law Firms (Potential Partners)
	Regulatory Bodies and Government Agencies
	Legal Aid Organizations and Non-Profits

A mix of research, brainstorming, discussion and thinking was carried out to identify users and stakeholders.

Elicitation Techniques

1) **BRAINSTORMING METHOD**

DEFINITION

Brainstorming is a group creativity technique used to generate a large number of ideas in a short time without judgment. It encourages open thinking and helps identify innovative and diverse requirements.

WHY BRAINSTORMING FOR OUR PROJECT?

- Platform requires knowledge from law, AI, UX, business
- Users, lawyers, developers have different perspectives
- Helps combine legal accuracy + usability + feasibility
- Enables innovative solutions like voice-based help, smart clause suggestions, multilingual support

PROCESS FOLLOWED

Participants: 10 (virtual)

Roles: Moderator, Scribe, Legal experts, Developers, Users

Steps followed:

1. Asked participants about problems with legal documents
2. Collected real-life pain points
3. Users shared expectations from the platform
4. All ideas written down without filtering
5. Discussed feasibility and legal correctness
6. Grouped ideas by theme
7. Prioritized using MoSCoW method
8. Final raw + prioritized requirement list documented

IDEAS GENERATED (CATEGORIZED)

A) Document Simplification

- AI plain-language summaries
- Risk clause identification
- Voice explanations

B) Document Generation & Editing
• Standard templates auto-filled using user details
• Collaborative commenting
• AI missing clause suggestions

C) Version Tracking & E-Signature
• Version history with comparison
• Rollback options
• Legally valid digital signatures

D) Lawyer Connect
• Consultation request system
• Lawyer document review before accepting case

E) Compliance & Security
• Mandatory clause checking
• Encrypted cloud document storage

F) User Support & Accessibility
• Multi-language support
• Voice input features

MOSCOW PRIORITIZATION :-

Must Have:

- AI document simplification
- Legal document generation
- Multi-party comments
- Version tracking and rollback
- E-signature
- Lawyer connect
- Mandatory clause checking
- Secure access control

Should Have:
• High-risk clause highlighting
• Lawyer annotations
• Strong privacy & security measures

Could Have:
• Voice-based help system
• Infographic summaries
• Multi-language UI

Won't Have Now:
• Blockchain contract validation
• Fully automated smart contracts

REQUIREMENTS DERIVED

Functional:
• Document summarizer
• Legal document generator
• E-signature support
• Lawyer consultation system
• Version control
• Multi-party document collaboration

Non-Functional:
• Data privacy & secure storage

USER STORIES:

User Story ID	User Story	Acceptance Criteria
1	As a user, I want complex legal documents to be simplified so that I can easily understand them.	• Plain-language summaries provided• Key clauses highlighted• AI suggests lawyer help if unsure
2	As a user, I want to generate legal documents using standard templates personalized with my details so that I can avoid drafting from scratch.	• Provides standard templates (NDA, Rent Agreement, etc.)• Auto-fills details from user input• Legally compliant structure
3	As a stakeholder, I want to comment and make suggestions collaboratively so that document changes are agreed upon by everyone.	• Inline comments supported• Author identity tracked• Approved comments update the document
4	As a user, I want version tracking so that I can restore previous document versions if needed.	• Each edit saved as a version• Rollback supported• Metadata maintained (editor, timestamp)
5	As a user, I want digital e-signatures so that documents can be finalized legally without printing.	• Valid e-signatures supported• Signed copies secured• Signature details stored securely
6	As a user, I want to request consultation with a lawyer so that I can get legal advice for complex issues.	• Lawyer request option available• Lawyer notified with case info• Accept/reject response visible to user

7	As a user, I want the system to check mandatory legal clauses so that I do not miss important requirements.	• AI checks for required clauses• Alerts for missing clauses• Suggests additions
8	As a user, I want my documents to be securely stored so that only authorized parties can access/edit them.	• Encrypted storage• Role-based access control• Logs maintained

2) INTERVIEWS

OVERVIEW

Interviews were conducted to gather expectations, challenges, and needs directly from stakeholders related to legal documentation and legal AI usage.

STAKEHOLDERS INTERVIEWED

1) Law Students
2) Person working in Legal-AI industry (Sarvam AI employee)
3) Common Users (non-legal background)

INTERVIEW SHEME

• Mode: Voice/Video call (one-to-one)
• Duration: 20–30 minutes each
• Documentation: Notes taken manually

INTERVIEW QUESTIONS

Law Students

1. Challenges in understanding legal documents?
2. Do you need assistance? How often?
3. Will you trust an AI tool for legal analysis?
4. Concerns about AI-generated documents (security, accuracy)?
5. Would lawyer assistance + AI be useful?

Employee – Legal AI industry (Sarvam AI)

1. How do you handle hallucinations of the model?
2. Biggest challenges in training?
3. Types of documents used?
4. Formatting rules?
5. How is privacy maintained?
6. Future scope of AI in legal domain?
7. Token limits?
8. Methods of digital authentication?
9. Source of legal information?

Users / Common People

1. How do you currently resolve legal queries?
2. Frequency of difficulty understanding legal documents?
3. Expected features in a legal document generator?
4. Preference for simple explanations by AI?
5. Importance of lawyer verification?
6. Hesitations/fears in AI-generated legal docs?

SUMMARY OF INTERVIEWS

Law Students

- Struggle with complex legal language
- Prefer AI summaries with lawyer support
- Security and reliability are major concerns

Common Users

- Rely on friends/lawyers but still face confusion
- Need simple UI, easy explanations, lawyer verification
- Worry about data privacy and AI accuracy

Industry Employee

- Data formatting issues still handled manually

• Uses verified legal sources (e.g., court records)
• Privacy and hallucination control are key challenges

FUNCTIONAL REQUIREMENTS DERIVED

1. Document generation using prompts
2. Document upload + summary/analysis
3. Lawyer connection feature
4. E-signature and authentication
5. Premium feature for lawyer escalation
6. User account and secure document management

NON-FUNCTIONAL REQUIREMENTS DERIVED

1. Security & privacy — end-to-end encryption
2. Scalability — handle many users smoothly
3. Performance — fast response to actions
4. Transparency — difference between AI & lawyer outputs
5. Accuracy — reduced hallucination
6. Usability — simple interface
7. Reliability — trusted LLM + verified lawyers

USER STORIES

User Story ID	User Story	Acceptance Criteria
US-I1	As a user, I want the system to explain legal documents in simple language so that I can understand them easily.	• System highlights complex clauses• Provides plain-English explanations• Suggests lawyer assistance if AI is unsure
US-I2	As a user, I want to generate legal documents by providing required inputs so that I can directly use them.	• Document generated quickly• Proper formatting as per legal standards• Content relevant to jurisdiction

US-I3	As a user, I want my generated documents to be verified and signed by experts so that I can trust their legal validity.	• Verification notification shown• System provides authentication stamp/QR• Signed copy stored securely
US-I4	As a user, I want to communicate with lawyers/legal experts so that I can get personalized help for legal issues.	• Chat support available• Lawyer details shared only after consent• Secure message exchange ensured
US-I5	As a user, I want my legal documents and details to remain private and secure so that I can safely use the platform.	• End-to-end encryption applied• Secure authentication required• No unauthorized access allowed

3) SURVEYS METHOD

DEFINITION

The Survey Elicitation Technique is a method of collecting structured input from stakeholders through carefully designed surveys. It is used to gather both functional requirements (specific features or services users need) and non-functional requirements (qualities such as trust, usability, and personalization) to guide system design and development. This technique helps in understanding user needs, expectations, and preferences, allowing for the mapping of these insights to system features.

WHY WE USED THIS METHOD

• Relevance to Target Users: Since our platform (AdvocAI) is user-centric, we needed direct feedback from potential users.

• Scalability: A single Google Form could reach a large and diverse audience quickly.

• Balanced Data Collection: The form had closed-ended questions for quantitative analysis and open-ended questions for qualitative insights.

• Ease of Participation: Stakeholders could respond anytime and anonymously, encouraging honest answers.

• Efficient Requirement Gathering: This allowed us to collect both functional and non-functional requirements simultaneously.

HOW WE CONDUCTED THE SURVEY

• Step 1: Preparation

We defined the objective of the survey: to understand how users interact with legal documents, their pain points, and the features they would like in AdvocAI.

• Step 2: Form Design

Included the following question categories:

- ○ Legal Document Features: Difficulty in understanding legal language, usefulness of summarization, types of documents (contracts, property, business, employment)
- ○ Lawyer Connection Features: Preference for risk/highlight extraction, interest in connecting with lawyers, preferred communication modes, selection criteria
- ○ AI Summariser Preferences: Comfort with AI for legal queries and kind of assistance expected
- ○ Feedback Inputs: Open-ended questions regarding challenges, trust issues, and suggestions

REQUIREMENTS EXTRACTED

FUNCTIONAL REQUIREMENTS (**FRs**)

1. Document Upload & Summarization

• Upload legal documents (contracts, property, corporate, HR, etc.)

• Generate summaries showing key risks, obligations, and clauses

2. Lawyer Connection & Consultation

• Option to connect with lawyers after AI summary

• Communication modes: chat, call, video, email

• Lawyer selection with filters: expertise, fees, experience, availability

3. AI Chatbot Assistance

• Provide legal definitions and explanations

• Step-by-step guidance for drafting simple legal documents (e.g., NDA)

• Suggest next steps for legal issues
• Refer user to a human lawyer when necessary

NON-FUNCTIONAL REQUIREMENTS (NFRs)

1. Accuracy & Reliability — Summaries and chatbot responses must be precise and consistent
2. Security — Ensure encrypted storage of documents and secure lawyer communication
3. Usability — Easy-to-use interface for non-legal users
4. Performance — Fast AI responses for summarization and help
5. Scalability — Handle multiple users and consultations parallelly
6. Trust — Clear explanation of AI decisions and verified lawyer profiles

USER STORIES

ID	User Story (Front Card)	Acceptance Criteria (Back Card)
US-S1	As a non-professional user, I want the facility to summarize general contracts and agreements, real estate and property-related documents, business contracts, application forms, and employment/HR documents so that I can easily understand them without wasting time.	• User can upload contracts, property, business, HR docs (PDF/Word) • System generates a clear, plain-language summary • Summary generated within acceptable time (≤5 minutes) • Disclaimer shown suggesting lawyer consultation if AI is unsure
US-S2	As a non-professional user, I want to connect with a lawyer via chat, audio/video call, or email after generating the AI summary so that I can confirm legal correctness.	• Consultation option available after summary • User can choose chat/call/video/email • Lawyer list visible with availability • Lawyer must request permission to access document • User is notified when lawyer accepts/schedules session

US-S3	As a non-professional user, I want to view lawyers' expertise, experience, fees, ratings, and reviews so that I can select the right lawyer.	• Lawyer profile visible to users • Lawyer must verify expertise to show on profile • Search and filter options available
US-S4	As a non-professional user, I want a chatbot that does not reveal my information to third parties so that I feel secure using the service.	• No data sharing with third parties • End-to-end encrypted communication • Privacy policy clearly shown • User can delete chat history any time • Logs visible only to logged-in authorized users
US-S5	As a legal professional, I want a feature to create a public profile and preview documents before accepting queries so that I can manage cases efficiently.	• Lawyer can create/edit profile • Profile shows expertise, experience, fees, ratings, reviews • Lawyer can preview documents securely • User notified if lawyer declines request

4) ANALYSIS OF EXISTING SYSTEMS

TECHNIQUE DEFINITION

This elicitation technique involves studying existing legal-tech platforms to understand:

• What features they already provide

• Where they perform well

• Where gaps exist that we can fulfill

Goal → Ensure we build a superior and differentiated solution instead of reinventing what already exists.

WHY WE USED THIS METHOD

• Legal-tech is already an active domain — existing solutions offer useful learnings

• Helps us design competitive advantages and address key user trust concerns

• Validates our unique feature → "Lawyer connectivity," missing in most AI

tools
• Helps avoid repeating the same limitations as competitors

COMPETITORS ANALYZED
• LegalReview.AI
• Flair
• LegalFly
(Features studied through demos, documentation, and user reviews)

ANALYSIS PROCESS
We compared competitor systems under three capability groups:
Core Features → Extract clauses, generate document summary
Value-Added Features → Collaboration, basic e-signature support
Weak / Missing Areas → Risk highlighting, human lawyer assistance

Mapping → Their gaps = Our opportunity
Used results to refine AdvocAI feature roadmap.

FUNCTIONAL REQUIREMENTS DERIVED
1) Upload legal documents in multiple formats
2) Extract key clauses (dates, obligations, penalties)
3) Highlight risky/unclear terms
4) Plain-English document summaries
5) Version tracking + rollback support
6) E-signature capability
7) Threaded collaborative comments
8) Connect users to verified lawyers
9) Secure export & sharing of documents

NON-FUNCTIONAL REQUIREMENTS DERIVED
• High accuracy in clause extraction & summarization
• Stable performance for large documents
• Strong security of sensitive legal data

• Scalable multi-user usage without performance drop

KEY FINDINGS (**GAP ANALYSIS**)

• Automation is strong, but comprehension support is weak
• Risky clause detection missing or inaccurate
• Users still struggle → No lawyer availability for confirmation
• Collaboration limited → mostly simple file sharing, not true commenting

Result → A clear need for explainability + trust + human expertise

CONCLUSION

Our system improves on competitors by offering:

• AI assistance for clarity
• Risk transparency
• Human-in-the-loop legal guidance
• Stronger collaboration and document lifecycle features

USER STORIES

User Story ID	User Story (Front Card)	Acceptance Criteria (Back Card)
1	As a small business owner, I want to upload my contract and receive a plain-English summary so that I can understand it without needing a lawyer every time.	• User can upload PDF/Word documents• System generates simplified summary• Summary includes key details like dates, obligations, penalties
2	As a startup founder, I want risky or unclear terms to be highlighted so that I do not accidentally agree to harmful conditions.	• System highlights ambiguous or high-risk clauses• Highlighted terms are clearly visible• System provides short explanations for each flagged clause

3	As a compliance officer, I want version tracking and comments so that my team can collaborate efficiently on legal document edits.	• System stores multiple versions• Users can compare changes• Threaded comments linked to specific document sections
4	As a general user, I want to ask a chatbot questions about legal clauses so that I can quickly understand document details.	• Chatbot provides contextually accurate answers• If uncertain, chatbot suggests lawyer consultation• Answers reference original clause for transparency

Functional Requirements

1. Landing Page & Information

FR-1 The system shall display a landing page describing the product benefits and features.
FR-2 The system shall show primary features such as Document Analyzer, Document Generator, and Connect (preview only).

2. User Registration & Login

FR-3 The system shall allow users to sign up using email/password.
FR-4 The system shall allow users to log in using existing credentials.

3. Basic Navigation

FR-5 The system shall provide navigation to the following pages:

- Home
- Document Analyzer
- Document Generator
- Connect

4. Call-to-Action Buttons

FR-6 The system shall provide “Sign Up” and “Sign In” buttons for onboarding.
FR-7 The system shall allow users to click “Create a Contract” (redirect only, no generation yet).
FR-8 The system shall allow users to click “Sign Document” (redirect only, no real signing yet).

5. Feature Preview Display

FR-9 The system shall visually show (static or mock) capabilities like:

- Clause simplification
- Risk alerts
- Smart document creation
- Secure e-signatures
- Document history timeline

Non-Functional Requirements

NFR-1 The homepage UI must load in under 5 seconds.
NFR-2 The UI must be clean, modern, and easy for new visitors to understand.
NFR-3 All navigation buttons must be responsive across desktop and mobile screen sizes.
NFR-4 Sign In / Sign Up pages must use secure HTTP protocol.
NFR-5 No features that are not yet developed should appear clickable without redirection messages.

User Stories

US-1 As a visitor, I want to understand what the platform does so that I can decide if it is useful to me.
US-2 As a visitor, I want to sign up or sign in so that I can access the platform.
US-3 As a visitor, I want to view all major features from the home page so that I know what the system can do.
US-4 As a visitor, I want to navigate between core sections (Analyzer / Generator / Connect) so that I can explore the product.

Use Cases

Use Case 1 — View Homepage

Actor: Visitor
Precondition: User opens the website
Flow:

1. System displays marketing header and feature overview
2. User reads summary of capabilities
3. User can continue to explore or register

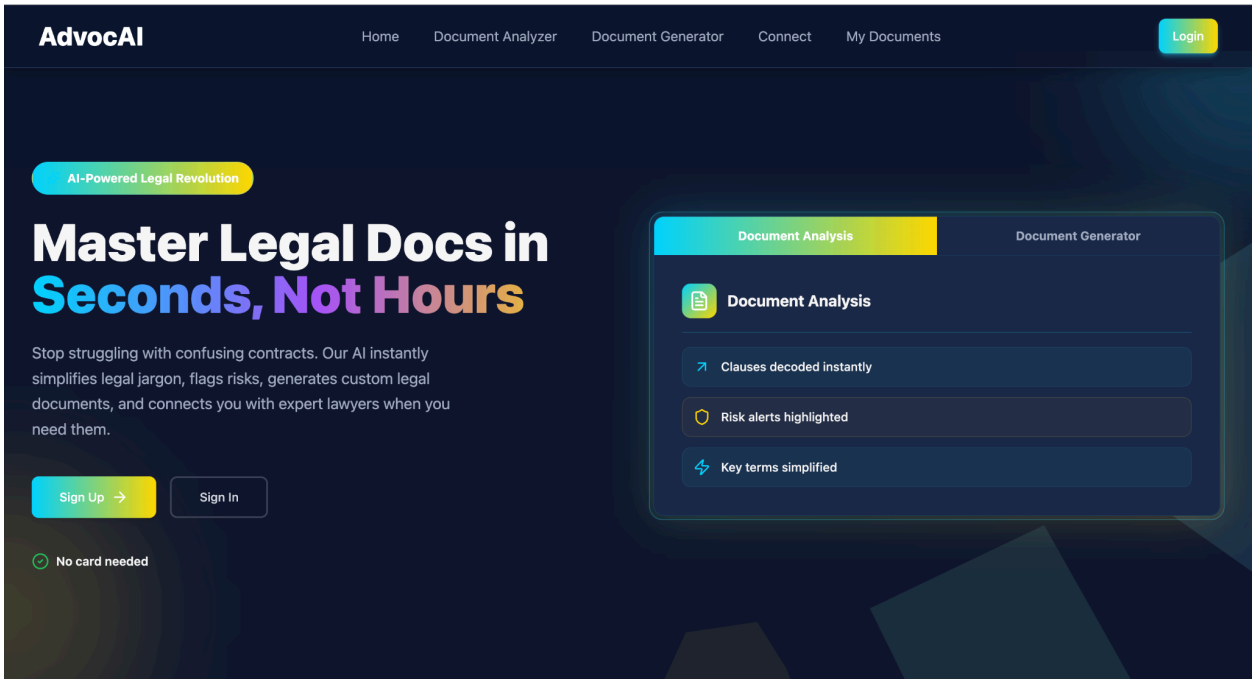
Use Case 2 — Navigate to Other Sections

Actor: Visitor
Precondition: User is on homepage
Flow:

1. User clicks navigation links (Document Analyzer / Document Generator / Connect)
2. System navigates to that screen or displays a preview page

Proof of Concept (POC)

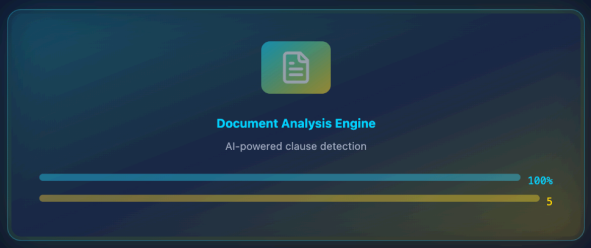
Use Case 1:



Core Capabilities

Powerful Features That Work Together

Every tool you need to master legal documents, from AI analysis to secure collaboration.



