COMP9900: Computer Science Project

School of Computer Science and Engineering, UNSW Sydney

Dr. Basem Suleiman

Project 63: SAT Decisions Scraper & Search Application with RAG

9900-T10A-CHOCOLATE

Miguel Ilagan (z5117944@ad.unsw.edu.au, z5117944, Data Engineer)

Guan Xin Miao (z5442846@ad.unsw.edu.au, z5442846, Front end Developer)

Will Ren (z5429870@ad.unsw.edu.au, z5429870, Machine Learning Engineer)

Zequn Wu (z5485493@ad.unsw.edu.au, z5485493, Product Owner)

Zhengxin Zeng (z5457832@ad.unsw.edu.au, z5457832, Data Architect)

Qinan Ji (z5399710@ad.unsw.edu.au, z5399710, Machine Learning Engineer)

March 11, 2024

Background

Problem domain and summary of solution

The State Administrative Tribunal has published many legal decisions over the past few years that provide insight for similar legal cases. However, these cases and decisions are not easy to search, analyse and summarise for other use purposes. For example, current searching methods depend on keyword search. However, for some complex cases this method is not very useful. Furthermore, there is insufficient time for judges, lawyers and the general public to read detailed and lengthy cases.

Problems being solved:

1. Scrape SAT data and build a SAT database:
   Scrape public SAT data, process and store them in a suitable database for downstream tasks.
2. Improve searching efficiency:
   Current search engines only support keyword search, which is often not accurate and efficient. Furthermore, since common users are typically unaware of technical jargon which can affect the accuracy of the cases presented to the user.
3. Inconvenient in case comparison:
   Currently there are no AI models that can help judges and lawyers compare legal cases. This means that they are left to make analysis themselves which is a time consuming process since most legal precedents are lengthy and contain a lot of detail.
4. Multiple format input:
   There is currently a lack of compatible document types where typically only text or images are accepted as inputs. Similarly, updating prior queries typically only accepts text as input without consideration of other formats such as audio or video. Furthermore, users typically are unable to submit or download multiple documents which can be beneficial for legal matters.
5. Legal reasoning:
   Current systems cannot produce AI generated analysis of legal cases that can facilitate judgements based on past adjudication.
6. Insufficient summary:
   In the current SAT database, only some of the cases have included a case summary. Hence, users often need to read and analyse the entirety of the text to gain insight on the case which is a very time consuming process.

Solution summary:

1. Auto adjudication scraping:
    - Scrape SAT website automatically, extract data and metadata.
    - Data is saved into a local database and frequently updated.
    - Method: Python framework called 'Scrapy' is used to perform web scraping. The data is then stored in PostgreSQL and FAISS databases.
2. AI language search with RAG and LLMs:
    - Support natural language search for users, not only keywords.
      Such as:
        "If there are some cases about rent lease disputes?"
        "Can I know something about contract conflict?"
    - Provide automatic completion of input content, allow AI to associate and supply users' input.
    - Allow users to submit multi-type documents to search.
3. AI generate summary:
    - Allows users to not need to read full pages of articles since the AI can generate summary for them.
    - Supports multiple document export types and diverse text display (e.g bold, change color, drawn lines, etc).
    - Technical realisation: RAG (LangChain, query rewriting, query routing, prompt engineering, multi-hop, multi-query) and LLMs
4. Related case recommendation
    - Recommend cases that are most similar with current cases to help judges and lawyers find reliable support.
    - Provide case comparison so users can find similarities in different cases.
    - Technical realization:RAG (hybrid search, reranking) and Reasoning LLMs
5. User friendly UI
    - Simple template with a few steps and buttons to filter.
    - Support multiple uploads.
    - Serious style, no need to be fashionable.
    - Conversational UI having histories.

Current existing systems and their benefits/limitations

Extensive research has been performed on RAG and its applications in the legal domain by referencing papers, open-sourced projects, and commercial platforms. Analysis of three existing systems is shown below.

Westlaw Precision Australia from Thomson [14]

- Problem it solves:
  Westlaw Precision Australia of Westlaw Precision Global is a legal research platform combining traditional search and AI assisted search with case law, statutes, and secondary legal resources. It solves the problem of inefficient search in tons of legal cases by enabling keyword, filtering and context aware retrieval.
- Strengths to consider in our project:
  - Keyword matching from a large database.
  - Filtering for legal cases such as jurisdiction, year etc.
  - Provide citation tracking such as links for users to click.
- Weakness and Limitations to avoid or overcome:
  - Having to pay hundreds of dollars per month to access this service which may prevent the public from accessing it.
  - The system has limited adaptability to SAT decisions in WA and the algorithms may not be fully suitable for SAT
  - Do not have the option to customise for SAT decisions and to evaluate the system performance.

LexisNexis Australia [14]

- Problem it solves:
  LexisNexis Australia addressed the problem of efficient legal search by providing a wide range of databases and enabling professionals to analyze legal texts efficiently.
- Strengths to consider in our project:
  - Advanced search functionality including natural language query, filters and alerts which allow users to find relevant sources quickly.
  - Offer extensive practical guidance and checklists, helping lawyers work efficiently.
  - Offers an legal research platform called Lexis+ AI which has key features including conversational search, document drafting, summarisation, and analysis while offering linked citations.
- Weakness and Limitations to avoid or overcome

    - Lexis+ AI is a paid subscription service which is quite costly, making it difficult for individual users, small firms, and the general public to access.
    - AI-generated responses can still produce inaccuracies or hallucinations in some situations.
    - Lexis+ AI is optimized for court case law, statutes, and general legal research. It does not specifically cater to SAT rulings.

RAGFlow (open-source, for general domains) [15]

- Problem it solves:
  RAGFlow is an open-source RAG platform designed to improve search and summarisation in various domains. Users can download the system and build their own knowledge base and RAG system with LLMs.
- Strengths to consider in our project:
    - Open-source and can be extended. Users can build their own RAG system based on it with some customization.
    - Supports different embedding and LLM models
    - Can be applied locally.
- Weakness and Limitations to avoid or overcome
    - Lacks legal domain optimisation since it is designed for general domain knowledge.
    - Still need technical people to install and configure.
    - No option to evaluate system performance.

More research, papers and references can be found at Confluence Page.

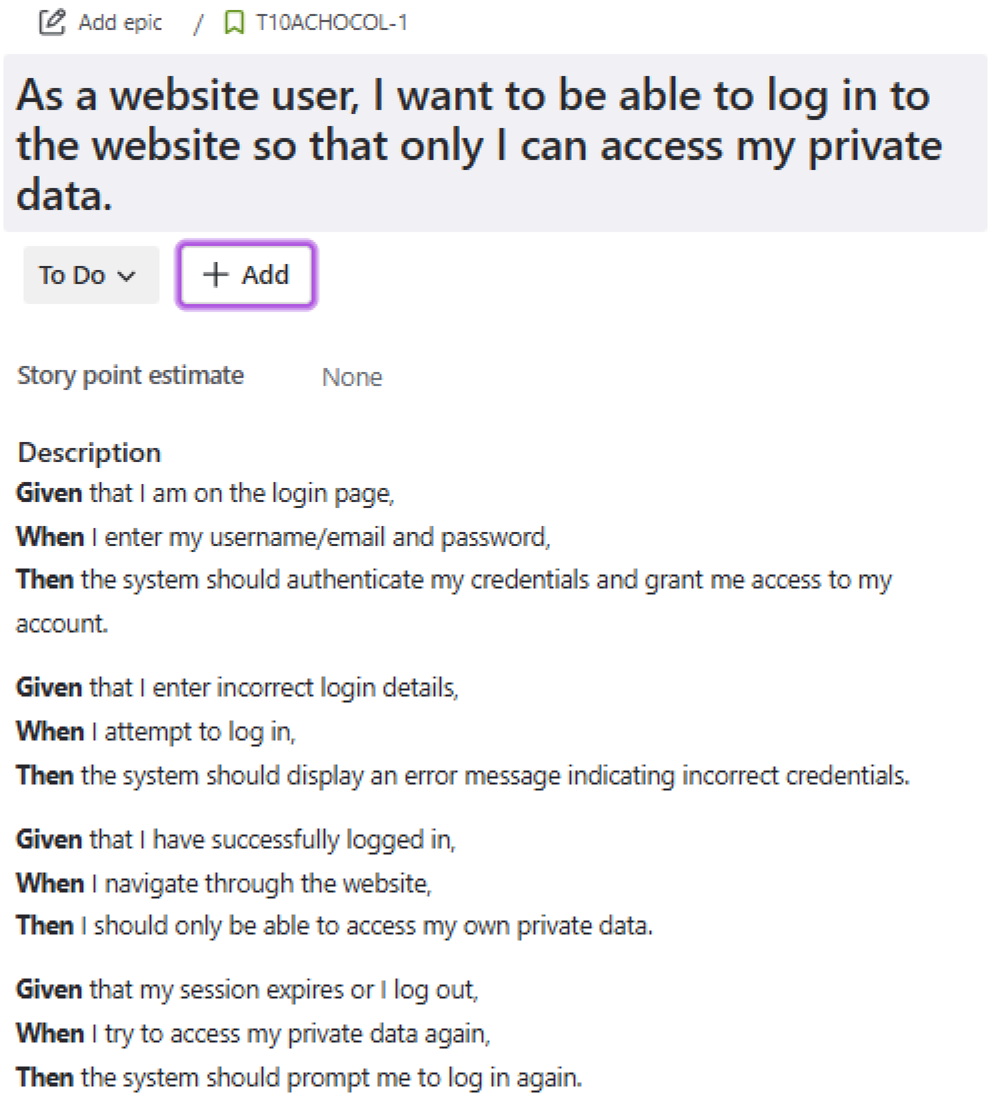
Firstly, our SAT RAG project will collect all SAT decisions data and their corresponding metadata (e.g background, summary, etc.). This data will then be pre-processed and categorised for better retrieval. Hybrid Search (Keyword search and Semantic search) is then implemented to find the most relevant documents efficiently. Documents are then reranked so that only the most relevant documents are considered. In addition, Agentic RAG with reasoning models will be applied to help users answer more complex questions regarding case analysis and judging. Of course, citations will be provided. Lastly, evaluation and optimization in the SAT domain will be implemented.

User Stories and Sprints

User groups

1. **All Website users:**

   When I enter the website, I don't want access to be limited to just my information, such as chat history, uploaded documents, and personal details. Therefore, I need a secure and reliable system to manage my private data.



Add epic / T10ACHOCOL-1

As a website user, I want to be able to log in to the website so that only I can access my private data.

To Do + Add

Story point estimate None

Description

Given that I am on the login page,
When I enter my username/email and password,
Then the system should authenticate my credentials and grant me access to my account.

Given that I enter incorrect login details,
When I attempt to log in,
Then the system should display an error message indicating incorrect credentials.

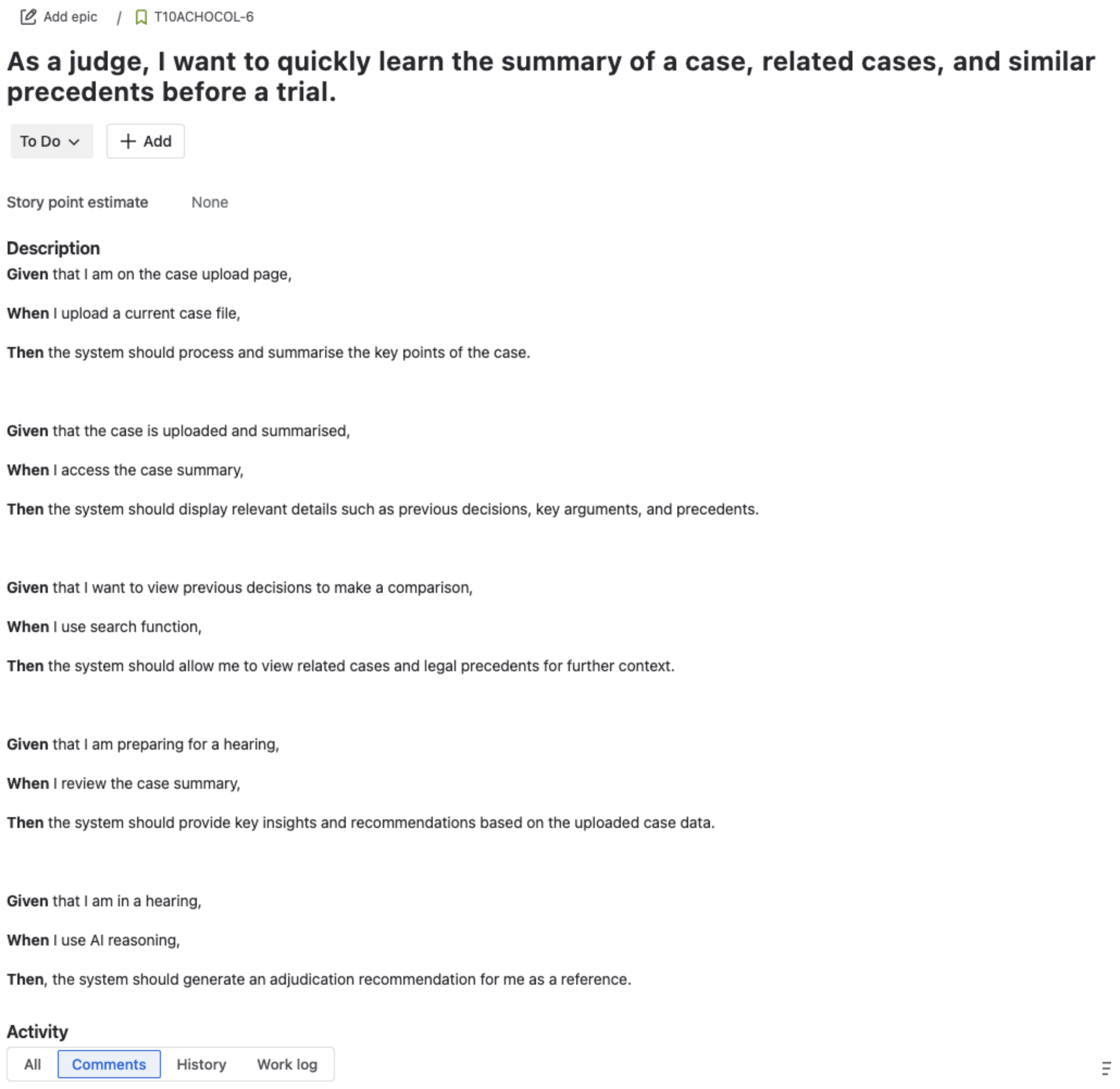
Given that I have successfully logged in,
When I navigate through the website,
Then I should only be able to access my own private data.

Given that my session expires or I log out,
When I try to access my private data again,
Then the system should prompt me to log in again.

2. **Judger:**

When judging, I need to find some legal provisions that can explain the case and make sure the decision has a reliable legal basis. For the AI model, I want AI to generate a preliminary decision recommendation as reference to improve adjudication efficiency.

In addition, I also want to take adjudication notes in a private database where I can refer to past thinking to make all decisions at a similar level.



As a judge, I want to quickly learn the summary of a case, related cases, and similar precedents before a trial.

Story point estimate None

Description

Given that I am on the case upload page,

When I upload a current case file,

Then the system should process and summarise the key points of the case.

Given that the case is uploaded and summarised,

When I access the case summary,

Then the system should display relevant details such as previous decisions, key arguments, and precedents.

Given that I want to view previous decisions to make a comparison,

When I use search function,

Then the system should allow me to view related cases and legal precedents for further context.

Given that I am preparing for a hearing,

When I review the case summary,

Then the system should provide key insights and recommendations based on the uploaded case data.