AI Document Analyzer

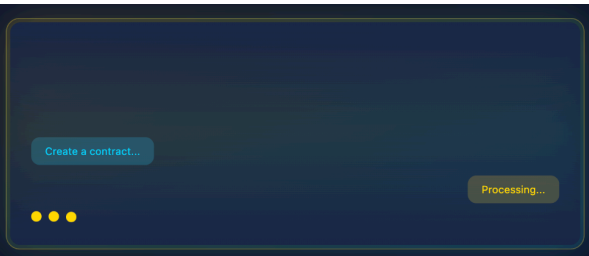
Transform complex legal documents into clear, actionable insights. Our advanced AI reads through pages of dense legal text and instantly highlights key terms, identifies risks, and provides plain-English summaries. Say goodbye to hours of frustrating document review.

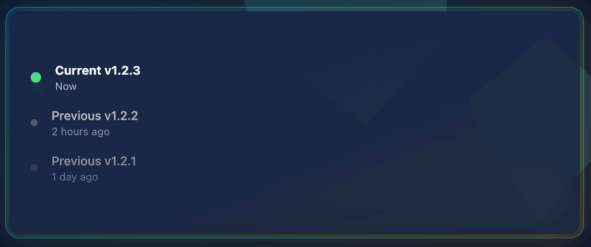
- Instant clause-by-clause breakdown
- Risk flagging and liability alerts
- Ask follow-up questions about any document

Smart Document Creator

Generate custom legal documents through an intuitive AI-powered chatbot. Describe exactly what you need, and our AI creates a tailored document. Edit, refine, and regenerate using natural conversation until your document is perfect.

- Conversational document generation
- Edit through natural chat interaction
- Download, edit, or regenerate instantly





Document History Timeline

Track every change made to your documents with a clear, chronological timeline. Access all previous versions of your documents with timestamps, see exactly what was modified, and instantly restore any earlier version with a single click.

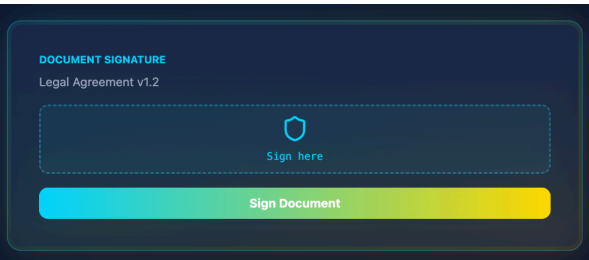
- Complete version history with timestamps
- Instantly restore any previous version
- Track changes made to your documents

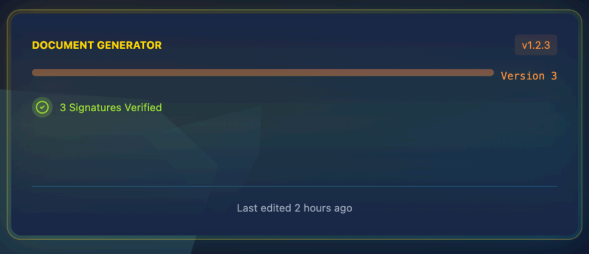
Secure E-Signature

Secure E-Signature

Sign documents with complete legal validity and security. Our e-signature solution is compliant with eIDAS, ESIGN Act, and other global regulations. Send signature requests to multiple parties, track completion status, and store signed documents securely.

- Legally binding digital signatures
- Multi-party signature workflows
- Bank-level encryption and audit trail





Generate, Track & Sign

Create custom documents with AI, track every version change in real-time, and collect legally binding e-signatures from multiple parties. Manage your entire document lifecycle in one platform.

- AI-powered custom document creation
- Complete version history with timestamps
- Legally binding e-signatures & verification
- Edit and regenerate with AI suggestions

Three Simple Steps to Mastery

From analysis to generation to signature in minutes

01

Analyze or Generate

Upload documents for instant AI analysis or create new ones from scratch with our intelligent generator.

02

Version & Edit

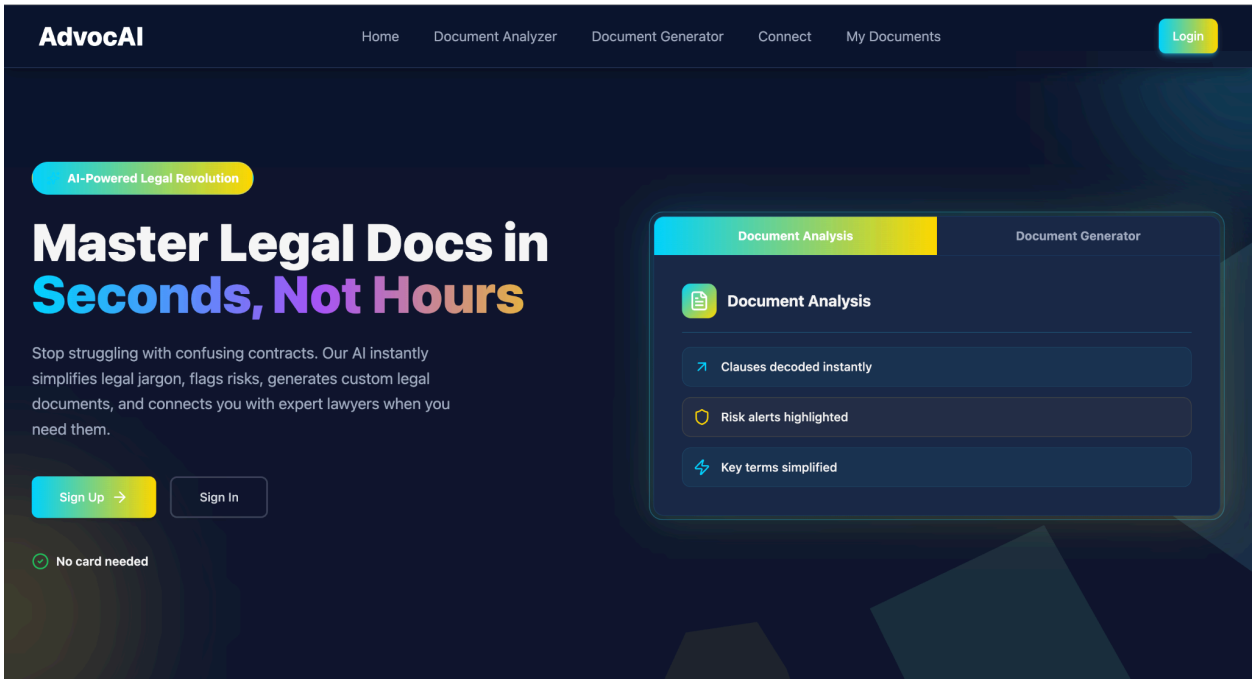
Make edits, request AI-powered changes, and track every version with complete revision history.

03

Sign & Deliver

Collect legally binding e-signatures from all parties with verification and secure audit trails.

Use Case 2:


AdvocAI
Home
Document Analyzer
Document Generator
Connect
My Documents
Login
AI-Powered Legal Revolution
Master Legal Docs in
Seconds, Not Hours
Stop struggling with confusing contracts. Our AI instantly simplifies legal jargon, flags risks, generates custom legal documents, and connects you with expert lawyers when you need them.
Sign Up
Sign In
No card needed
Document Analysis
Document Generator
Document Analysis
Clauses decoded instantly
Risk alerts highlighted
Key terms simplified

Sprint 2

FUNCTIONAL REQUIREMENTS

1. User Account & Access

FR-2.1 User Registration

The system shall enable users to create an account using basic details such as name, email, and password.

FR-2.2 User Login

The system shall allow registered users to securely log in using their credentials.

FR-2.3 User Logout

The system shall allow authenticated users to securely log out of their account.

2. Document Upload & Basic Summary

FR-2.4 Document Upload (PDF)

The system shall allow users to upload legal documents in PDF format.

FR-2.5 Basic Document Summary

The system shall generate a plain-English basic summary of the uploaded legal document.

3. Document Generation (Via AI)

FR-2.6 Prompt-Based Document Generation

The system shall allow users to request document creation by providing necessary details through input prompts.

FR-2.7 Template-Based Output

The system shall generate a simple legal document template (such as an NDA or rent agreement) based on the user's inputs.

NON FUNCTIONAL REQUIREMENTS

1. **NFR-1: Performance**
 • The system shall upload PDF documents within 5–10 seconds.
 • The system shall generate a basic summary or document template within 10 seconds.

2. **NFR-2: Security**
 • Only authenticated users shall be able to upload or generate documents.

3. **NFR-3: Usability**
 • The user interface shall be simple and easy for first-time users.
 • All buttons and forms shall be clearly labeled and understandable.

4. **NFR-4: Reliability**
 • The system shall remain stable and functional during demonstration.

USER STORIES

1. **US-2.1 User Authentication**

As a user, I want to sign up and log in so that I can securely access the system features.

Acceptance Criteria:

• User can create an account using email and password.
• User can log in with valid credentials.

• Unauthorized users cannot access upload or generation features.

2. **US-2.2 Basic Document Summary**
As a user, I want to upload a legal document and receive a simple summary so that I can understand the content easily.
Acceptance Criteria:
• User can upload a PDF document.
• System displays a basic English summary.
• Summary is relevant to the content of the uploaded document.

3. **US-2.3 Basic Document Generation**
As a user, I want to generate a basic legal document template by providing necessary details so that I do not have to draft it from scratch.
Acceptance Criteria:
• System provides basic legal templates (e.g., NDA).
• User can input key details (names, purpose, dates, etc.).
• System generates a formatted legal document template based on user inputs.

USE CASES

USE CASE 1: **USER AUTHENTICATION** (**SIGN UP** / **LOGIN**)
Actor: User
Precondition: User is not logged in
Postcondition: User is logged in and can access the system
Basic Flow:
1) User opens the website.
2) User selects “Sign Up” or “Login”.
3) System displays the authentication form.
4) User enters valid credentials.
5) System verifies credentials.
6) System redirects user to the dashboard.

Alternative Flow: Invalid credentials → System shows error message

USE CASE 2: **UPLOAD DOCUMENT FOR SUMMARY**
Actor: Logged-in User
Precondition: User must be logged in
Postcondition: A basic summary is displayed
Basic Flow:

1) User opens the Document Summary page.
2) User uploads a PDF document.
3) System processes the uploaded file.
4) System generates a plain-language summary.
5) System displays the summary to the user.

Alternative Flow: Unsupported file → System shows upload error

USE CASE 3: **GENERATE LEGAL DOCUMENT TEMPLATE**
Actor: Logged-in User
Precondition: User must be logged in
Postcondition: A basic legal document template is displayed
Basic Flow:

1) User navigates to the Document Generation page.
2) User selects a document type (e.g., NDA).
3) User provides required inputs.
4) System processes the user details.
5) System displays the generated document template.

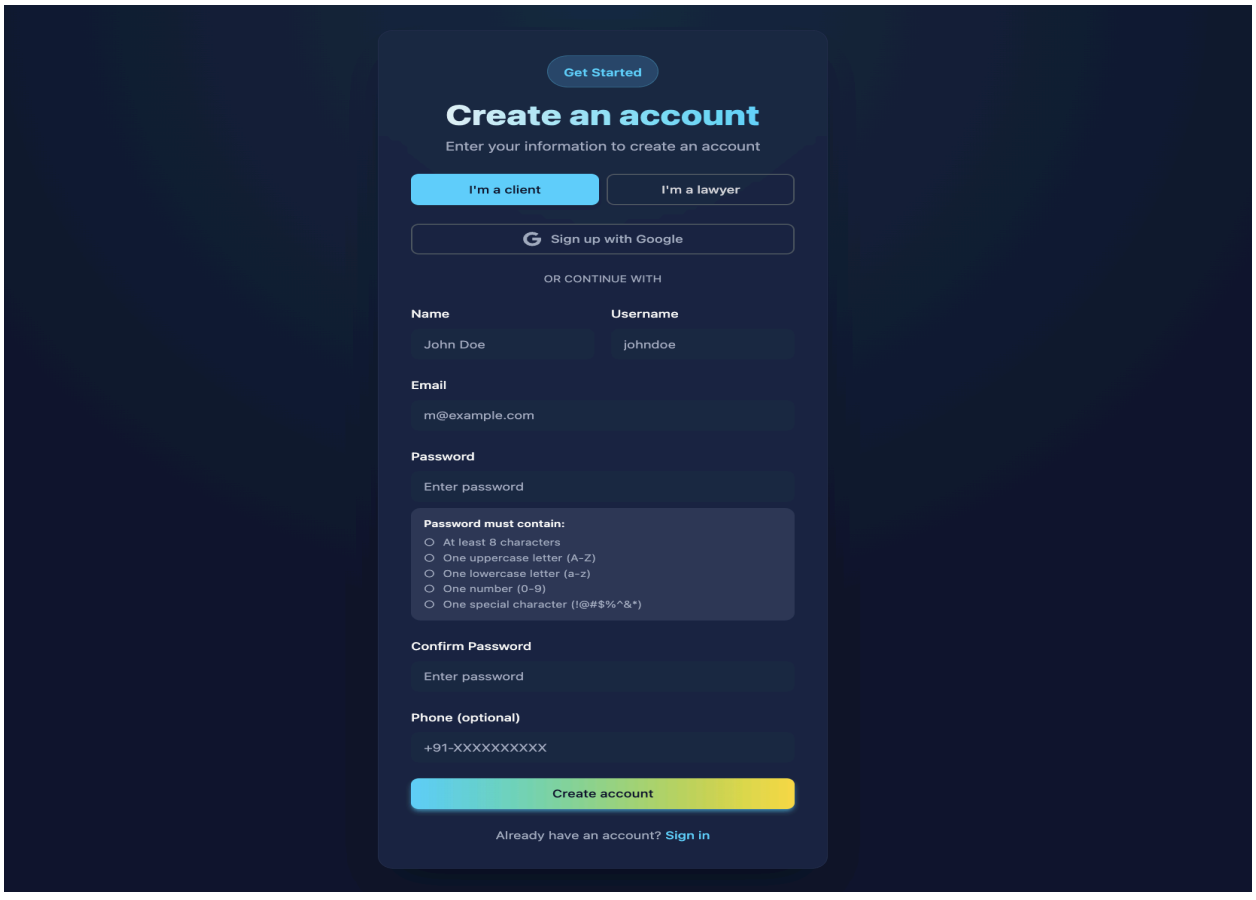
Alternative Flow: Missing required information → System requests corrections

Proof of Concept (POC)

1) Auth Flow

Sign-Up Flow – Client

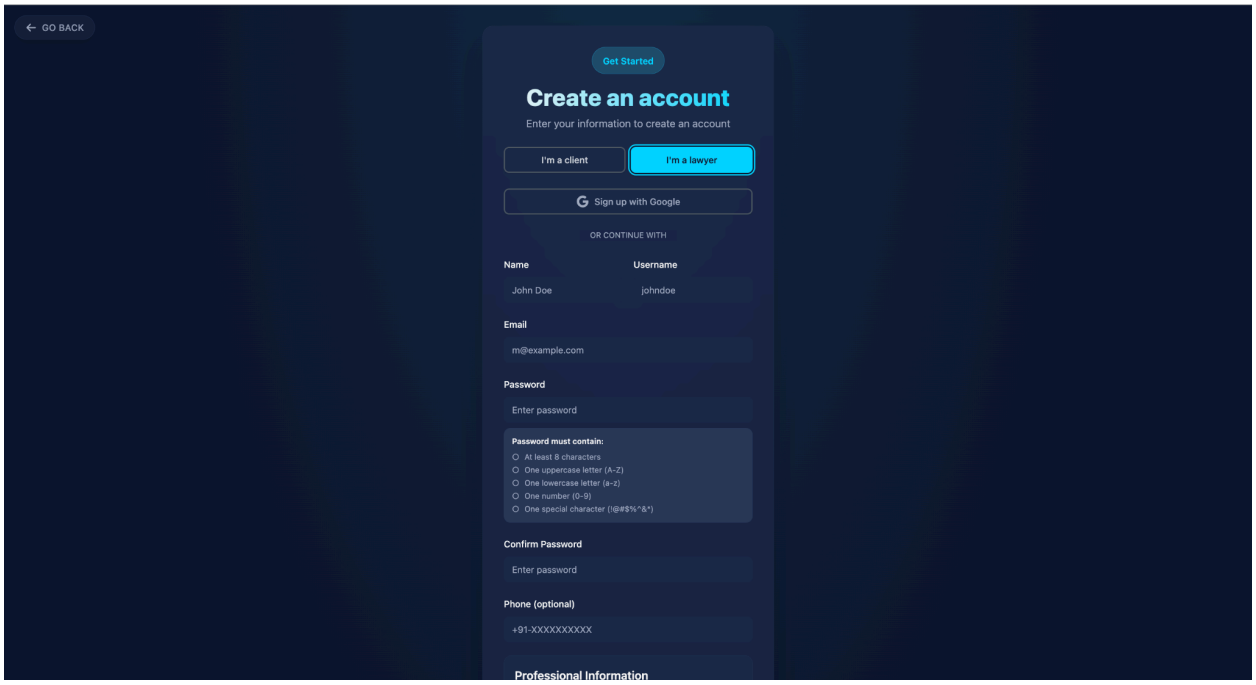
1) User selects “Create Account”.
2) User selects “I’m a Client”.
3) System displays a registration form.
4) User enters name, username, email, and password.
5) System displays password strength validation.
6) User submits the form.
7) Account is created and system redirects user to the login page.