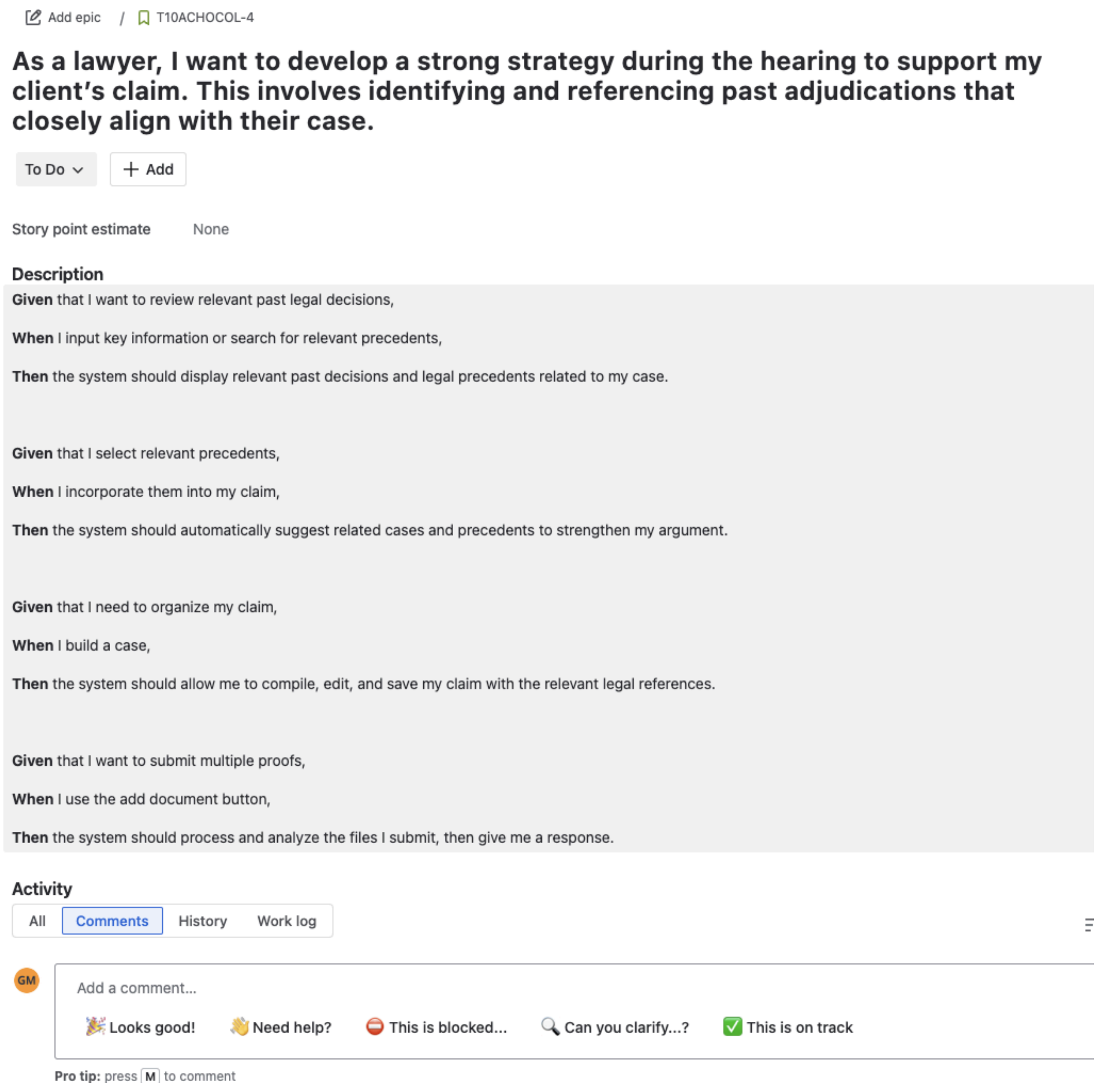
Given that I am in a hearing,

When I use AI reasoning,

Then, the system should generate an adjudication recommendation for me as a reference.

Activity

3. **Lawyer:**
   I want AI to generate some legal reasoning to support, including: some key points in related law, some similar trial cases, and differences between different cases. Also, I may have multiple types of evidence, like text and audio.



As a lawyer, I want to develop a strong strategy during the hearing to support my client's claim. This involves identifying and referencing past adjudications that closely align with their case.

Story point estimate None

Description

Given that I want to review relevant past legal decisions,

When I input key information or search for relevant precedents,

Then the system should display relevant past decisions and legal precedents related to my case.

Given that I select relevant precedents,

When I incorporate them into my claim,

Then the system should automatically suggest related cases and precedents to strengthen my argument.

Given that I need to organize my claim,

When I build a case,

Then the system should allow me to compile, edit, and save my claim with the relevant legal references.

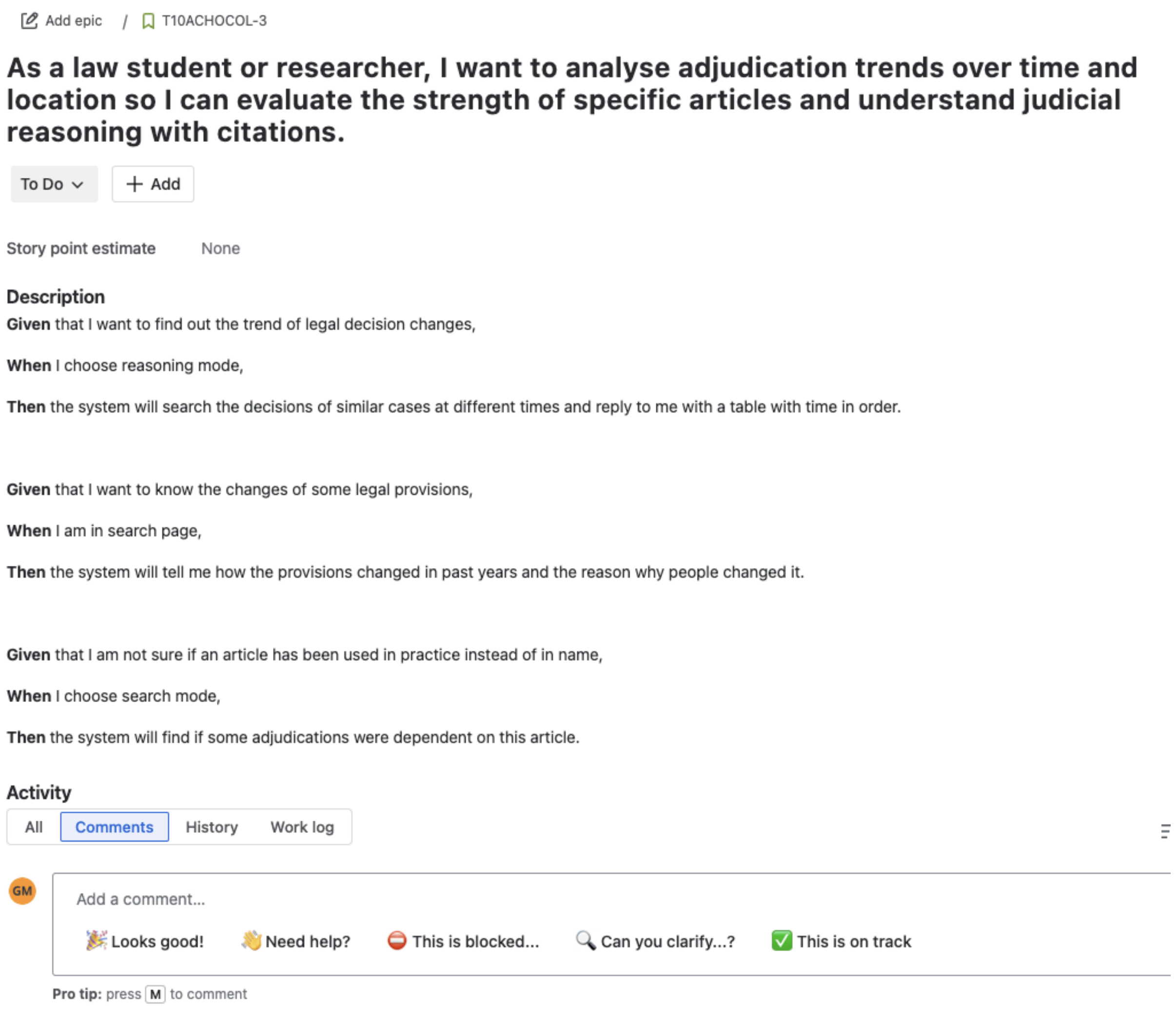
Given that I want to submit multiple proofs,

When I use the add document button,

Then the system should process and analyze the files I submit, then give me a response.

4. **Law student & research:**

   I need a trend of adjudication results that depend on time and location. I need to find the strength of specific articles in certain cases. I will also need a report on usage of particular law provisions, including cite numbers and logical reasoning behind different adjudications made by judges.



Add epic / T10ACHOCOL-3

As a law student or researcher, I want to analyse adjudication trends over time and location so I can evaluate the strength of specific articles and understand judicial reasoning with citations.

To Do + Add

Story point estimate None

Description

Given that I want to find out the trend of legal decision changes,

When I choose reasoning mode,

Then the system will search the decisions of similar cases at different times and reply to me with a table with time in order.

Given that I want to know the changes of some legal provisions,

When I am in search page,

Then the system will tell me how the provisions changed in past years and the reason why people changed it.

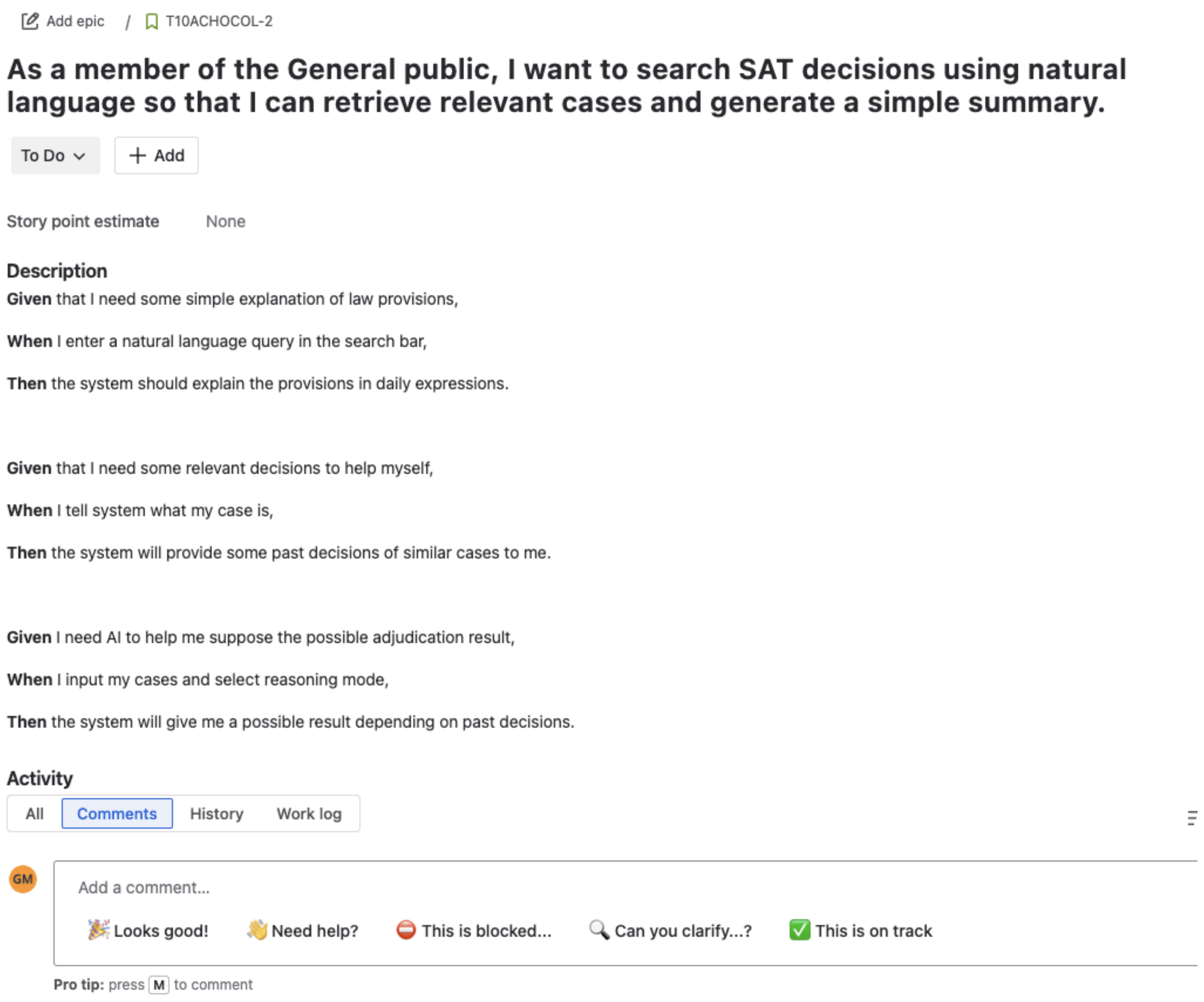
Given that I am not sure if an article has been used in practice instead of in name,

When I choose search mode,

Then the system will find if some adjudications were dependent on this article.

Activity

5. **General public:**
   For the general public, I need a simple summary of related laws. I can use simple words to find something I want to know, since I am not a professional. I also want to know what I can do based on past cases.



As a member of the General public, I want to search SAT decisions using natural language so that I can retrieve relevant cases and generate a simple summary.

Story point estimate None

Description

Given that I need some simple explanation of law provisions,

When I enter a natural language query in the search bar,

Then the system should explain the provisions in daily expressions.

Given that I need some relevant decisions to help myself,

When I tell system what my case is,

Then the system will provide some past decisions of similar cases to me.

Given I need AI to help me suppose the possible adjudication result,

When I input my cases and select reasoning mode,

Then the system will give me a possible result depending on past decisions.

Jira Issue/Ticket and Acceptance Criteria

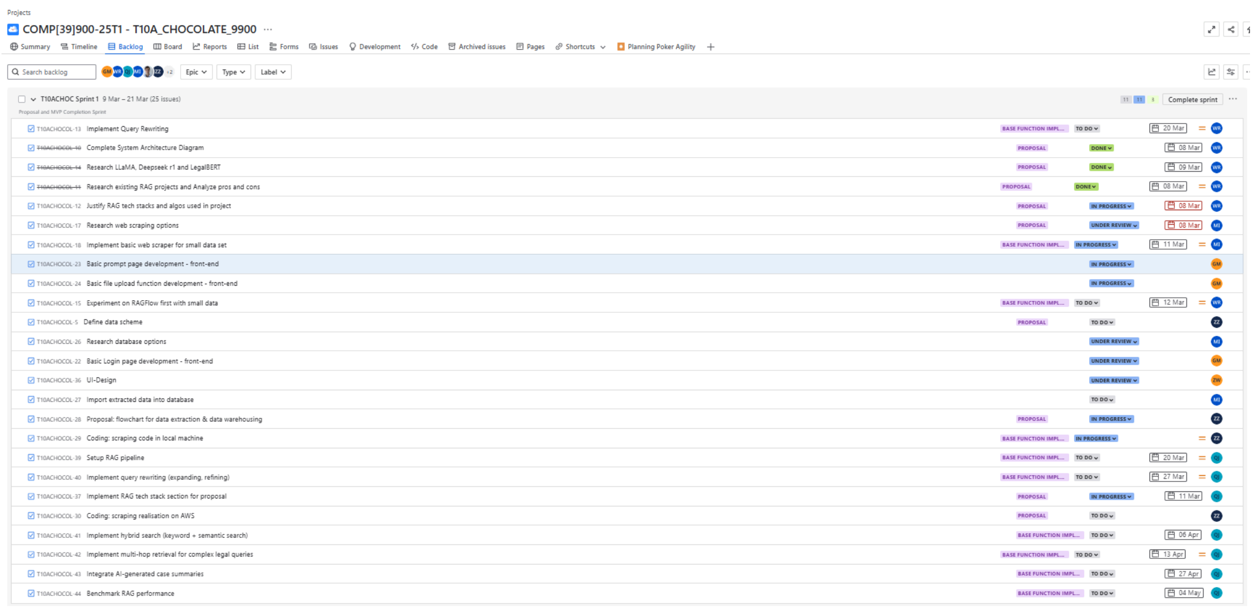
Jira Ticket	Issue/Ticket	Acceptance Criteria	Story Points
T10ACHOCOL-11	Research existing R&D projects and Analyze pros and cons	A comparison analysis of 3 existing projects in Proposal.	2
T10ACHOCOL-12	Justify R&D tech stacks and align used in project	Justification part in Proposal	1
T10ACHOCOL-17	Research web scraping options	Identify web crawler tools that meet the project requirements	2
T10ACHOCOL-18	Implement basic web scraper for small data set	Web scraper to be able to fetch data from HTML, PDF, and docx files.	2
T10ACHOCOL-19	Experiment on RAGFlow first with small data	Generate a result from a small set of data.	3
T10ACHOCOL-22	Basic login page development - front-end	Login page with username and password input bar, avatar,	2

		sign up and forgot password hyperlink.	
T10ACHOCOL-23	Basic prompt page development - front-end	User input window, Result window, Chat history window.	3
T10ACHOCOL-24	Define data scheme	Research SAT decision samples, figure out data schema including what type of metadata, category, data, summary, hyperlink etc.	2
T10ACHOCOL-25	Basic file upload function development - front-end	The user should be able to upload PDF, docx, or HTML files from this page.	2
T10ACHOCOL-26	Research database options	Identify database that meet the project requirements	2
T10ACHOCOL-27	Import extracted data into the database	Data should be correctly inserted into the database.	2