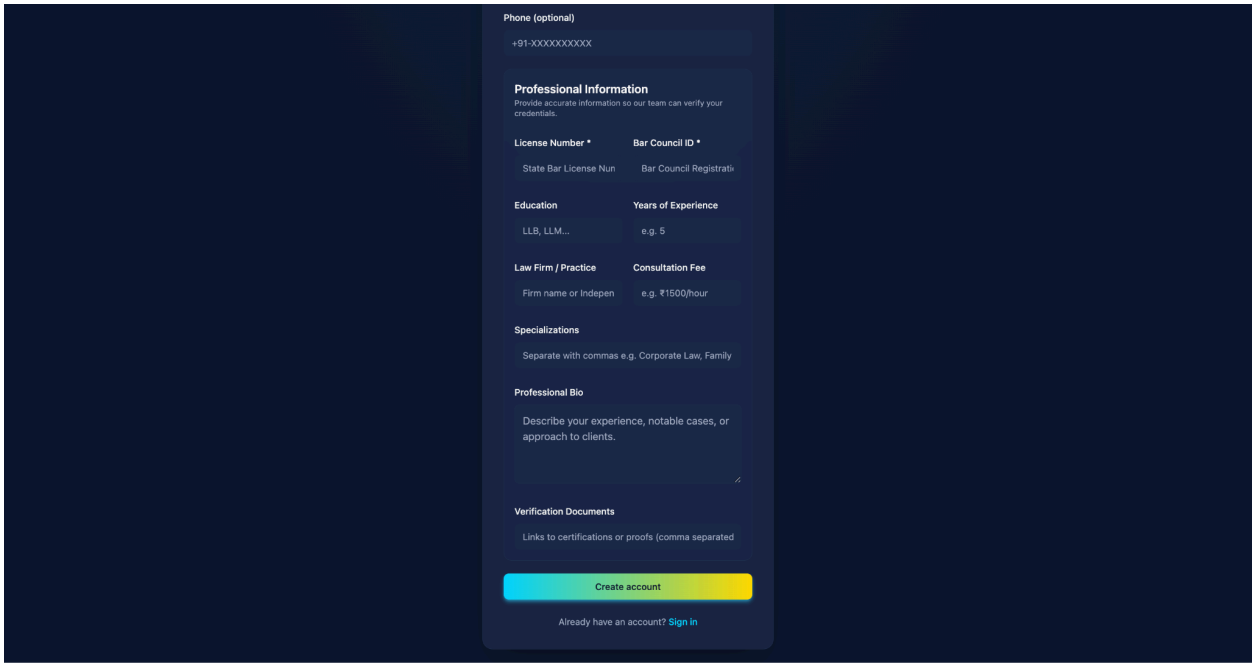


Sign-Up Flow – Lawyer

1) User selects “Create Account”.
2) User selects “I’m a Lawyer”.

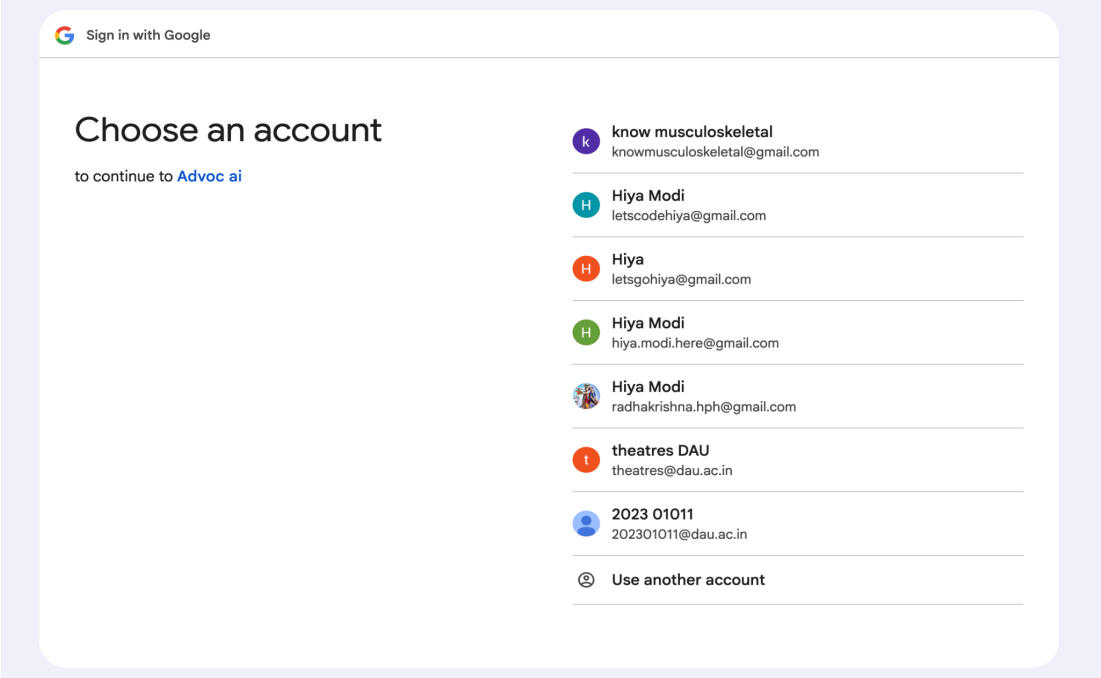
3) System displays additional required fields:
License Number, Bar Council ID, Experience, Specialization, Consultation Fee, Professional Bio, Verification Documents.
4) User submits the form with mandatory information.
5) System securely stores lawyer data for verification.





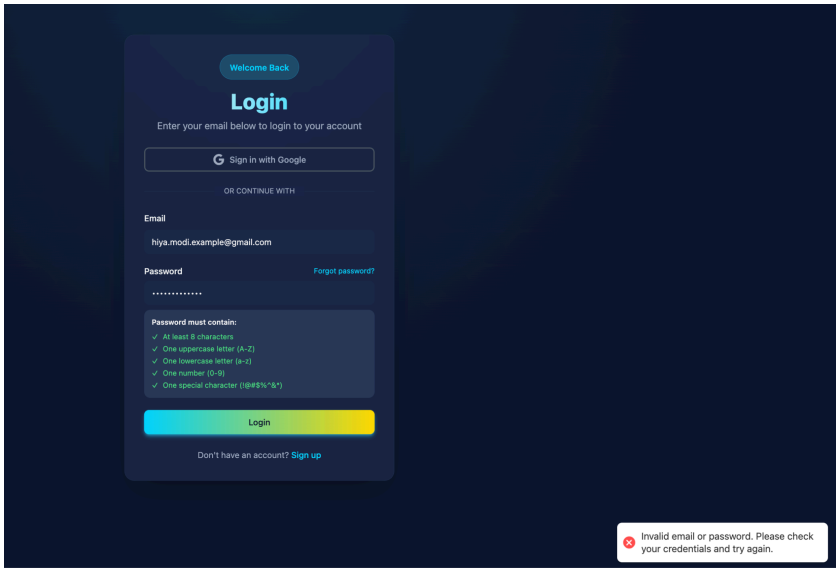
Google Sign-In Flow

1) User selects “Sign up with Google”.
2) Google account selection screen appears.
3) User grants permission for authentication.
4) System logs in the user and redirects to dashboard.



Login Flow

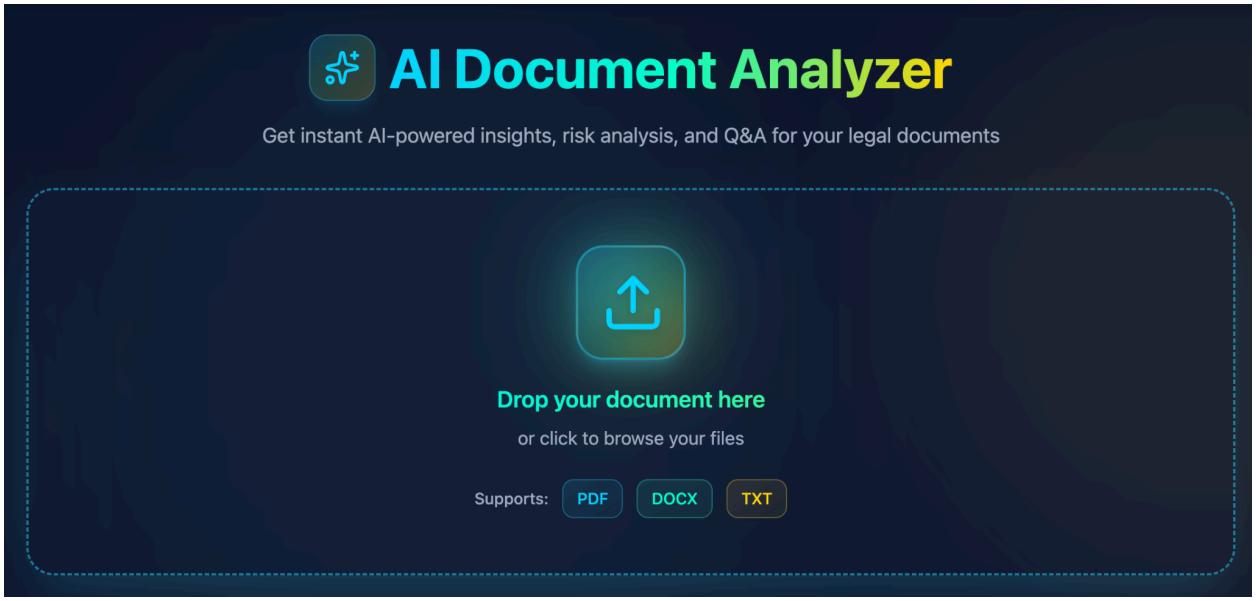
1) User enters registered email and password.
2) System validates the entered credentials.
3) If valid, user is redirected to dashboard.
4) If invalid, system displays an authentication error message.



2) Document Analyzer

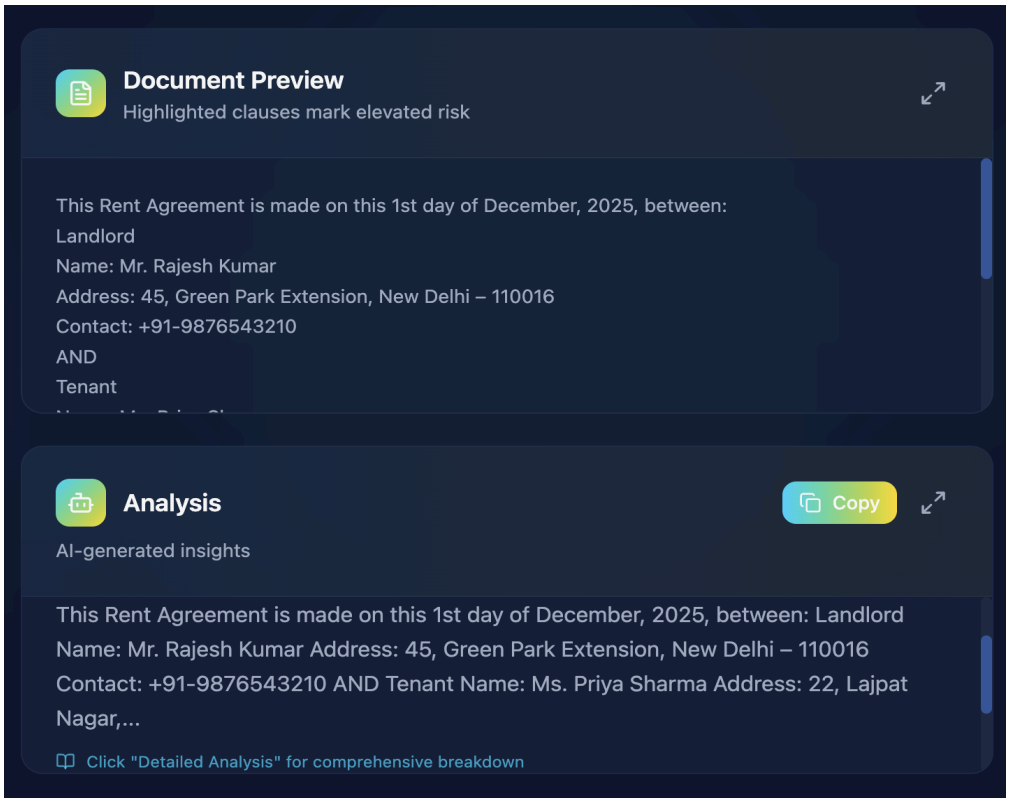
Upload Screen

1) User navigates to AI Document Analyzer.
2) User uploads a legal document (PDF, DOCX, TXT).
3) System extracts document content and analyzes clauses.



Basic Analysis Output

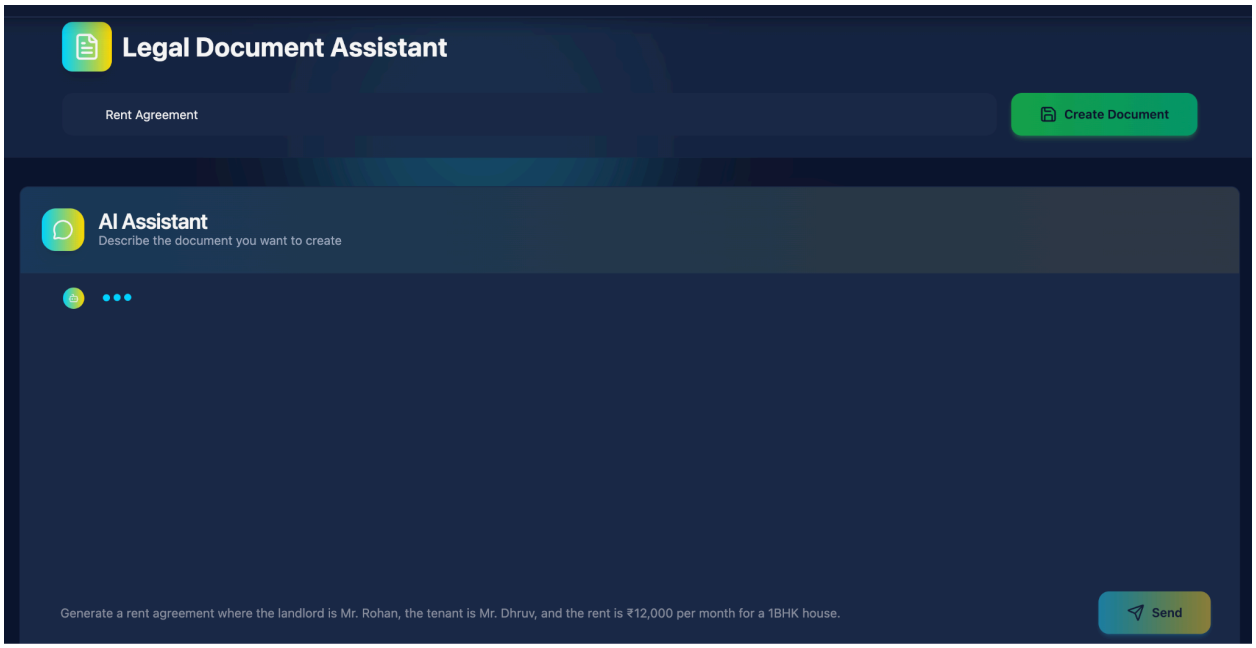
1) System displays document preview.
2) AI generates a plain-language summary and insights.
3) User can copy or explore the detailed analysis.



3) Document Generator

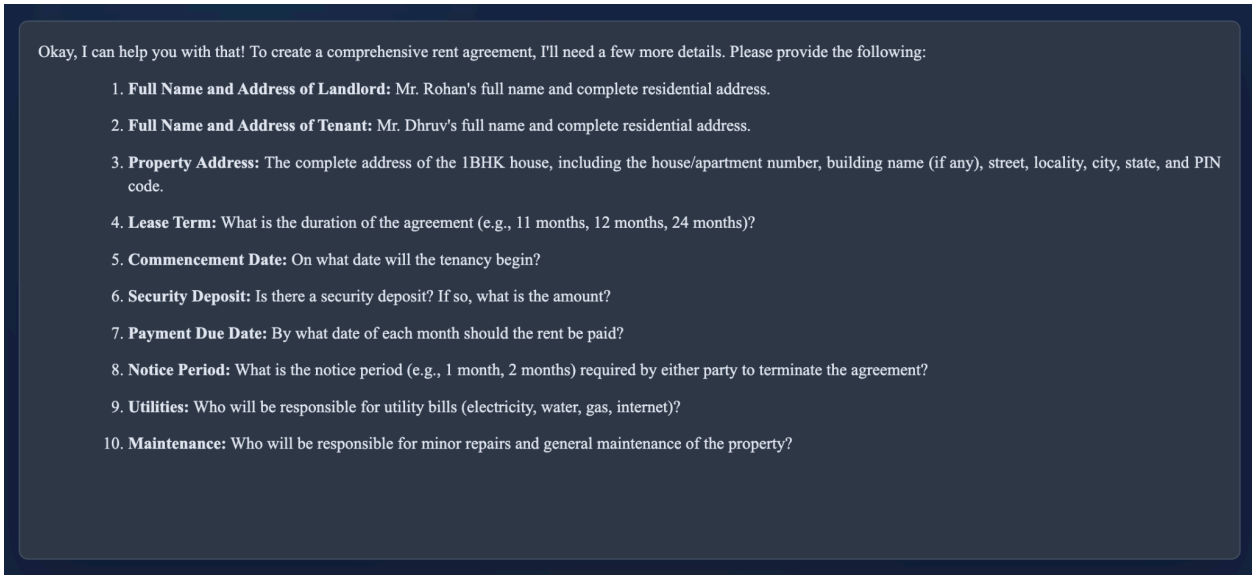
Prompt Based Input

1) User opens the Legal Document Assistant screen.
2) User selects the legal document type (e.g., Rent Agreement).
3) User inputs basic case details through a guided prompt.