T10ACHOCOL-28	Proposals flowchart for data extraction & data warehousing	Completed the structure design and placed it into the document	2
T10ACHOCOL-29	Develop scraping model in local machine	Scrape web data into local machine	2
T10ACHOCOL-30	Coding scraping realisation on AWS	Scraping program will be worked in AWS	2
T10ACHOCOL-36	UI Design	High quality UI design for all necessary web pages.	3
T10ACHOCOL-39	Setup RAG pipeline	Implement the basic RAG pipeline with vector retrieval and document embeddings.	3
T10ACHOCOL-40	Implement query rewriting (expanding, refining)	Improve user queries by adding synonym expansion and structured transformations for better search results.	3
T10ACHOCOL-41	Implement hybrid search (keyword + semantic search)	Combine BM25 for keyword matching and FAISS for vector-based similarity search.	4

T10ACHOCOL-42	Implement multi-hop retrieval for complex legal queries	Allow the system to retrieve relevant cases step-by-step for multi-layered reasoning.	4
T10ACHOCOL-43	Integrate AI-generated case summaries	Use the selected LLM (QwQ-32B) to generate legal case summaries.	5
T10ACHOCOL-20	Connect RAG component to read data from database in order to retrieve relevant cases	The RAG system is successfully connected to the designated database. Can fetch relevant legal documents from the database based on user queries	3
T10ACHOCOL-21	Improve retrieval mechanisms, refine prompt engineering, and minimize hallucinations to ensure relevant responses. Improve generation capability.	Improved Retrieval Accuracy and response quality. Less hallucination and errors.	5

Product Backlog, Sprints


Projects

COMP[39]900-25T1 - T10A_CHOCOLATE_9900

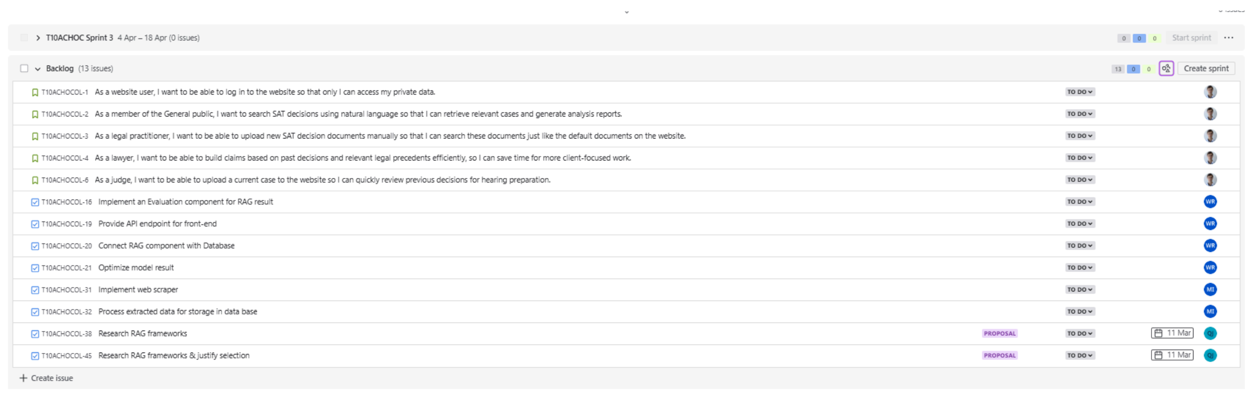
Summary | Timeline | Backlog | Board | Reports | List | Forms | Issues | Development | Code | Archived issues | Pages | Shortcuts | Planning Poker Agility

Search backlog | Epic | Type | Label

T10ACHOC Sprint 1 9 Mar – 21 Mar (25 issues) — 11 11 3 — Complete sprint

Proposal and MVP Completion Sprint

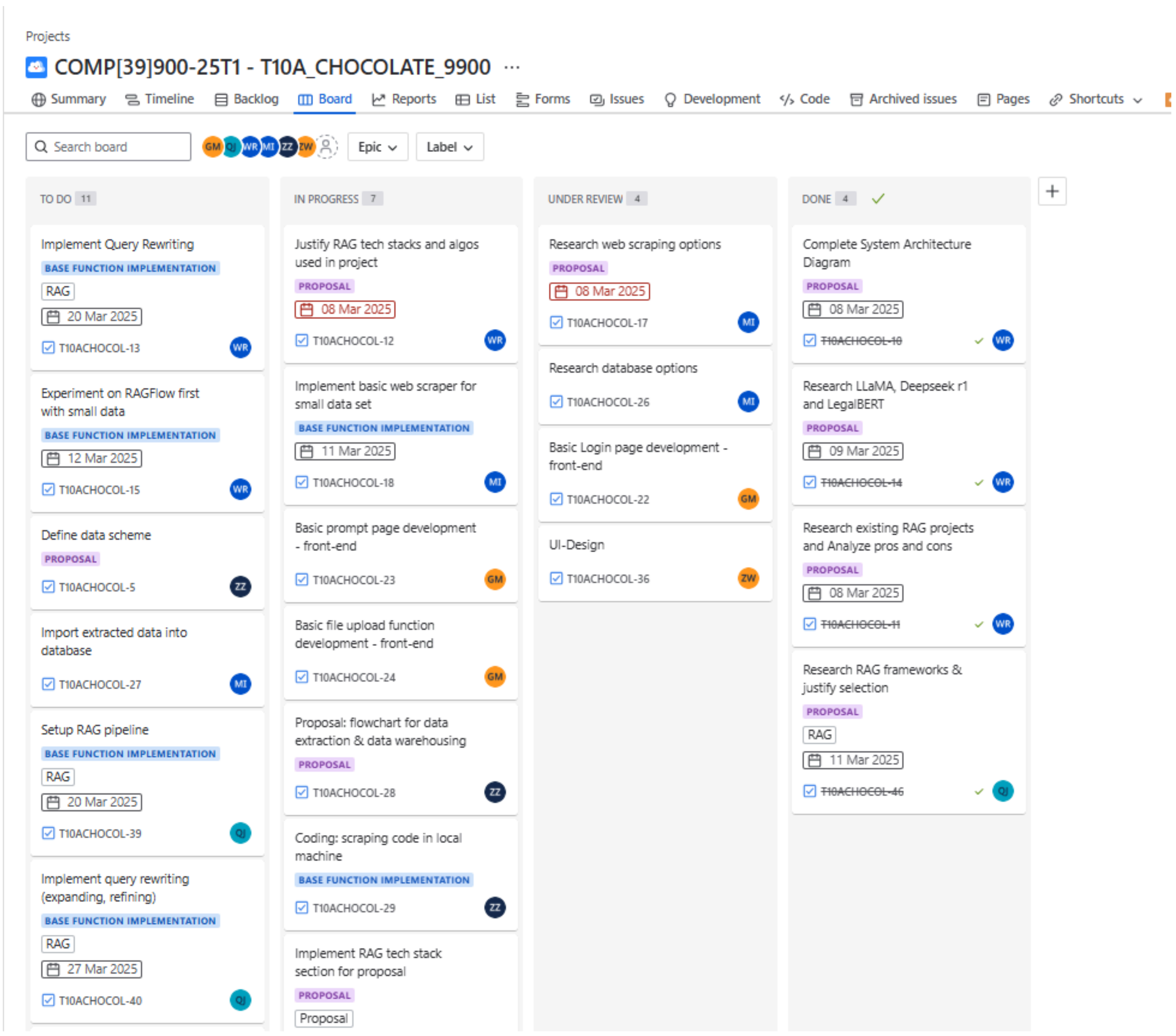
T10ACHOCOL-13	Implement Query Rewriting	BASE FUNCTION IMPL...	TO DO	20 Mar	=	WR
~~T10ACHOCOL-10~~	Complete System Architecture Diagram	PROPOSAL	DONE	08 Mar		WR
~~T10ACHOCOL-14~~	Research LLaMA, Deepseek r1 and LegalBERT	PROPOSAL	DONE	09 Mar		WR
~~T10ACHOCOL-11~~	Research existing RAG projects and Analyze pros and cons	PROPOSAL	DONE	08 Mar	=	WR
T10ACHOCOL-12	Justify RAG tech stacks and algos used in project	PROPOSAL	IN PROGRESS	08 Mar		WR
T10ACHOCOL-17	Research web scraping options	PROPOSAL	UNDER REVIEW	08 Mar		MI
T10ACHOCOL-18	Implement basic web scraper for small data set	BASE FUNCTION IMPL...	IN PROGRESS	11 Mar	=	MI
T10ACHOCOL-23	Basic prompt page development - front-end		IN PROGRESS			GM
T10ACHOCOL-24	Basic file upload function development - front-end		IN PROGRESS			GM
T10ACHOCOL-15	Experiment on RAGFlow first with small data	BASE FUNCTION IMPL...	TO DO	12 Mar	=	WR
T10ACHOCOL-5	Define data scheme	PROPOSAL	TO DO			ZZ
T10ACHOCOL-26	Research database options		UNDER REVIEW			MI
T10ACHOCOL-22	Basic Login page development - front-end		UNDER REVIEW			GM
T10ACHOCOL-36	UI-Design		UNDER REVIEW			ZW
T10ACHOCOL-27	Import extracted data into database		TO DO			MI
T10ACHOCOL-28	Proposal: flowchart for data extraction & data warehousing	PROPOSAL	IN PROGRESS			ZZ
T10ACHOCOL-29	Coding: scraping code in local machine	BASE FUNCTION IMPL...	IN PROGRESS		=	ZZ
T10ACHOCOL-39	Setup RAG pipeline	BASE FUNCTION IMPL...	TO DO	20 Mar	=	QJ
T10ACHOCOL-40	Implement query rewriting (expanding, refining)	BASE FUNCTION IMPL...	TO DO	27 Mar	=	QJ
T10ACHOCOL-37	Implement RAG tech stack section for proposal	PROPOSAL	IN PROGRESS	11 Mar		QJ
T10ACHOCOL-30	Coding: scraping realisation on AWS	PROPOSAL	TO DO			ZZ
T10ACHOCOL-41	Implement hybrid search (keyword + semantic search)	BASE FUNCTION IMPL...	TO DO	06 Apr		QJ
T10ACHOCOL-42	Implement multi-hop retrieval for complex legal queries	BASE FUNCTION IMPL...	TO DO	13 Apr	=	QJ
T10ACHOCOL-43	Integrate AI-generated case summaries	BASE FUNCTION IMPL...	TO DO	27 Apr		QJ
T10ACHOCOL-44	Benchmark RAG performance	BASE FUNCTION IMPL...	TO DO	04 May		QJ