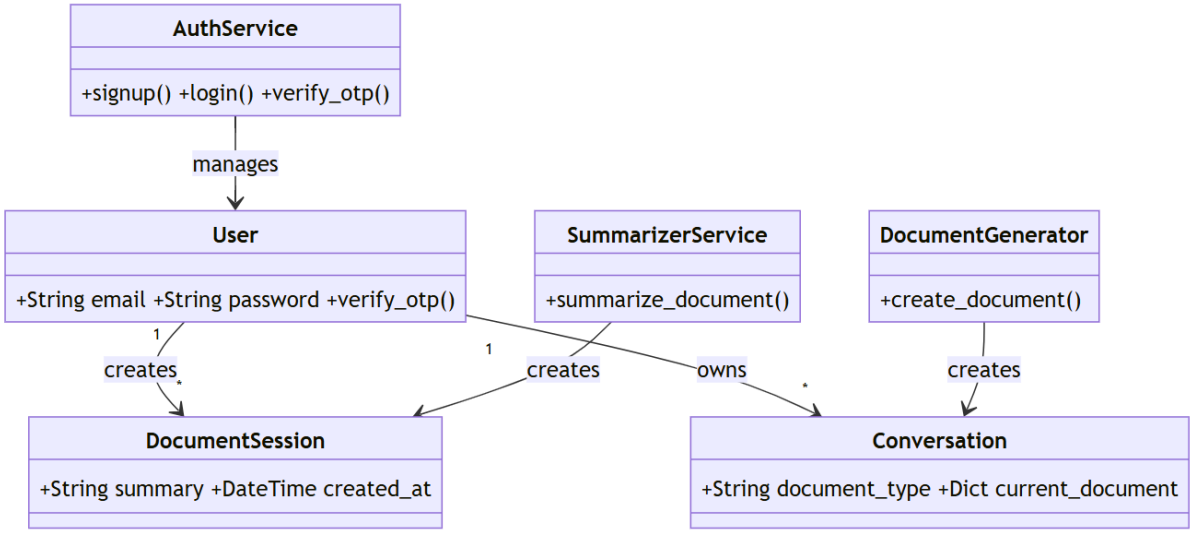


AI Template Customization

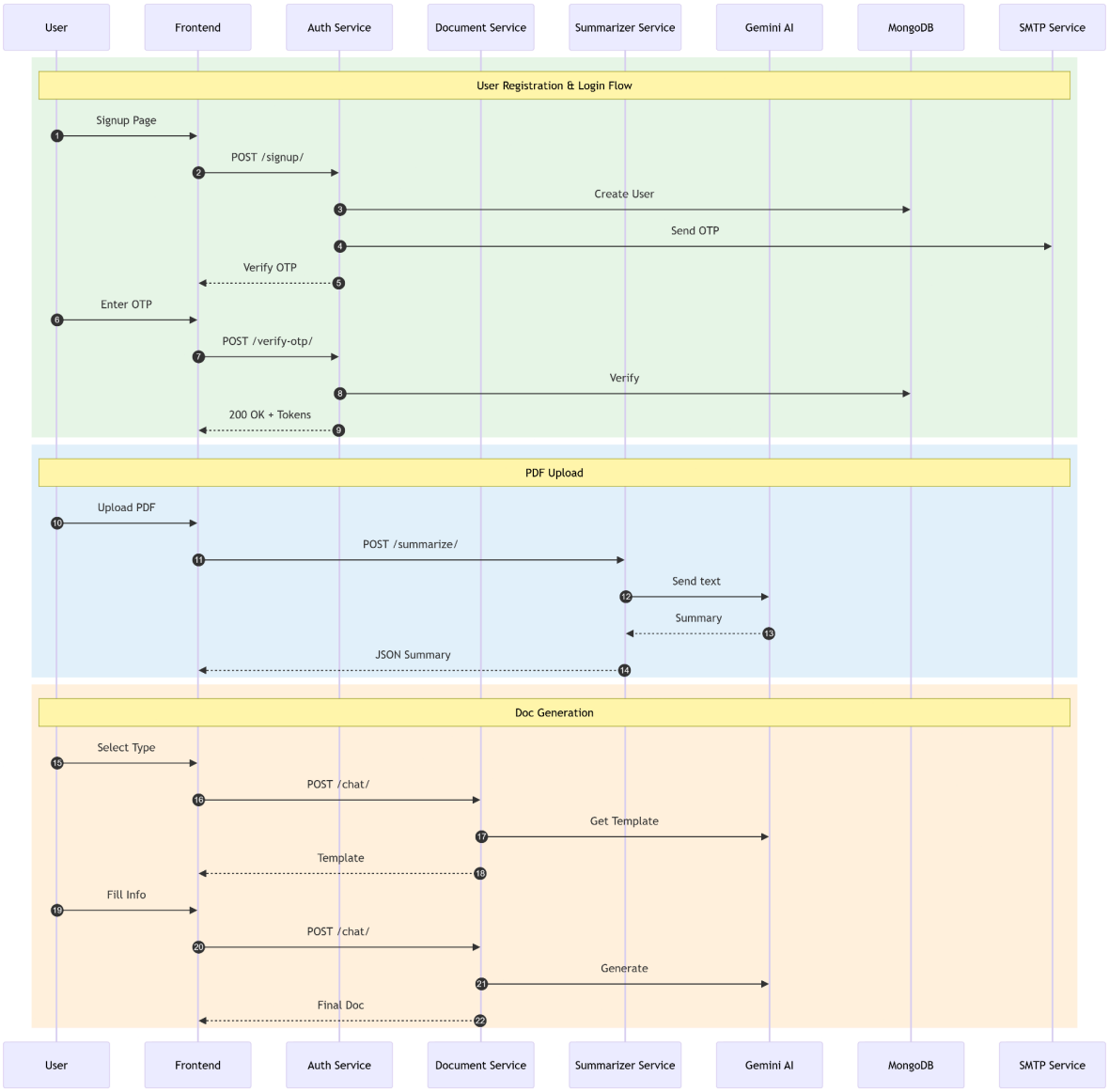
1) AI asks for additional details (names, addresses, rent, duration, etc.).
2) System ensures required clause information is collected.
3) System generates a personalized legal document draft.



CLASS DIAGRAM



SEQUENCE DIAGRAM



Sprint 3

FUNCTIONAL REQUIREMENTS

1. **AI Active Document Support**

FR-3.1 The system shall explain complex legal terms in simple and understandable language.

FR-3.2 The system shall allow users to ask follow-up questions for clause-based clarification.

FR-3.3 The system shall provide contextual AI responses using improved backend intelligence (LangChain).

2. **Custom Document Generation**

FR-3.4 The system shall customize standard legal templates based on user-entered case details.

FR-3.5 The system shall allow users to review the generated customized legal document.

3. **Lawyer Connect Prototype**

FR-3.6 The system shall provide basic visibility of lawyer profiles for users.

FR-3.7 The system shall allow users to send a connection request to lawyers as a prototype feature.

NON-FUNCTIONAL REQUIREMENTS

1. **Performance**

NFR-3.1 The system shall provide quick AI responses for legal term explanations and clarifications.

NFR-3.2 The system shall process customized template generation within a reasonable time for demo purposes.

2. **Accuracy & Reliability**

NFR-3.3 The chatbot shall provide logically consistent and context-aware clarifications for uploaded/generated content.

NFR-3.4 The system shall leverage LangChain to reduce hallucinations and improve accuracy in responses.

3. **Usability**

NFR-3.5 The AI explanations shall be easy to understand even for non-legal users.

NFR-3.6 The UI shall maintain a simple and intuitive interaction flow for asking clarifying questions and generating documents.

4. **Security**

NFR-3.7 User authentication shall remain mandatory for using AI document operations.

NFR-3.8 All user inputs and generated document details shall be securely handled.

5. **Transparency & Trust**

NFR-3.9 The system shall inform the user that the responses are AI-generated and may require legal expert review in real use cases.

USER STORIES

US-3.1 Document Summary Explanation

As a user, I want complex legal terms to be explained in simple language so that I can easily understand the meaning of different clauses.

Acceptance Criteria:

- System explains highlighted terms in easy-to-understand text.
- User can request clause-specific clarification through the AI chatbot.

US-3.2 Chat-Based Clarification

As a user, I want to ask follow-up questions to clarify any part of the document so that I understand my rights and obligations clearly.

Acceptance Criteria:

- System allows users to enter queries related to document content.

• AI provides contextual responses based on specific clauses.

US-3.3 Customized Template Generation

As a user, I want the legal document generator to customize templates based on my details so that I do not have to create documents from scratch.

Acceptance Criteria:

• System accepts user input (e.g., names, dates, property details).

• System inserts user details into standard templates.

• A customized document draft is generated.

US-3.4 Lawyer Connect Prototype

As a user, I want to see available lawyer profiles so that I can decide whom to reach out to for advice when needed.

Acceptance Criteria:

• System displays basic lawyer details (name, specialization).

• User can express interest or request connection as a demo feature.

US-3.5 AI Response Quality Enhancement

As a user, I want the chatbot to provide more accurate and relevant responses so that I can trust the legal guidance given.

Acceptance Criteria:

• System uses improved backend models (LangChain) to minimize errors.

• Responses are context-aware and consistent with user queries.

USE CASES

USE CASE 1: Clause-Based AI Chat Clarification

Actor: Logged-in User

Precondition: User has accessed summary or term explanation

Postcondition: User receives chatbot response based on specific

clause/query
Basic Flow:

1) User enters a chat query related to a clause or term.
2) System interprets the query and retrieves clause context.
3) System provides clarification through conversational response.
4) User may continue asking follow-up questions.

USE CASE 2: Customized Legal Document Generation & Chat

Actor: Logged-in User
Precondition: User provides required input details (parties, dates, etc.)
Postcondition: Customized legal document template is generated
Basic Flow:

1) User selects required legal document type (e.g., rent agreement).
2) User fills required case information.
3) System applies user data into the standard template.
4) System displays the customized legal draft.
5) System displays chat option

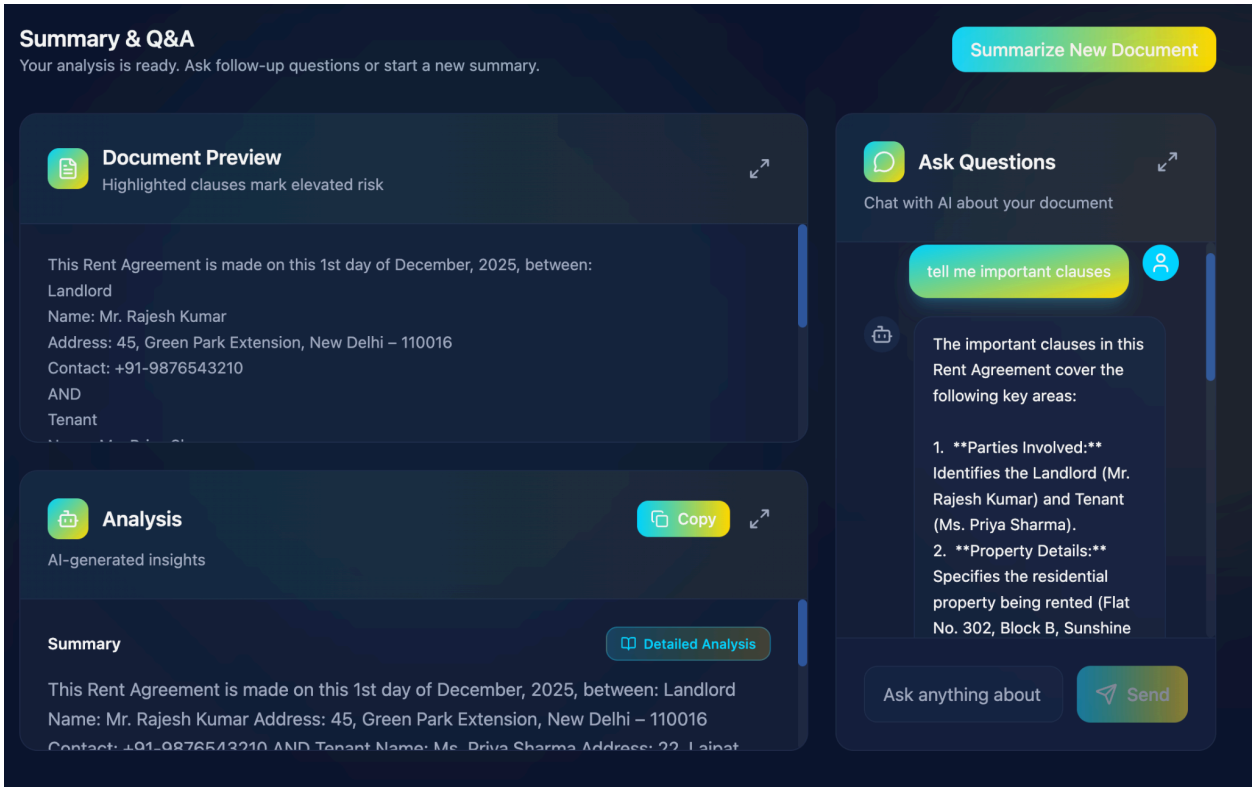
USE CASE 3: Lawyer Connect Prototype

Actor: Logged-in User
Precondition: User is logged into the system
Postcondition: User sends request/interest for lawyer connection
Basic Flow:

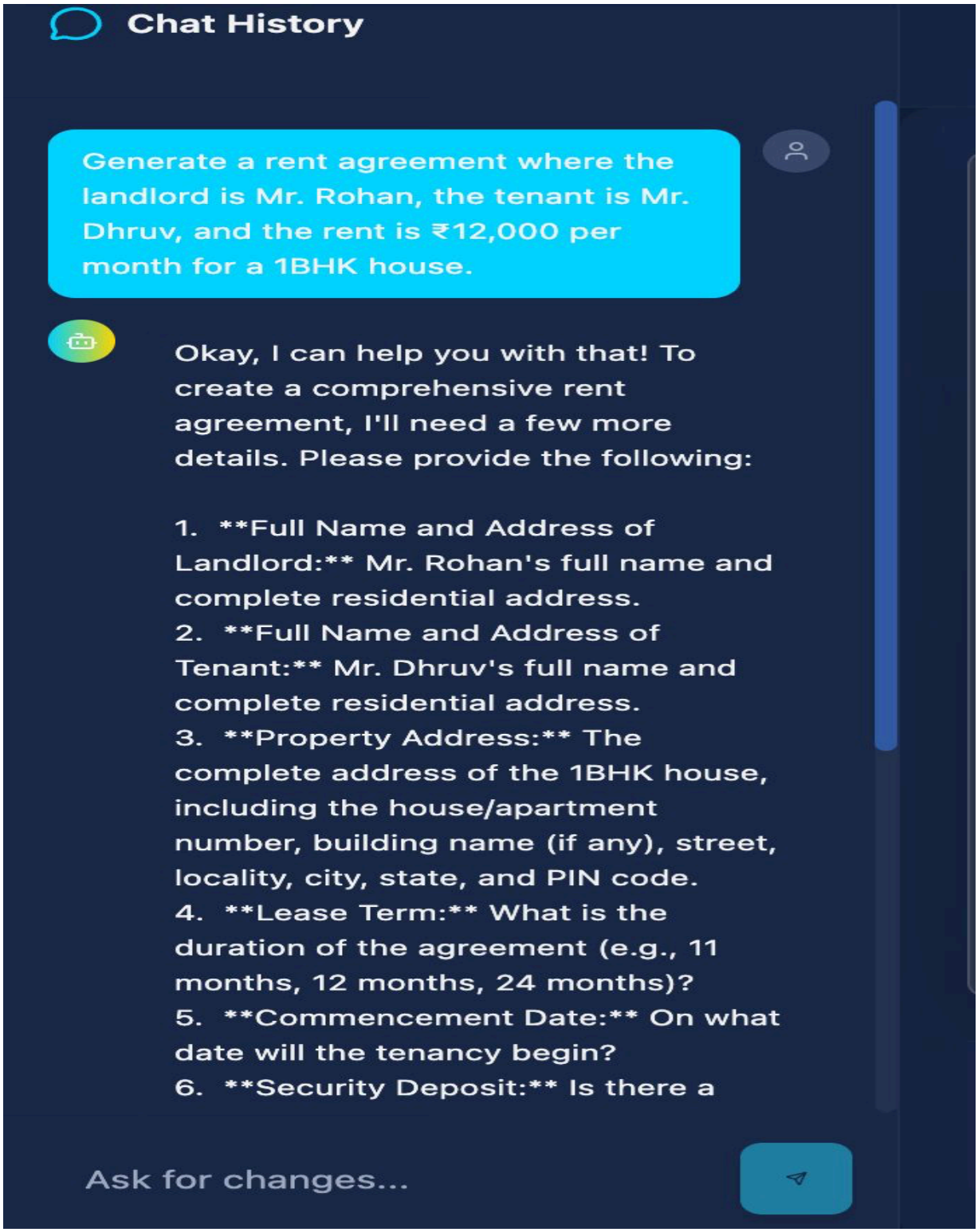
1) User opens list of available lawyers.
2) System displays basic lawyer information such as specialization.
3) User selects a lawyer profile.
4) User sends a connection/interest request (prototype stage).