T10ACHOC Sprint 3 4 Apr – 18 Apr (0 issues) — 0 0 0 — Start sprint

Backlog (13 issues) — 13 0 0 — Create sprint

T10ACHOCOL-1	As a website user, I want to be able to log in to the website so that only I can access my private data.		TO DO		
T10ACHOCOL-2	As a member of the General public, I want to search SAT decisions using natural language so that I can retrieve relevant cases and generate analysis reports.		TO DO		
T10ACHOCOL-3	As a legal practitioner, I want to be able to upload new SAT decision documents manually so that I can search these documents just like the default documents on the website.		TO DO		
T10ACHOCOL-4	As a lawyer, I want to be able to build claims based on past decisions and relevant legal precedents efficiently, so I can save time for more client-focused work.		TO DO		
T10ACHOCOL-6	As a judge, I want to be able to upload a current case to the website so I can quickly review previous decisions for hearing preparation.		TO DO		
T10ACHOCOL-16	Implement an Evaluation component for RAG result		TO DO		WR
T10ACHOCOL-19	Provide API endpoint for front-end		TO DO		WR
T10ACHOCOL-20	Connect RAG component with Database		TO DO		WR
T10ACHOCOL-21	Optimize model result		TO DO		WR
T10ACHOCOL-31	Implement web scraper		TO DO		MI
T10ACHOCOL-32	Process extracted data for storage in data base		TO DO		MI
T10ACHOCOL-38	Research RAG frameworks	PROPOSAL	TO DO	11 Mar	QJ
T10ACHOCOL-45	Research RAG frameworks & justify selection	PROPOSAL	TO DO	11 Mar	QJ

+ Create issue



Projects

COMP[39]900-25T1 - T10A_CHOCOLATE_9900 ···

TO DO 11

- Implement Query Rewriting — BASE FUNCTION IMPLEMENTATION — RAG — 20 Mar 2025 — T10ACHOCOL-13 — WR
- Experiment on RAGFlow first with small data — BASE FUNCTION IMPLEMENTATION — 12 Mar 2025 — T10ACHOCOL-15 — WR
- Define data scheme — PROPOSAL — T10ACHOCOL-5 — ZZ
- Import extracted data into database — T10ACHOCOL-27 — MI
- Setup RAG pipeline — BASE FUNCTION IMPLEMENTATION — RAG — 20 Mar 2025 — T10ACHOCOL-39 — QJ
- Implement query rewriting (expanding, refining) — BASE FUNCTION IMPLEMENTATION — RAG — 27 Mar 2025 — T10ACHOCOL-40 — QJ

IN PROGRESS 7

- Justify RAG tech stacks and algos used in project — PROPOSAL — 08 Mar 2025 — T10ACHOCOL-12 — WR
- Implement basic web scraper for small data set — BASE FUNCTION IMPLEMENTATION — 11 Mar 2025 — T10ACHOCOL-18 — MI
- Basic prompt page development - front-end — T10ACHOCOL-23 — GM
- Basic file upload function development - front-end — T10ACHOCOL-24 — GM
- Proposal: flowchart for data extraction & data warehousing — PROPOSAL — T10ACHOCOL-28 — ZZ
- Coding: scraping code in local machine — BASE FUNCTION IMPLEMENTATION — T10ACHOCOL-29 — ZZ
- Implement RAG tech stack section for proposal — PROPOSAL — Proposal

UNDER REVIEW 4

- Research web scraping options — PROPOSAL — 08 Mar 2025 — T10ACHOCOL-17 — MI
- Research database options — T10ACHOCOL-26 — MI
- Basic Login page development - front-end — T10ACHOCOL-22 — GM
- UI-Design — T10ACHOCOL-36 — ZW

DONE 4 ✓

- Complete System Architecture Diagram — PROPOSAL — 08 Mar 2025 — ~~T10ACHOCOL-10~~ ✓ WR
- Research LLaMA, Deepseek r1 and LegalBERT — PROPOSAL — 09 Mar 2025 — ~~T10ACHOCOL-14~~ ✓ WR
- Research existing RAG projects and Analyze pros and cons — PROPOSAL — 08 Mar 2025 — ~~T10ACHOCOL-11~~ ✓ WR
- Research RAG frameworks & justify selection — PROPOSAL — RAG — 11 Mar 2025 — ~~T10ACHOCOL-46~~ ✓ QJ

In this project we split our delivery into 6 parts:

1. Data Acquisition Layer
2. Data Processing & Embedding Generation Layer
3. Data Storage Layer
4. AI and RAG Integration Layer
5. API and Business Logic Layer
6. Web Frontend Layer

1. Data Acquisition Layer:

This layer includes an **Automated Web Scraper** and a **Manual Document Upload Module**. The web scraper serves as the website’s primary data source, which allows users to search keywords and retrieve

SAT decisions from automatically. Additionally, users can upload documents manually, then the back-end program will be processed and indexed in the same way as web-scraped content.

2. Data Processing & Embedding Generation Layer:
In this layer, the product should have a Document Parser & Data Cleaning function. The document parser extracts raw text from documents scraped from websites or uploaded from users, converts it into plain text, cleans document text, and normalizes metadata. The Embedding Generation Module also utilises AI models to convert text into vector embeddings. This module also supports similarity searches and contextual generation. Generated data will be integrated with Retrieval-Augmented Generation (RAG).

3. Data Storage Layer:
In this layer the system stores case metadata, documents, and other structured data in a database such as PostgreSQL or MongoDB. For embedding storage, the system will apply extensions such as FAISS or pgvector to make vector-based retrieval efficiently. The website also applies a similarity search function, using fast nearest-neighbor searches to identify context-relevant decisions based on the query user input.

4. AI and RAG Integration Layer:
In this layer, the program processes user input and generates AI-driven results based on SAT documents. It applies query embedding, converting natural language queries(user input) into vector embeddings using the same model as document embeddings. Searches the vector database to find the most relevant SAT decisions.
Then the decision content and metadata will be combined with the language model to generate contextually relevant summaries.
Additionally, the system applies a user feedback function to refine AI-generated summaries over time.

5. API and Business Logic Layer:

- Search API:
  Accepts natural language queries for user input, runs embedding generation and vector search functions, returns results with AI-generated summaries for front-end display.
- Upload API:
  Manages document uploads, embedding generation, and indexing into storage systems.
- User Management & Authentication:
  Manages user sessions, roles, and access rights to different functionalities.

6. Web Frontend Layer

- Search Interface:
  A web-based UI for users to enter natural language queries.
- Results Display:
  Presents decision details, metadata and AI-generated summaries.
- Admin Panel:
  A window to upload new documents and view system metrics.
- Responsive Design：
  Ensures the application is accessible and size dynamic across devices.