Proof of Concept (POC)

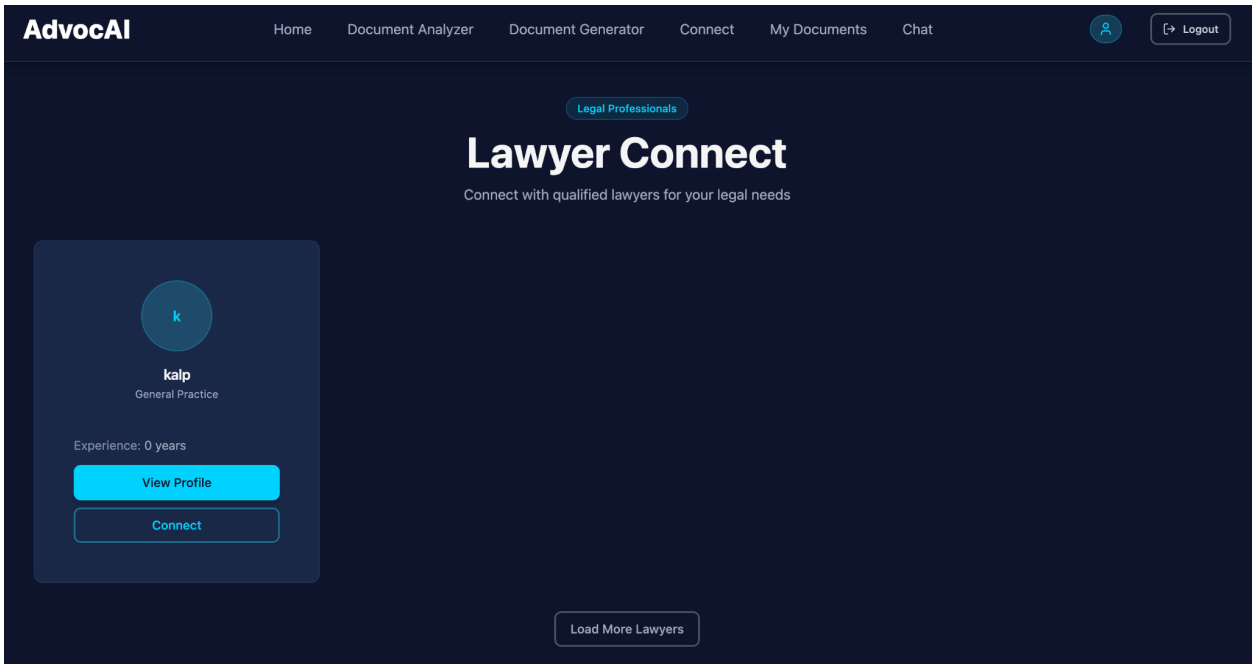
Use Case 1:



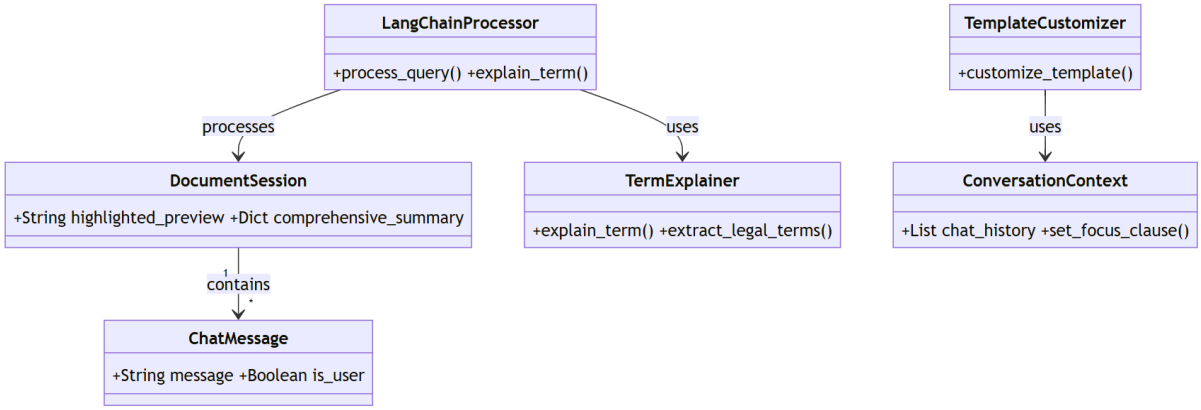
Use Case 2:


Chat History
Generate a rent agreement where the landlord is Mr. Rohan, the tenant is Mr. Dhruv, and the rent is ₹12,000 per month for a 1BHK house.
Okay, I can help you with that! To create a comprehensive rent agreement, I'll need a few more details. Please provide the following:
1. **Full Name and Address of Landlord:** Mr. Rohan's full name and complete residential address.
2. **Full Name and Address of Tenant:** Mr. Dhruv's full name and complete residential address.
3. **Property Address:** The complete address of the 1BHK house, including the house/apartment number, building name (if any), street, locality, city, state, and PIN code.
4. **Lease Term:** What is the duration of the agreement (e.g., 11 months, 12 months, 24 months)?
5. **Commencement Date:** On what date will the tenancy begin?
6. **Security Deposit:** Is there a
Ask for changes...

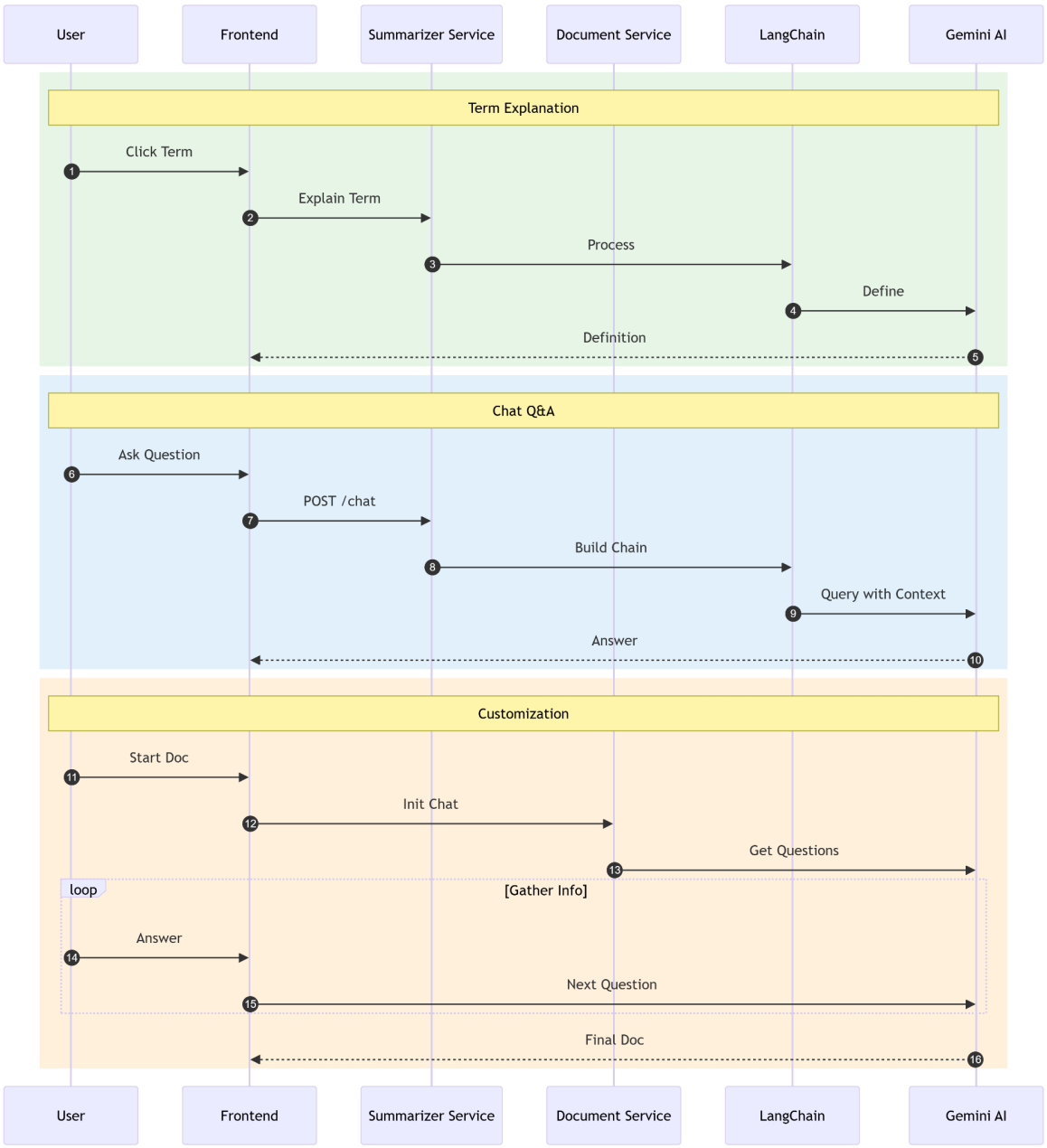
Use Case 3:



CLASS DIAGRAM



SEQUENCE DIAGRAMS



Sprint 4

FUNCTIONAL REQUIREMENTS

1. Document Analysis & High-Risk Clause Detection

FR-4.1 The system shall identify and highlight high-risk or unclear clauses in the uploaded document.

FR-4.2 The system shall provide explanations or guidance about the detected high-risk clauses to the user.

2. Version Control & Rollback

FR-4.3 The system shall save each edited version of a document separately.

FR-4.4 The system shall allow users to roll back to any previous document version on request.

3. E-Signature Support

FR-4.5 The system shall allow users to request e-signature on generated documents.

FR-4.6 The system shall mark the signed document appropriately to indicate signature completion.

4. AI Assistance Improvement (LangChain)

FR-4.7 The system shall enhance the AI summarizer accuracy using LangChain for improved clause detection.

FR-4.8 The system shall generate more relevant and context-aware summaries with fewer hallucinations.

NON-FUNCTIONAL REQUIREMENTS

1. **Accuracy & Reliability**

NFR-4.1 The system shall accurately detect and highlight high-risk clauses in legal documents.

NFR-4.2 The system shall reduce hallucinations and improve reliability of clause analysis using LangChain.

2. **Performance**

NFR-4.3 High-risk clause detection shall be completed within a reasonable time for user interaction.

NFR-4.4 Version rollback actions shall execute without noticeable system delay.

3. **Security**

NFR-4.5 The system shall ensure secure handling of signatures applied to legal documents.

NFR-4.6 All version history shall be stored securely and protected from unauthorized access.

4. **Usability**

NFR-4.7 The version control interface shall be easy for users to understand and operate.

NFR-4.8 High-risk clause warnings shall be clearly visible and understandable to non-legal users.

5. **Transparency**

NFR-4.9 The system shall clearly show whether a clause is flagged as risky and provide a reason for flagging.

NFR-4.10 The system shall notify users that e-signatures applied are AI-assisted and may need professional validation for legal compliance.

USER STORIES

US-4.1 High-Risk Clause Detection

As a user, I want the system to highlight high-risk clauses in my legal document so that I can avoid legal mistakes or complications.

Acceptance Criteria:

- System analyzes document content for potentially risky clauses.
- System highlights risky parts and explains why they are risky.
- System allows the user to review highlighted content clearly.

US-4.2 Version Tracking & Rollback

As a user, I want every document update to be stored as a separate version so that I can restore a previous version anytime if needed.

Acceptance Criteria:

- Each document edit creates a separate saved version.
- System shows version history to the user.
- User can roll back to any earlier version.

US-4.3 E-Signature Support

As a user, I want to request e-signing of generated documents so that I can finalize agreements digitally without printing.

Acceptance Criteria:

- System provides an option to request digital e-signature.
- System marks the document after signing for confirmation.
- Signed copy remains securely stored in the system.

US-4.4 Improved AI Quality (LangChain)

As a user, I want more accurate summarization and explanations so that I can rely on the system responses.

Acceptance Criteria:

- System uses enhanced AI (LangChain) for deeper clause analysis.
- Reduced hallucination and improved correctness in output.
- Explanations are context-aware and legally meaningful.

USE CASE

USE CASE 1: HIGH-RISK CLAUSE DETECTION

Actor: Logged-in User

Precondition: A legal document has been uploaded and processed by the system.

Postcondition: High-risk clauses in the document are highlighted and explained to the user.

Basic Flow:

1) User opens the analyzed document view.
2) User clicks on the option to "Highlight High-Risk Clauses".
3) System scans the document clauses using the AI/ LangChain backend.
4) System highlights clauses detected as high-risk.
5) System displays a short explanation for each high-risk clause.

Alternative Flow:

• If no high-risk clauses are found -> System informs the user that no risky clauses were detected.

USE CASE 2: VERSION TRACKING AND ROLLBACK

Actor: Logged-in User

Precondition: At least one document has been edited and saved multiple times.

Postcondition: User either views version history or restores a previous version of the document.

Basic Flow:

1) User opens a document from their document list.
2) User selects the "Version History" option.
3) System displays a list of saved versions with basic metadata (e.g., time of save).
4) User selects a specific version to view.

5) System shows the selected version content to the user.

6) User chooses "Restore this Version".

7) System restores the selected version as the current active version.

Alternative Flow:

• If no previous versions exist -> System shows a message that only the current version is available.

USE CASE 3: REQUEST E-SIGNATURE ON DOCUMENT

Actor: Logged-in User

Precondition: A final version of a generated legal document is available.

Postcondition: E-signature is requested and the document is marked as signature-pending or signed (prototype-level).

Basic Flow:

1) User opens a generated legal document.

2) User selects the option "Request E-Signature".

3) System asks the user to confirm the document for signing.

4) User confirms the request.

5) System updates the document status to indicate that e-signing has been requested.

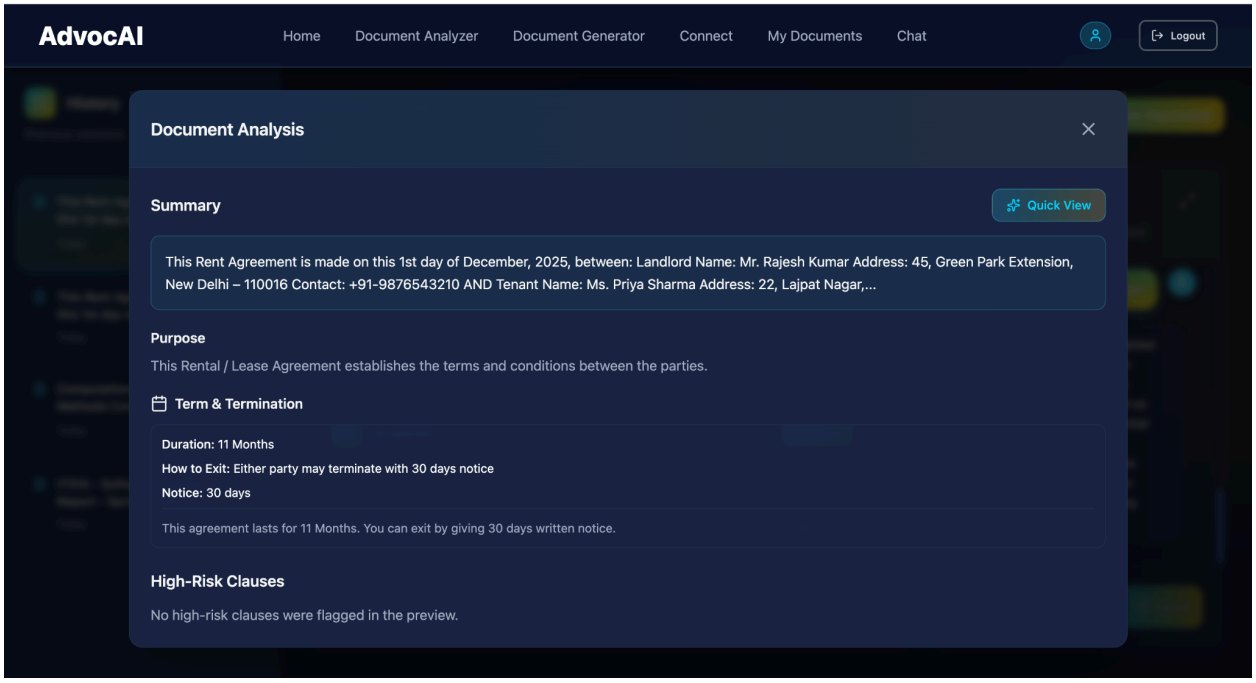
6) System marks the document visually (e.g., tag or label) to show that it is in e-signature flow or signed (as per prototype).

Alternative Flow:

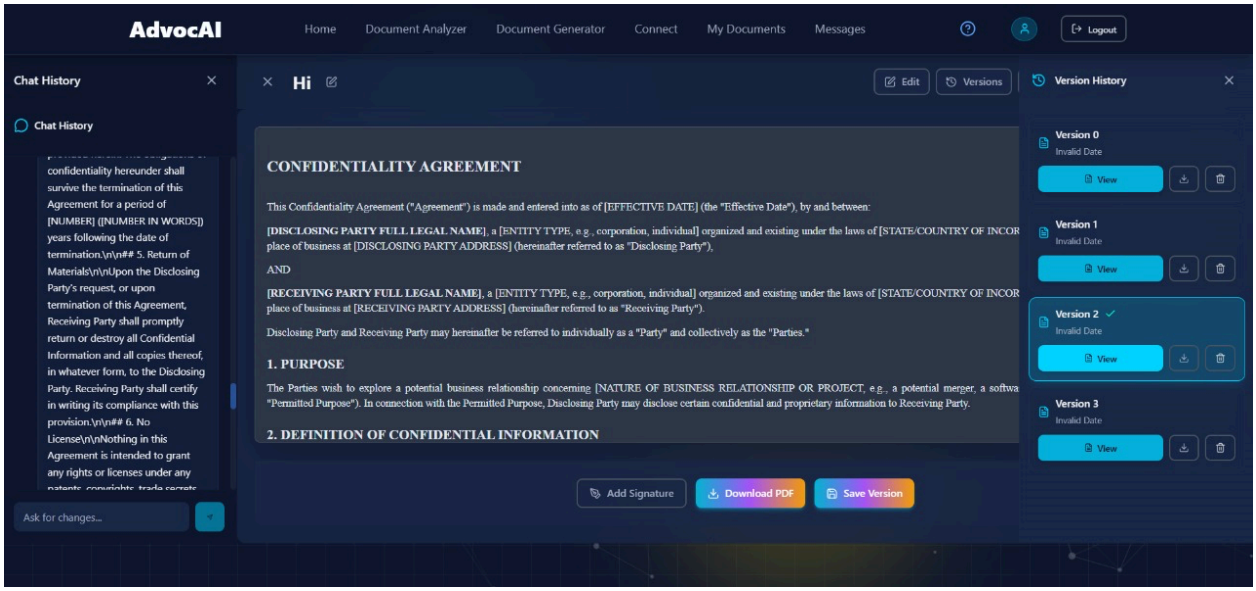
• If the document is not eligible for e-signature in prototype -> System shows a message explaining that e-signature is not available for this document.