Timeline / Milestones of each sprint

Milestone 1: Base Function Implementation (Sprint 1) (week 3 - week 5)

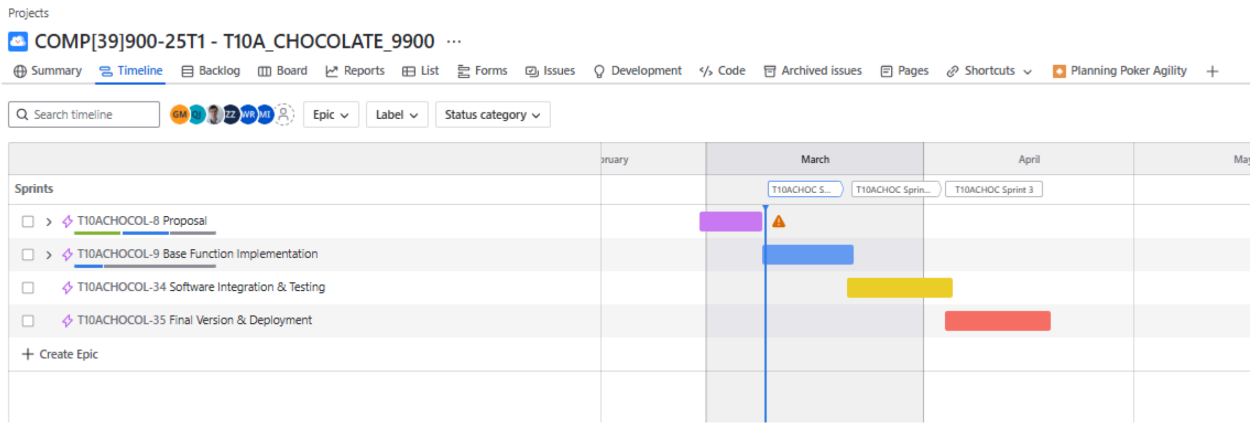
- **Front-End Development**: Complete all necessary front-end pages.
- **Database Setup**: Build all relational tables.
- **Web Scraper**: Scrape all base SAT documents, clean and classify the data, and store it in the database.
- **RAG Program**: Implement the base RAG (Retrieval-Augmented Generation) system with fundamental search functionality.
- *Note*: Hard coding is acceptable at this stage as individual components are developed separately.

Milestone 2: Software Integration & Testing (Sprint 2) (week 6 - week 8)

- **Website Version 1.0**: Integrate all components (front-end, back-end, database, and scraper) and deploy an initial version.
- **System Testing**: Conduct end-to-end testing with real-life data.
- **Client Feedback**: Collect opinions and plan for the next update based on user needs.

Milestone 3: Final Version & Deployment (Sprint 3) (week 8 -week 10)

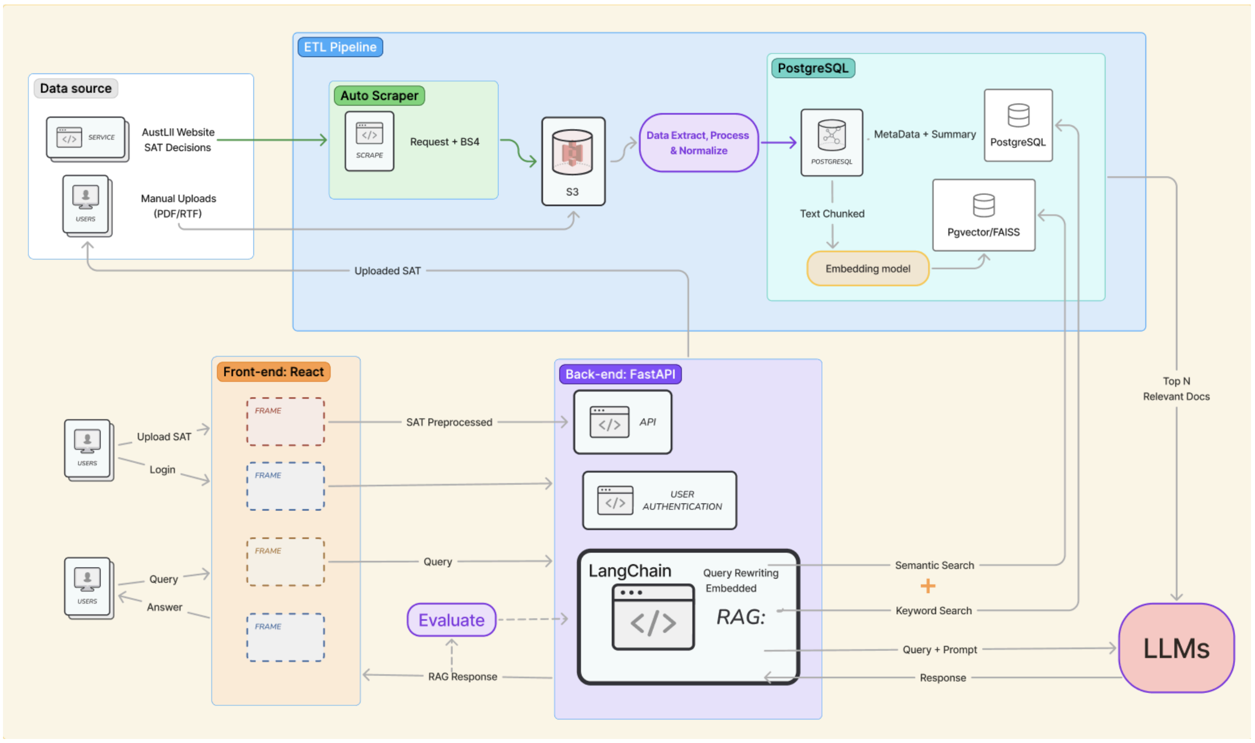
- **Bug Fixing**: Address defects found in version 1.0.
- **Feature Updates**: Implement necessary function enhancements.
- **Final Release**: Deploy the final version and conduct a product presentation.



T10ACHOCOL-8 Proposal
T10ACHOCOL-9 Base Function Implementation
T10ACHOCOL-34 Software Integration & Testing
T10ACHOCOL-35 Final Version & Deployment

Technical Design

System Architecture Diagram



Our project's overall design is depicted in the above diagram. Further detail is explored in the following sections.