Proof of Concept (POC)

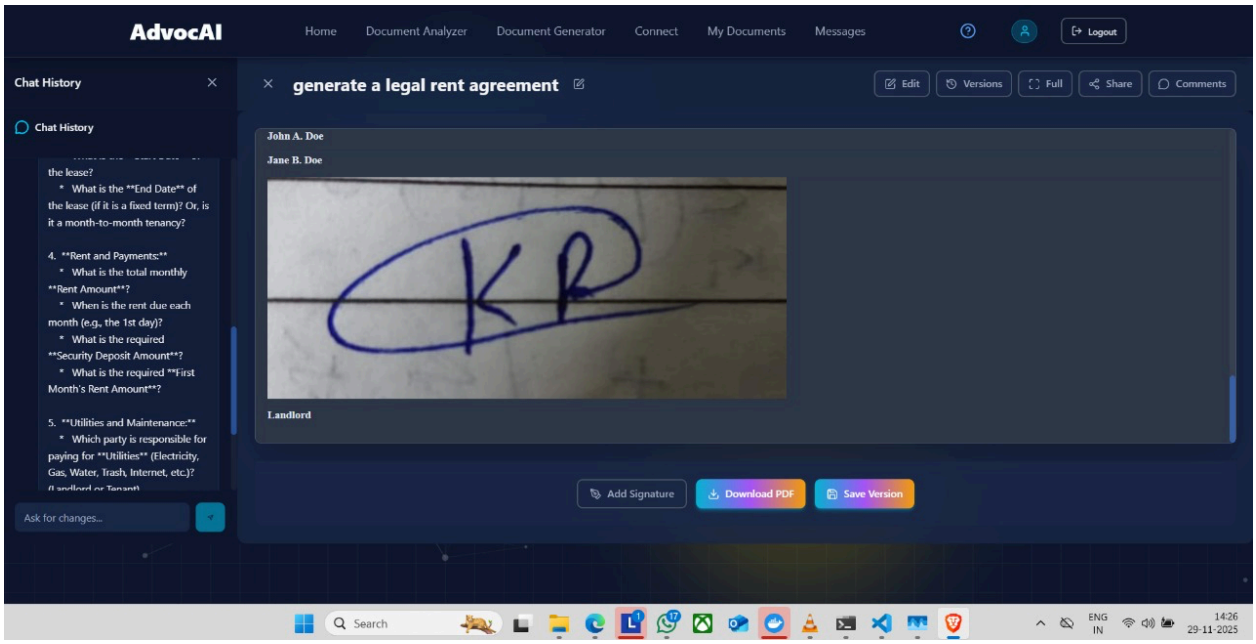
Use Case 1:



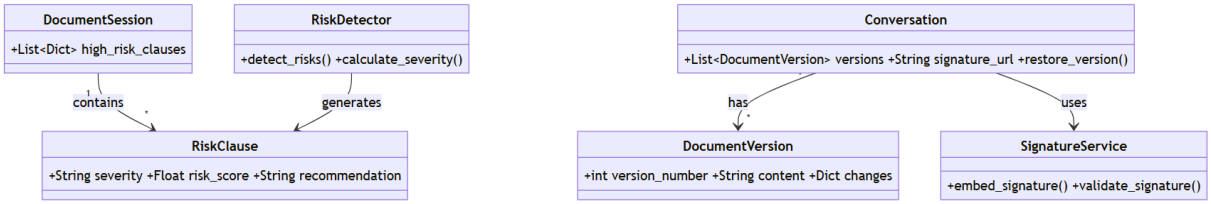
Use Case 2:



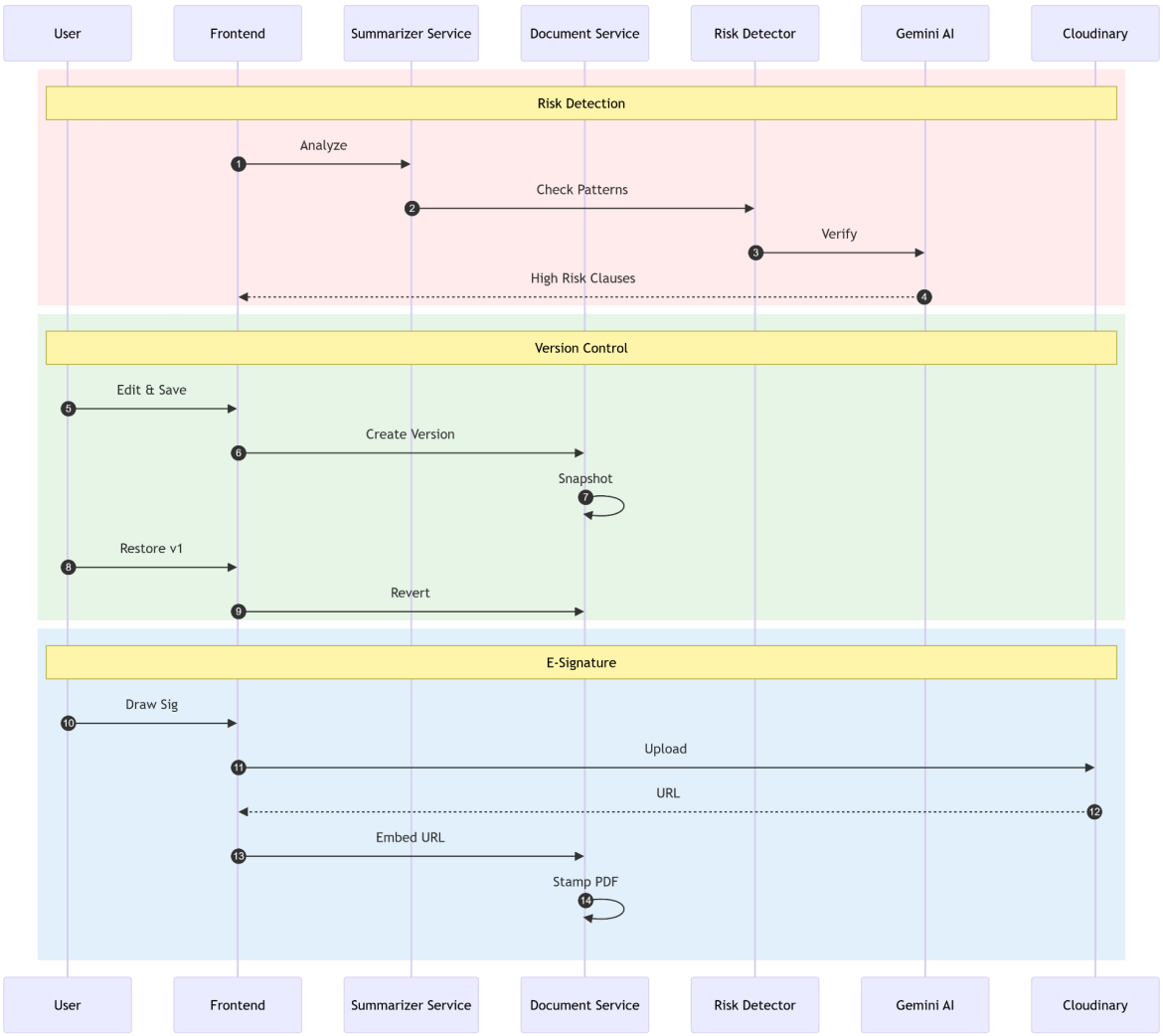
Use Case 3:



CLASS DIAGRAM



SEQUENCE DIAGRAM



Sprint 5

FUNCTIONAL REQUIREMENTS

1. **Collaborative Commenting**

FR-5.1 The system shall allow multiple users to add comments and suggestions on a document.

FR-5.2 The system shall allow all collaborators to view and respond to each other's comments.

FR-5.3 The system shall update the document with suggested changes only after approval by all involved parties.

2. Lawyer Connect Advancement

FR-5.4 The system shall allow users to request a lawyer consultation through the platform.

FR-5.5 The system shall allow lawyers to view consultation requests including basic case details and documents.

FR-5.6 The system shall allow lawyers to accept or reject the consultation request.

FR-5.7 The system shall escalate the request into further communication when a lawyer accepts the case.

3. Collaboration Behavior Rules

FR-5.8 The system shall notify all collaborators when new comments or suggestions are added.

FR-5.9 The system shall track who made which comment and at what time.

NON-FUNCTIONAL REQUIREMENTS

1. Reliability & Accuracy

NFR-5.1 System shall correctly sync comments across all collaborators.

NFR-5.2 Lawyer decisions (accept/reject) shall update instantly without conflicts.

2. Usability

NFR-5.3 The commenting interface shall be intuitive and simple to use.

NFR-5.4 All users shall clearly see which comments require approval.

3. Performance

NFR-5.5 Adding or approving comments shall reflect within a reasonable response time.

NFR-5.6 Lawyer actions (view, accept, reject) shall display results without delay.

4. Security

NFR-5.7 Only authorized parties shall access document and comments.

NFR-5.8 Lawyer access to user documents must be permitted explicitly by the user.

5. Transparency

NFR-5.9 System shall show the identity of comment authors and lawyer decisions clearly.

NFR-5.10 Users shall be informed when consultation has escalated with a lawyer.

USER STORIES

US-5.1 Collaborative Commenting

As a stakeholder, I want to add suggestions and comments on documents so that multiple parties can collaborate before finalizing the agreement.

Acceptance Criteria:

- Multi-party comments allowed
- All users can view/respond to suggestions
- Approved suggestions update the document

US-5.2 Review & Approve Changes

As a collaborator, I want to approve or reject suggestions so that only agreed-upon changes are applied.

Acceptance Criteria:

- Comments have approve/reject option
- Document updates after full approval
- Notification sent to all parties

US-5.3 Lawyer Consultation Support

As a user, I want to request consultation with a lawyer so that I can receive legal advice on my case.

Acceptance Criteria:

- User can send consultation request
- Lawyer sees document + case details
- Lawyer accepts/rejects consultation

US-5.4 Lawyer Case Handling

As a lawyer, I want to view request details and take action so that I can manage clients efficiently.

Acceptance Criteria:

- Lawyer dashboard with new requests
- Lawyer decision logged and reflected to user
- Accepted cases escalate to communication flow

USE CASES

USE CASE 1: Multi-Party Commenting

Actor: Stakeholders (Multiple Users)
Precondition: Document is generated and shared among participants
Postcondition: Comments appear in document context for all users
Basic Flow:

1) User selects text portion or section
2) User adds comment/suggestion
3) System records author + timestamp
4) System notifies other stakeholders
5) Others can reply or react to the comment

USE CASE 2: Approve Changes

Actor: Authorized Stakeholders
Precondition: Comments exist and require review
Postcondition: Document updated if all approvals received
Basic Flow:

1) User opens comment thread
2) User selects Approve/Reject option
3) System records decision
4) If all approve → System updates document version

USE CASE 3: Request Lawyer Consultation

Actor: Logged-in User
Precondition: User has a document requiring legal advice
Postcondition: Lawyer receives request with details
Basic Flow:

1) User opens Lawyer Connect
2) User selects lawyer and sends request
3) System forwards request with case details + document
4) Lawyer receives notification

USE CASE 4: Lawyer Accept/Reject Consultation

Actor: Lawyer

Precondition: Lawyer has pending consultation requests

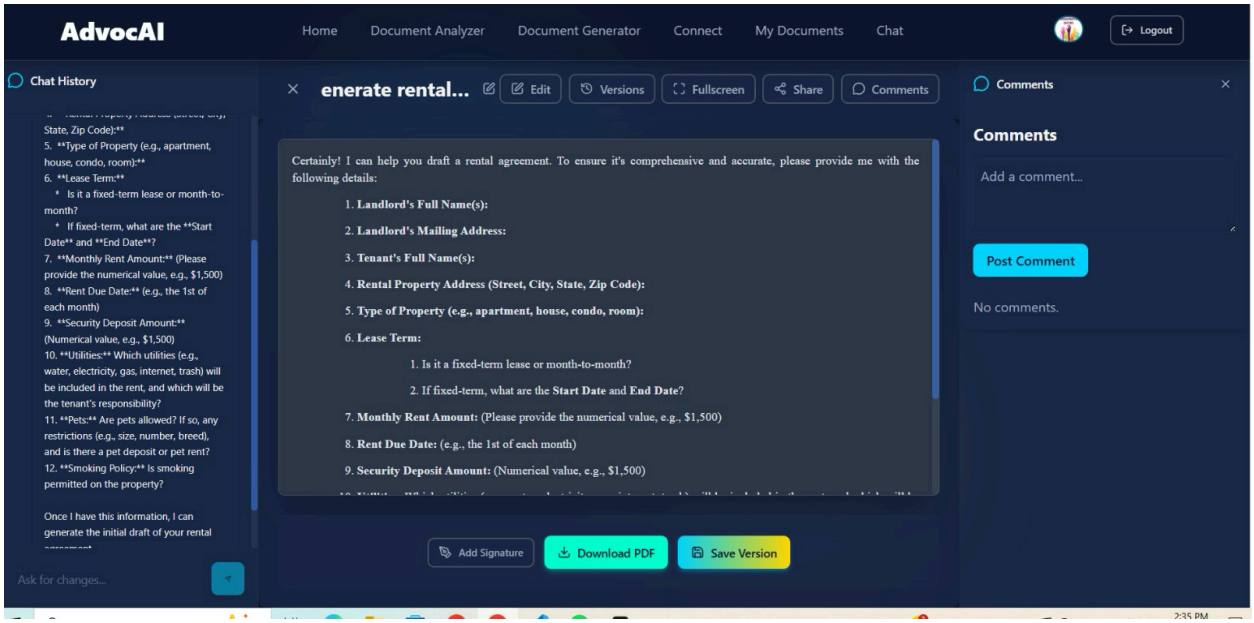
Postcondition: Case request status updated for user

Basic Flow:

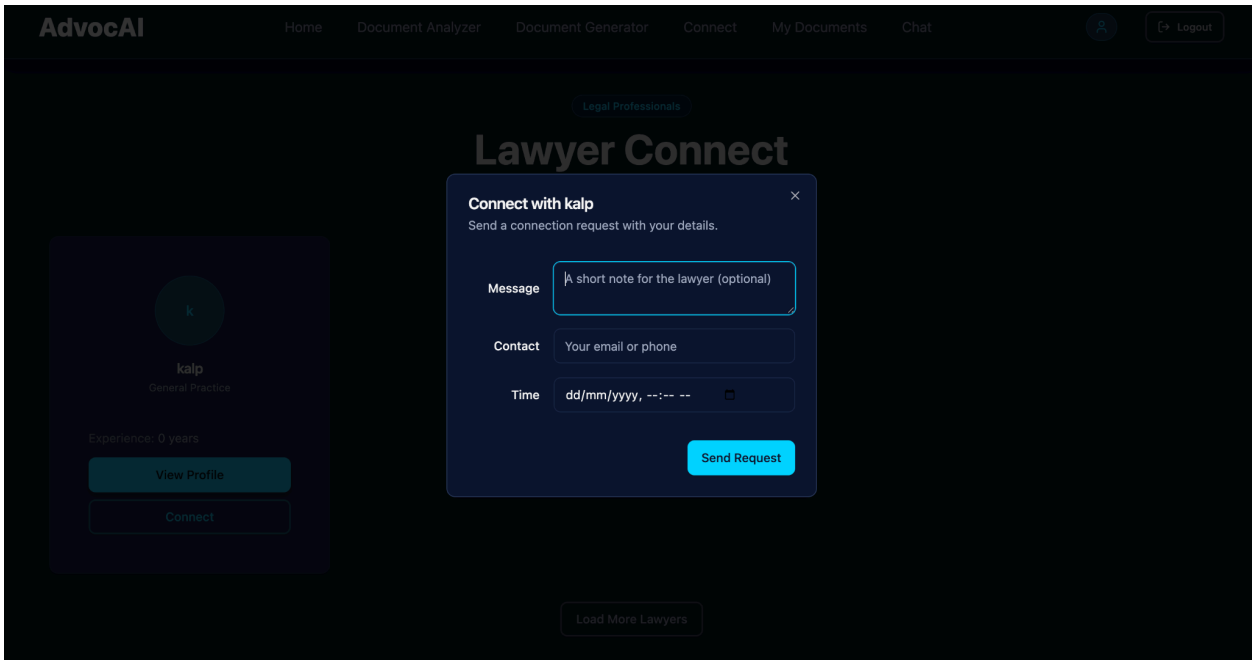
1) Lawyer opens received requests dashboard
2) Lawyer reviews user details and document
3) Lawyer accepts or rejects consultation
4) System updates request status and notifies user

Proof of Concept (POC)

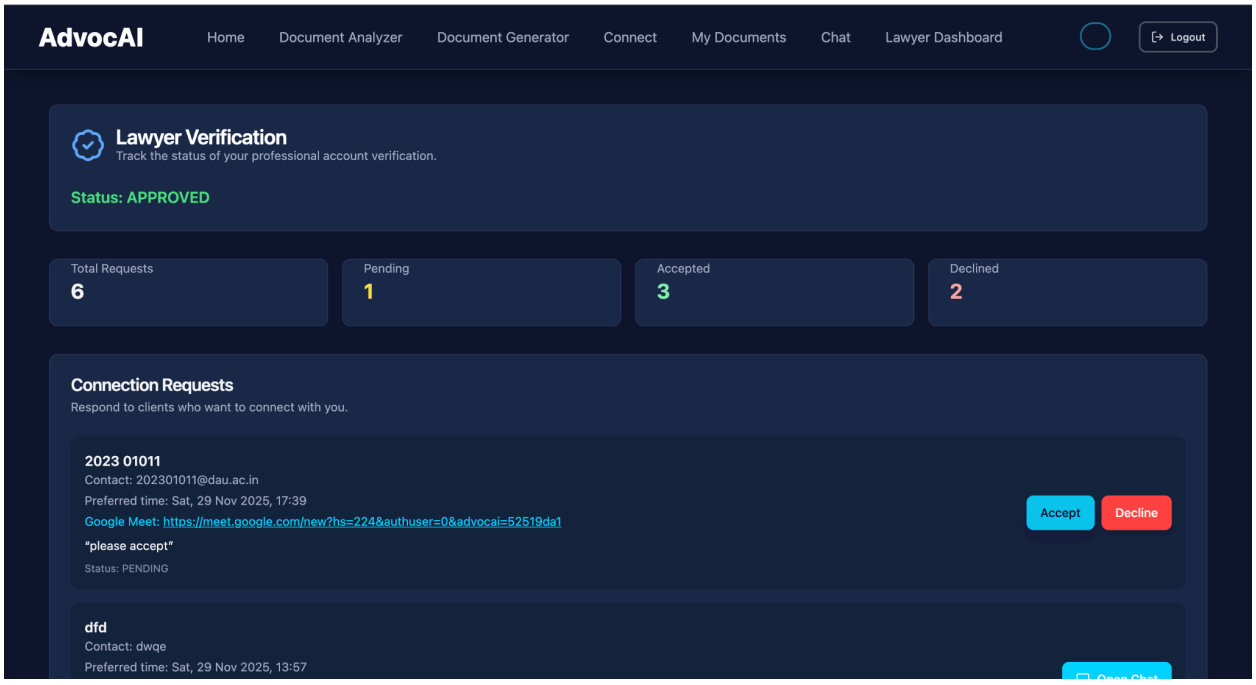
Use Case 1:

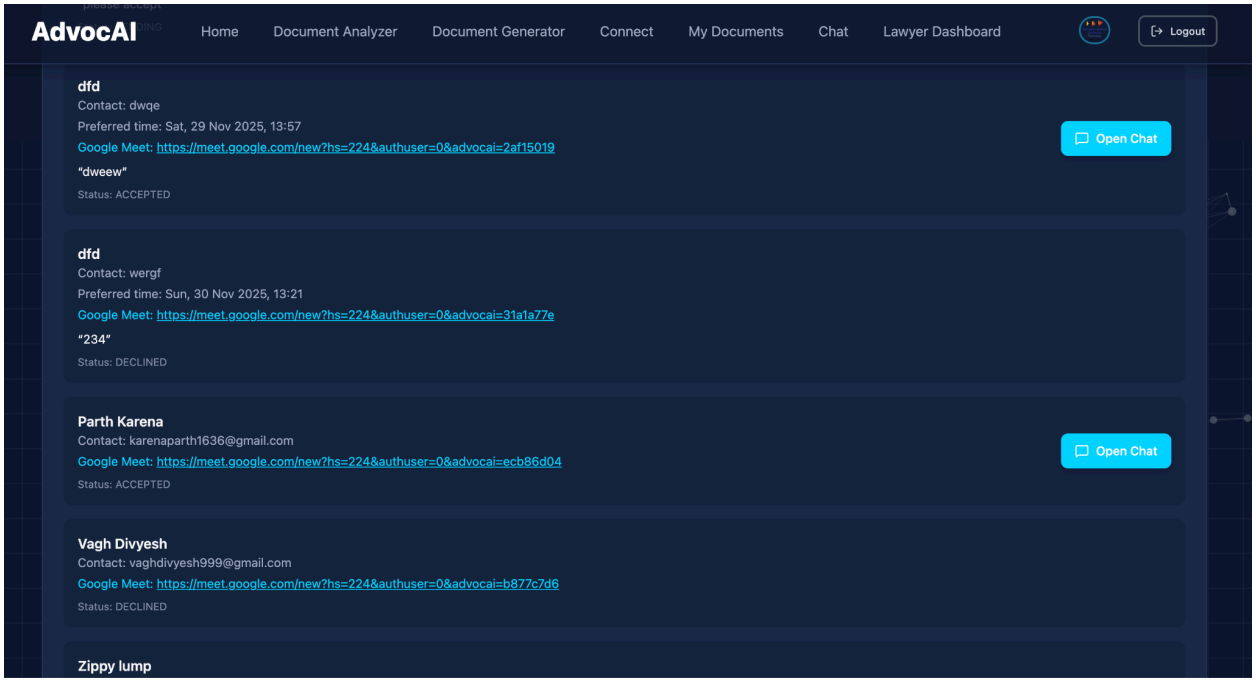


Use Case 2:

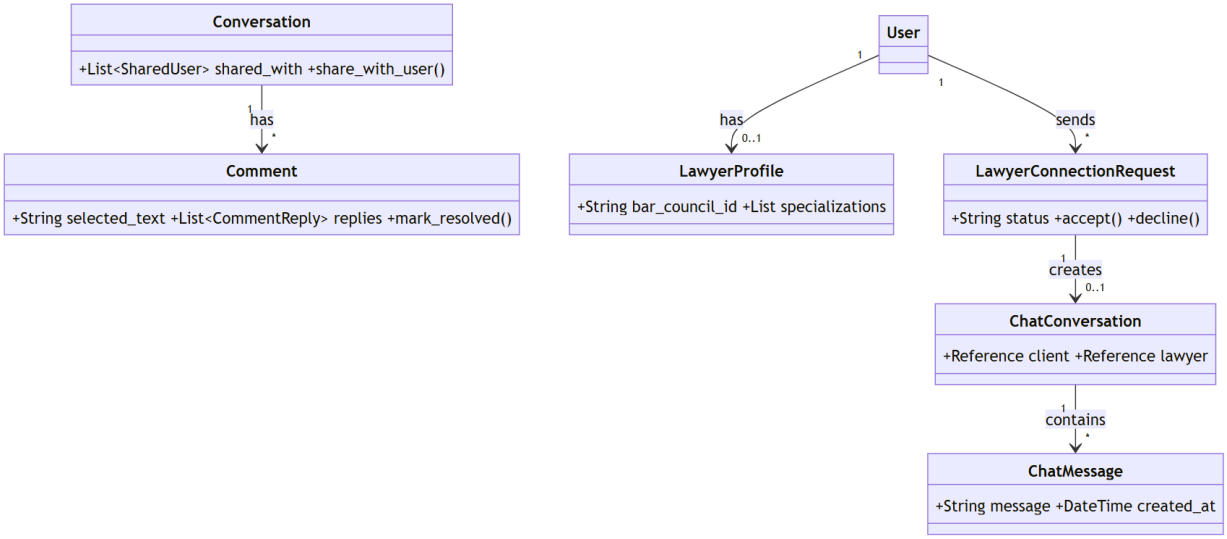


Use Case 3:

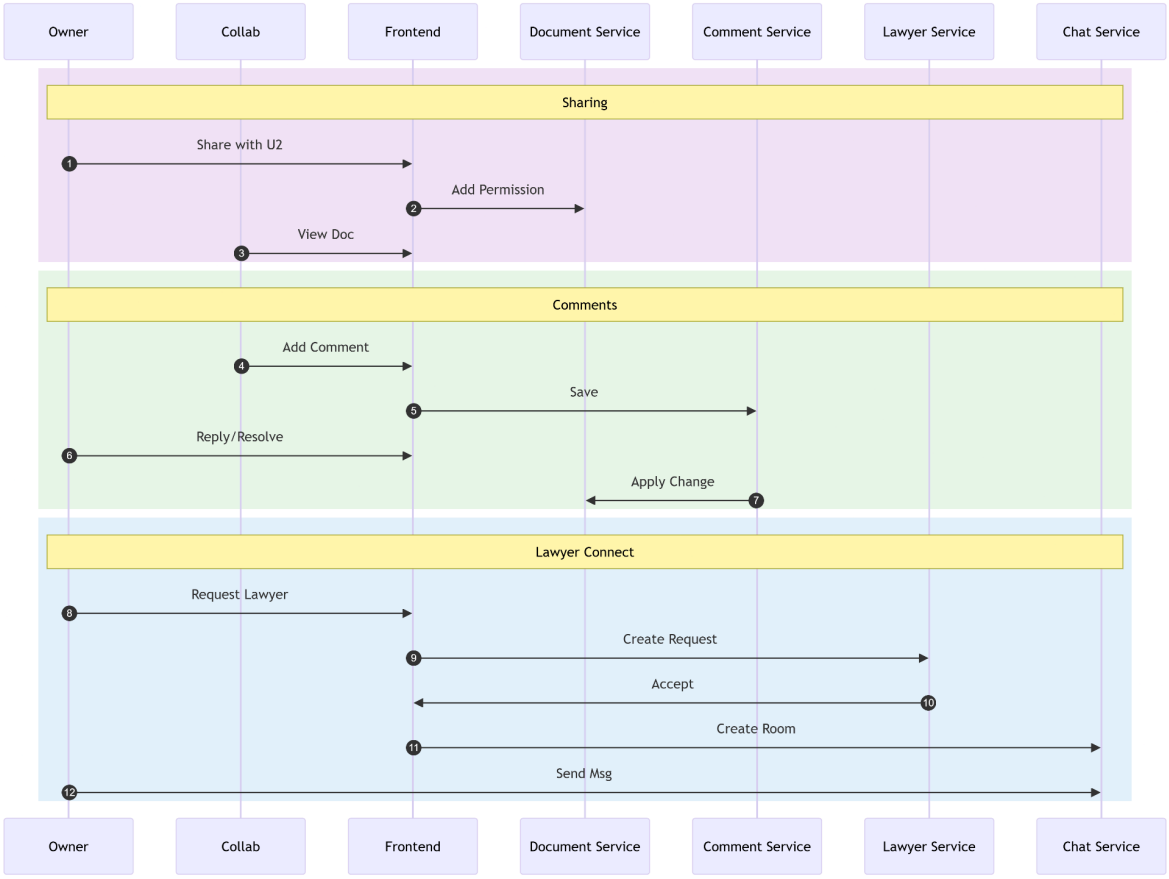




CLASS DIAGRAM



SEQUENCE DIAGRAM



SYSTEM DESIGN REPORT

1. Design Goals

Definition

Main goal of our project AdvocAI is to democratizes legal document generation and analysis, and hence bridge the gap between complex legal frameworks and everyday users with help of Generative AI. Aim is to provide high-quality, context-aware legal drafting, risk assessment, and document summarization with a user-friendly interface.

Trade-offs

- Latency and Accuracy: The system prioritizes the accuracy and legal relevance of the generated content over small response times. Using powerful Large Language Models (LLMs) like Gemini, we accept a slight latency in generation to ensure the output is legally sound and contextually appropriate.
- Flexibility and Structure: We opt for a NoSQL (MongoDB) approach for storing document content and conversation history. Hence trades the strict relational integrity of SQL with the flexibility to handle unstructured text, varying document formats, and rapid iteration of data schemas without complex migrations.
- Real-time Complexity and Simplicity: To support collaborative features and interactive chat, we implemented WebSockets (Django Channels). This increases architectural complexity compared to a standard REST API but is essential for providing a responsive, "live" assistant experience.

2. Subsystem Decomposition

Layers & Partitions

The system follows a 3-tier architecture:

1. Presentation Layer (Frontend): Built with React 19 and Vite, this layer handles user interaction, state management, and real-time updates. It talks with the backend via REST APIs and WebSockets.
2. Application Layer (Backend): A Django-based core that orchestrates business logic. It is partitioned into modular apps:
    - authentication: Manages user identity and security.

    - ai_generator: Handles the logic for drafting legal documents.
    - document_summarizer: Dedicated service for analyzing and summarizing texts.
    - chat: Manages the conversational context and message history.
3. Data Layer: A hybrid storage solution using PostgreSQL/SQLite for relational user data, MongoDB for unstructured document content, and Redis for caching and message brokering.

Cohesion and Coupling

- High Cohesion: Each module (e.g., lawyer, documents) is self-contained with its own models, views, and logic, ensuring that related functionality stays together.
- Loose Coupling: The frontend is mostly decoupled from the backend logic, interacting only through API's. The AI integration is abstracted into utility services, allowing the underlying models (Gemini) to be swapped without changing the core application logic.

3. Concurrency

Identification of Threads

- Request Handling: The Django web server (running via Daphne or Gunicorn) utilizes worker threads to handle multiple incoming HTTP and WebSocket requests concurrently.
- Background Processing: Heavy computational tasks—such as document risk analysis and long-form summarization—are offloaded to Celery workers. These run as separate processes, and hence ensure that the main user interface remains working even while complex AI operations are performed in the background.
- Async I/O: The system leverages Python's asyncio capabilities within Django Channels to manage thousands of concurrent WebSocket connections efficiently, crucial for the real-time chat features.

4. Hardware/Software Mapping

Special Purpose & Allocation

- Client: The application runs in the user's browser, leveraging client-side resources for rendering the rich text editor and UI components.
- Server: The backend is containerized, allowing it to be deployed on

standard cloud infrastructure (e.g., AWS, Render).
- AI Computation: Instead of maintaining expensive GPU clusters, the system maps the "intelligence" hardware to external API providers (Google, OpenAI), significantly reducing operational costs and maintenance overhead.

Connectivity

- REST API: Standard HTTPS is used for transactional data (saving documents, logging in).
- WebSockets (WSS): Persistent connections are established for the chat interface to push AI responses and updates to the client instantly.

5. Data Management

Persistent Objects and Databases

Store (MongoDB): Data such as User Accounts, Authentication Tokens, and Permissions are stored. The conversations and document_versions are stored in MongoDB. Complex, nested JSON structures (like chat history and document metadata) can be stored efficiently using it.

- Caching (Redis): Redis is used as a high-speed cache for session storage and also as a message broker for the Celery task queue, thus ensuring fast data retrieval for the frequently accessed resources.

Data Structure
Documents are modelled not as static files but as versioned entities. Each document entry in the database contains various versions, so that users can track changes, revert back to previous drafts, and hence maintain a history of the document's evolution.

6. Global Resource Handling

Access Control

- Authentication: The system uses JWT (JSON Web Tokens) for stateless, secure authentication. API request is validated using a signed token.
- Role-Based Access: Distinct permissions are enforced for different

user types (e.g., User vs. Lawyer). Middleware intercepts requests to ensure users can only access documents they own or have been explicitly shared with.

Security

- Environment Isolation: Sensitive keys (API credentials, database URLs) are managed via environment variables and not in the code.
- CORS Policies: Cross Origin Resource Sharing policies are defined strictly hence allowing requests only from trusted frontend domains.

7. Software Control

Architecture Style

- The system has a modular monolith structure. Hence providing the simplicity with a single codebase along with maintaining clear boundaries between different functional areas (Apps), which make it easier to develop and test.
- Specific actions trigger asynchronous events. For example, uploading a PDF triggers a "document analysis" event that is picked up by a background worker, decoupling the upload action from the processing logic.

8. Boundary Conditions

Initialization
Upon startup, the system performs a "health check" sequence:

1. Establishes connections to MongoDB and Redis.
2. Validates the presence of critical API keys (Gemini, OpenAI).
3. Loads necessary machine learning configurations.

Failure Handling

- In case, if the AI service is unreachable, the system can catch the exception and provide a error message to the user instead of crashing which helps the user to easily identify the reason for failure.
- Task Retries: Background tasks (like summarization) are configured with retry logic. If a network problem occurs, task is requeue and retrie automatically, thus ensuring data consistency.
- Termination: The system also handles shutdown signals so that

active database connection is closed and running tasks have a grace period to complete.

Subsystem Selection

1. Subsystem Description

AdvocAI is decomposed into seven distinct subsystems. This decomposition aligns with the functional boundaries of the application, ensuring that each subsystem has a clear, single responsibility.

1. Client Subsystem (Frontend)

- Description: A Single Page Application (SPA) built with React 19 and Vite. It is the user interface, that handles all user interactions, rendering the rich text editor, and managing local state.
- Responsibilities:
    - Rendering the dashboard and document editor.
    - Managing WebSocket connections for real-time chat.
    - Formatting and displaying Markdown/JSON responses from the backend.

2. Authentication Subsystem

- Description: A separate module for managing user identity, security, and access control. It uses the JWT handling and OAuth integrations.
- Responsibility:
    - User registration and login (Email/Password & Google OAuth).
    - Issues and verifies JSON Web Tokens (JWT).
    - Manages user roles and permissions.

3. Document Management Subsystem

- Description: It is the core business logic engine for the application. It manages the complete lifecycle of the generated legal documents that includes creation, versioning, storage, and retrieval.
- Responsibility:
    - CRUD operations for documents and conversations.
    - Handling document version control (save trace of edits).
    - Manages sharing permissions (view/edit access).

    - Persisting data to MongoDB.

4. AI Service Subsystem

- Description: It handles all interactions with Large Language Models (LLMs). It acts as an internal service provider for other subsystems.
- Responsibility:
    - Constructing prompts for legal drafting and chat.
    - Managing API keys and rate limits for Google Gemini.
    - Streaming responses for real-time feedback.
    - Parsing raw AI output into structured JSON/Markdown.

5. Analysis & Summarization Subsystem

- Description: It focuses on processing of existing documents to extract insights, summaries, detect risks, and generate summaries.
- Responsibility:
    - Classifies document types (e.g., NDA, Employment Contract).
    - Identify high-risk clauses using heuristic and AI-based patterns.
    - Generate plain-language easy to understand summaries.
    - Manage background tasks via Celery for long-running analyses.

6. Communication Subsystem

- Description: The real-time transport layer that enables instant interaction between the user and the system.
- Responsibility:
    - Managing WebSocket connections (Django Channels).
    - Broadcasting document updates to collaborating users.
    - Streaming AI chat responses chunk-by-chunk.

7. DOCUMENT Generation

· Description: The Document Generation Subsystem is the creative engine of AdvocAI. While the *Analysis Subsystem* reads and understands existing documents, the *Generation Subsystem* is responsible for drafting new legal content from scratch or modifying existing content based on user instructions. It acts as the bridge between user intent (expressed via chat or forms) and the final structured legal document.

· Responsiblities

    - Prompt Engineering: Constructing complex, context-aware prompts that instruct the LLM to act as a legal expert.
    - Context Management: Maintaining the "memory" of the current drafting session (e.g., knowing that "Party A" is "Acme Corp" from previous messages).
    - Format Enforcement: Ensure that the output is not just text, but is structured with Markdown/JSON that the frontend can render and give it as a formatted document.
    - Iterative Refinement: Handles further requests (e.g., "Make the termination clause stricter") by giving the previous draft back into the context.

2. Subsystem Selection Justification

The selection of these subsystems is driven by the principles of High Cohesion and Low Coupling, ensuring a robust and maintainable architecture.

Cohesion Analysis (Interaction Within Subsystems)

The design maximizes interaction *within* subsystems, meaning components that change together stay together.

- AI Service Cohesion: The ai_generator subsystem contains all logic related to prompt engineering, model selection, and API error handling. If we switch from Gemini to OpenAI, only this subsystem needs to change. The documents subsystem doesn't need to know *how* the text is generated, only that it receives text.
- Document Management Cohesion: All logic regarding *how* a document is stored (MongoDB structure, version arrays) is contained within documents. The Frontend doesn't know about MongoDB schemas; it just receives a JSON object.
- Analysis Cohesion: The document_summarizer groups all "read-only" analytical tasks. Risk detection logic, which requires complex pattern matching, is kept separate from the "write-heavy" document creation logic, preventing the core editor from becoming bloated with analysis code.

Coupling Analysis (Interaction Across Boundaries)

The design minimizes dependencies between subsystems, using clear interfaces for necessary interactions.

- Loose Coupling via APIs: The Client Subsystem is completely decoupled from the backend logic. It interacts only through defined REST endpoints and WebSocket events. Alowing the frontend to be rewritten (e.g., to a mobile app) without touching the backend.
- Service-Oriented Architecture: The AI Service is designed as a utility. It does not depend on the Document Management subsystem. It accepts a string (prompt), returns a string (response). This "functional" style means the AI Service has zero knowledge of the database, making it extremely loosely coupled.

Call Pattern Analysis

Caller Subsystem	Callee Subsystem		Justification
Client	Authentication		Necessary: The client must authenticate before accessing other resources.
Client	Document Mgmt		Necessary: The client needs a way to persist user work.
Document Mgmt	AI Service		One-Way Dependency: The Document system "consumes" the AI service to generate content. The AI service never calls back to the

			Document system.
Analysis	AI Service		One-Way Dependency: The Analysis system uses the AI service as a tool to process text.
Communication	Document Mgmt		Integration: The real-time chat needs to persist history. This is a controlled dependency where the communication layer hands off data to the storage layer.

Layers and Partion's Analysis

Based on the subsystems identified in the previous design report, the AdvocAI system architecture utilizes a hybrid approach. It employs Layers to separate levels of abstraction (from user interface to raw intelligence) and Partitions to divide the complex business logic into manageable, functional domains.

Here is the detailed analysis of the subsystems according to your criteria.

1. Layered Decomposition (Vertical Separation)

The system is organized into three distinct layers. This structure adheres to the strict dependency rule: Layer N depends only on Layer $N-1$ and has no knowledge of Layer $N+1$.

Layer 1: Presentation Layer (Top Level)

- Subsystem: Client Subsystem (Frontend)

- Role: Provides the user interface and captures user intent.
- Analysis:
    - Dependency: It depends heavily on the Application Layer below it to fetch data, authenticate users, and save documents.
    - Knowledge: It knows about the API endpoints provided by the lower layer.
    - Abstraction: This is the highest level of abstraction. It is about "User Clicks," "Drag and Drop," and "Visual Rendering," and not the database queries or AI prompts.

Layer 2: Application Layer (Middle Level)

- Subsystems: Document Management, Analysis & Summarization, Authentication, Communication, Document Generation.
- Role: Orchestrates the core business logic of the application. It translates user requests from the Presentation Layer into specific logical operations.
- Analysis:
    - Dependency: It depends on the Service Layer (AI Service) below it to generate content. It does *not* depend on the Presentation Layer (it doesn't care if the request comes from a React app, a mobile app, or a curl command).
    - Service Provision: It provides specific business services (e.g., create_contract, login_user, summarize_pdf) to the Presentation Layer.

Layer 3: Service & Utility Layer (Bottom Level)

- Subsystem: AI Service Subsystem
- Role: Provides raw, generalized intelligence and utility services.
- Analysis:
    - Dependency: It has zero dependencies on the layers above. It is not aware about a User is, nor it knows what Frontend is like. It only care about how to take a text prompt and return a text completion.
    - Knowledge: It is completely ignorant of the higher layers.
    - Abstraction: This is the lowest level of abstraction in this context. It works with raw strings, tokens, and API connections to Gemini

2. Horizontal Separation (Partitions)

In the Application Layer (Layer 2), the system is complex to be a single monolithic block. Therefore, it is divided into Partitions. These subsystems are at same level of abstraction and are weakly coupled.

Partition 1: Document Management Subsystem

- Focus: The "File Cabinet" and "Editor" logic.
- Services Provided: CRUD operations, version control, permission handling.
- Relationship: It operates alongside the Analysis partition. It manages the *structure* and *storage* of the legal documents.

Partition 2: Analysis & Summarization Subsystem

- Focus: The "Legal Researcher" logic.
- Services Provided: Risk detection, clause extraction, summarization.
- Relationship: This partition is distinct from Document Management. While Document Management handles *saving* the file, this partition handles *reading and understanding* the file. They are peers; one handles storage, the other handles computation.

Partition 3: Authentication Subsystem

- Focus: The "Security Guard" logic.
- Services Provided: Identity verification, token issuance.
- Relationship: This is a distinct functional domain. It provides identity services that are used by the other partitions (to check "who is this?"), but its internal logic (hashing passwords, validating Google tokens) is completely independent of legal documents.

Partition 4: Communication Subsystem

- Focus: The "Messenger" logic.
- Services Provided: Real-time WebSocket channels, chat history.
- Relationship: This partition handles the *transport* of messages. It sits alongside the Document partition (which stores the result of the chat) but focuses specifically on the real-time delivery mechanism.

Partition 5: Document Generation Subsystem

- Focus: The "Drafter" or "Creative Engine" logic.
- Services Provided: Prompt engineering, context management,

structured document creation (Markdown/JSON), iterative refinement of drafts based on user feedback.

- Relationship: This partition acts as the creative peer to the Analysis partition. While Analysis reads documents, this partition writes them. It consumes raw intelligence from the AI Service Layer and produces structured content that is handed off to the Document Management partition for storage and the Communication partition for display.

3. Summary of Relationships

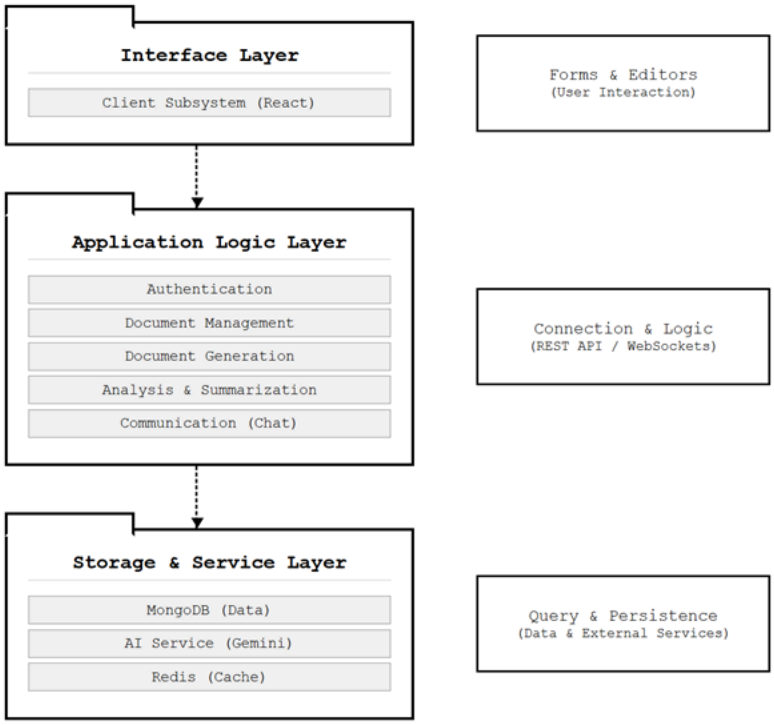
Subsystem	Classification	Justification
Client	Layer (Top)	Calls the Application layer. Never called by others. High abstraction (UI).
Document Mgmt	Partition (Middle Layer)	Peer to Analysis/Auth. Calls AI Service (Lower). Called by Client (Higher).
Analysis	Partition (Middle Layer)	Peer to Docs/Auth. Calls AI Service (Lower). Called by Client (Higher).
Authentication	Partition (Middle Layer)	Peer to Docs/Analysis. Provides security services to other partitions.

AI Service	Layer (Bottom)	Called by Application layer partitions. Calls no one (internal). Low abstraction (Raw I/O).
Doc Generation	Partition (Middle Layer)	Peer to Analysis/Docs. Calls AI Service (Lower). Called by Client/Comm (Higher). Focuses on *creation* logic.

Adherence to Criteria

1. Layering Rule: The AI Service (Bottom) never calls the Document Management (Middle). The Document Management (Middle) never calls the Client (Top). This strictly follows the "Layer can only depend on lower layers" rule.
2. Partitioning Rule: The Document Management and Analysis subsystems are at the same level of abstraction (Business Logic). They are separated to ensure that complex analysis logic doesn't clutter the document storage logic, adhering to the "independent subsystems on the same level" rule.

Architectural Diagram


Interface Layer
Client Subsystem (React)
Application Logic Layer
Authentication
Document Management
Document Generation
Analysis & Summarization
Communication (Chat)
Storage & Service Layer
MongoDB (Data)
AI Service (Gemini)
Redis (Cache)
Forms & Editors
(User Interaction)
Connection & Logic
(REST API / WebSockets)
Query & Persistence
(Data & External Services)

OBJECT DESIGN REPORT

1. Introduction

System is decomposed into subsystems (Client, Authentication, Document Management, Analysis, Generation, AI Service). This phase defines the solution objects, components, applies design patterns, and specify class interfaces. Thus reduce the gap between the high-level system architecture and the concrete implementation code.

2. Component Selection & Reuse Strategy

White-box Reuse (Inheritance) and Black-box Reuse (Composition/Aggregation) is used to maximize efficiency and reliability.

2.1. Reuse

We selected the following existing frameworks and libraries:

- Django & Django REST Framework (Backend Core)
    - White-box Reuse (Inheritance).
    - Our models (e.g., User) inherit from django.db.models.Model. Our API views inherit from rest_framework.views.APIView.

      This provides robust, security-tested implementations of HTTP handling, ORM, and Authentication.

- React (Frontend UI)
    - Black-box Reuse (Composition).
    - UI is built by composing standard HTML elements and third-party libraries (e.g., lucide-react for icons) into complex components like DocumentEditor.
- LangChain & Google Generative AI
    - Black-box Reuse (Delegation).
    - The system uses complex NLP tasks to these libraries. We do not modify their internal logic but interact with their public APIs.
- Celery & Redis
    - Is used for asynchronous task queue management.

2.2. New Solution Objects

DocumentSession (MongoDB Model): A flexible schema object designed to store unstructured analysis results (risks, summaries) that don't fit neatly

into SQL rows.

Risk Detector: A domain-specific service object that orchestrates the logic of parsing legal text and mapping it to specific risk categories.

3. Design Patterns Applied

To solve recurring design problems, the following patterns were identified and applied:

3.1. Adapter Pattern

- Problem: The external Google Gemini API has a complex interface that might change or need to be swapped for OpenAI in the future.
- Solution: The utils/gemini_client.py module acts as an Adapter.
    - Interface: It provides a simple function get_gemini_response(prompt).
    - Implementation: It encapsulates the specific configuration, authentication, and error handling required by the Google SDK. The rest of the application depends on this adapter, not the raw SDK.

3.2. Observer Pattern (Pub-Sub)

- Problem: The frontend needs to know immediately when a document is updated by another user or when AI analysis is complete.
- Solution: Implemented via Django Channels (WebSockets).
    - Subject: The DocumentConsumer class.
    - Observers: Connected frontend clients.
    - Mechanism: When an event occurs (e.g., document_content_change), the consumer broadcasts a message to the specific "Group" (channel) that clients are subscribed to.

3.3. Strategy Pattern (Implicit)

- Problem: Different types of documents (NDAs vs. Employment Contracts) require different analysis prompts.
- Solution: The document_classifier.py determines the document type, and the system selects the appropriate "Strategy" (prompt template

and risk rules) from enhanced_risk_patterns.py to execute the analysis.

4. Subsystem Interface & Class Specification

This section specifies the key classes, their responsibilities, and interfaces for the primary subsystems.

4.1. Analysis Subsystem

- Class: DocumentSummarizer
  - Responsibility: Orchestrates the chunking, analysis, and aggregation of document text.
  - Operations:
    - generate_document_analysis(text: str) -> Dict: Public entry point.
    - _chunk_document(text: str) -> List[str]: Private helper to handle token limits.
    - _analyze_chunk_with_llm(chunk: str) -> Dict: Delegates to AI Adapter.
  - Constraints: Must handle API rate limits gracefully (implemented via retry logic).

4.2. Document Management Subsystem

- Class: MongoClient (Wrapper)
  - Responsibility: Abstracting MongoDB CRUD operations.
  - Operations:
    - save_conversation(title, messages, content) -> str: Creates new entry.
    - update_conversation(id, content) -> bool: Appends new version.
    - get_document_version(id, version_num) -> str: Retrieves historical data.
  - Visibility: Public methods used by Django Views; internal connection logic is private.

4.3. Communication Subsystem

- Class: DocumentConsumer (inherits AsyncWebsocketConsumer)
  - Responsibility: Handling real-time WebSocket events.
  - Operations:

        - connect(): Authenticates and adds user to the document group.
        - receive(text_data): Routes incoming JSON messages to handlers (e.g., handle_chat_message).
        - disconnect(): Cleans up group membership.

5. Object Model Restructuring & Optimization

During the design process, the object model was refined to address performance and extensibility:

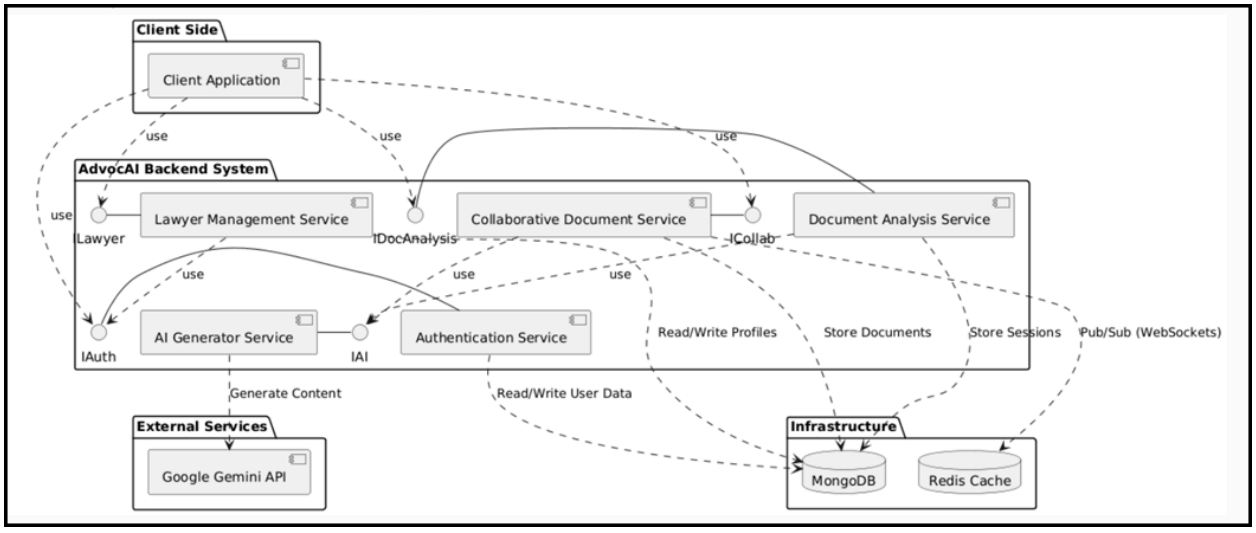
- Restructuring for Extensibility:
    - Initially, risk patterns were hardcoded. We restructured this into enhanced_risk_patterns.py, a configuration-driven module. This allows adding new document types (e.g., "Lease Agreement") by simply adding a new dictionary entry, without modifying the core logic.
- Optimization for Performance:
    - Asynchronous Processing: Heavy analysis tasks were moved from the APIView (synchronous) to Celery Tasks (asynchronous). This prevents the HTTP request from timing out while the AI processes large documents.
    - Caching: We introduced cache_utils.py to store AI responses for identical text chunks using Redis. This reduces API costs and latency for repeated analyses.

6. Conclusion

The Object Design of AdvocAI successfully bridges the architectural requirements with concrete implementation details. By reusing robust frameworks (Django/React) and isolating volatile external dependencies (AI APIs) through the Adapter pattern, the design ensures maintainability. The specification of clear interfaces for the Analysis and Document subsystems allows for parallel development and future scalability.

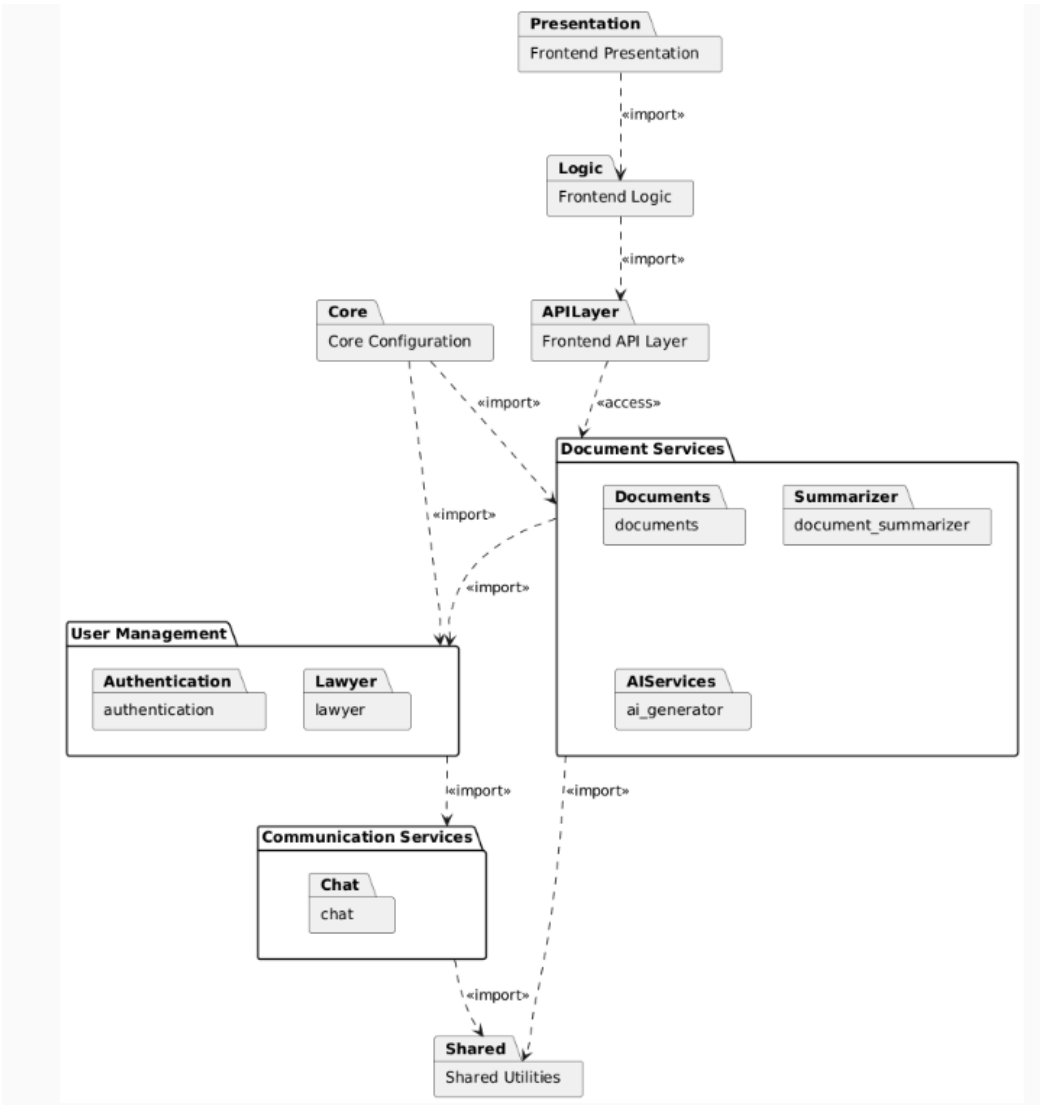
COMPONENT DIAGRAM

Depicts how components are wired together to form bigger component or system, Component interacts with each other though interfaces, Connect the required interface of one component with the provided interface of another component.



PACKAGE DIAGRAM

Groups of related classes to identify and to understand dependencies


Presentation
Frontend Presentation
«import»
Logic
Frontend Logic
«import»
Core
Core Configuration
APILayer
Frontend API Layer
«import»
«access»
Document Services
Documents
documents
Summarizer
document_summarizer
«import»
«import»
User Management
Authentication
authentication
Lawyer
lawyer
AIServices
ai_generator
«import»
«import»
Communication Services
Chat
chat
«import»
Shared
Shared Utilities