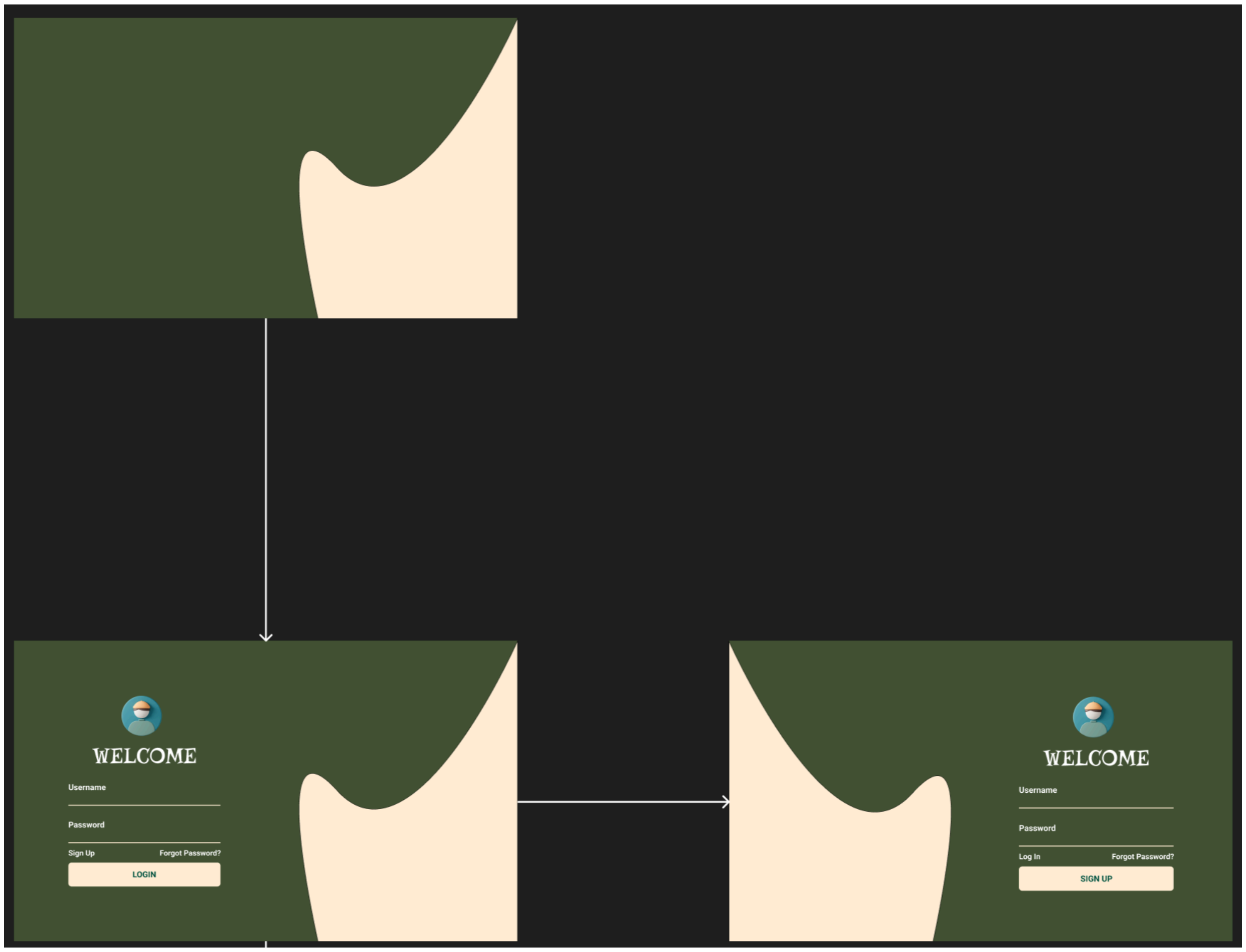
Frontend

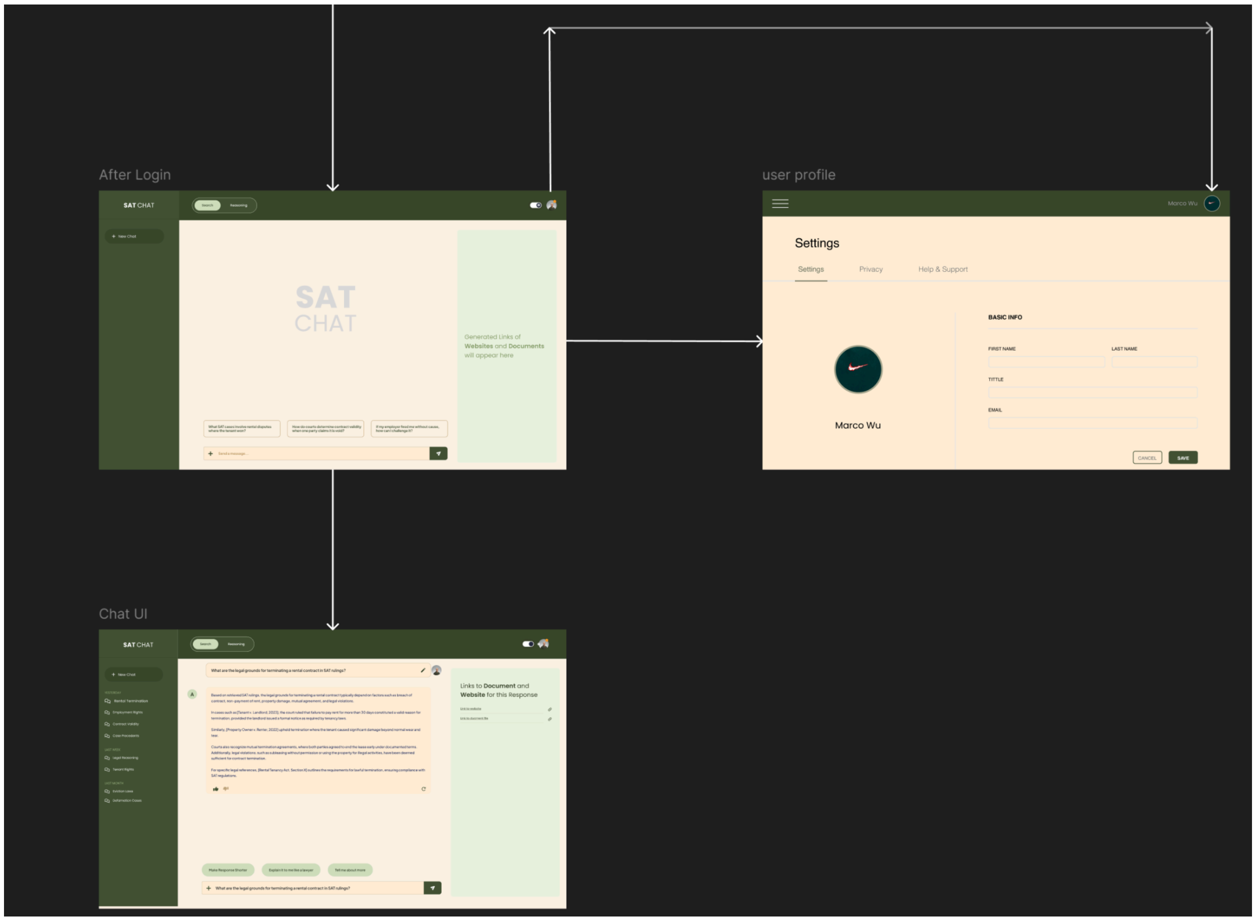
Users need an intuitive interface to get contextually relevant results and upload new SAT decision documents.

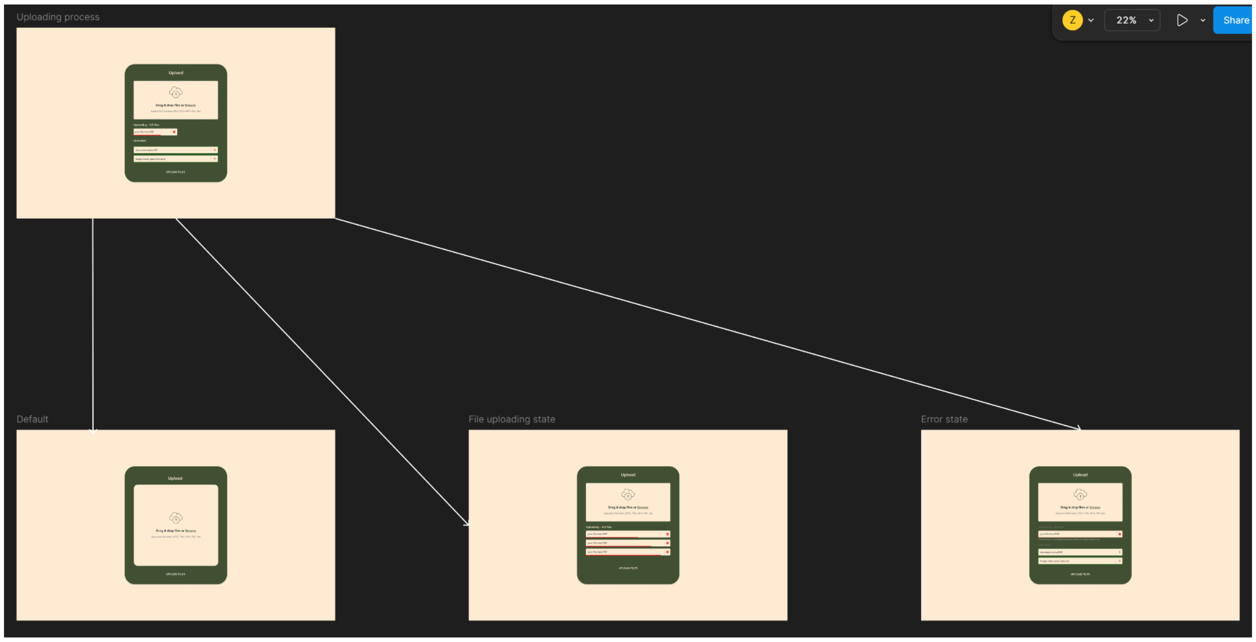
- Solution Plan: Use React based UI with the following functions:
    - Register / Login
    - Multi-types documents upload
    - Search result with relevant summaries and insights
    - Specific bar to show website links or pictures
    - Personal pages to manage user information
    - Show AI generated response after asking
- Alternatives:
    - Vue.js
    - Angular

- Justification: React is the most popular frontend framework in Australia. Furthermore, its ecosystem and ease of integration with FastAPI make it the best choice for this project.
- Novelty: AI-powered search assistance, AI-generated summaries, conversational UI for legal queries, real-time feedback on searching queries and document uploads.

Here is link of figma:
https://www.figma.com/design/JARSiB8reXAC1A1WJOfjhf/Website?node-id=0-1&m=dev&t=8cOlA2CBJUEKEn9Q-1

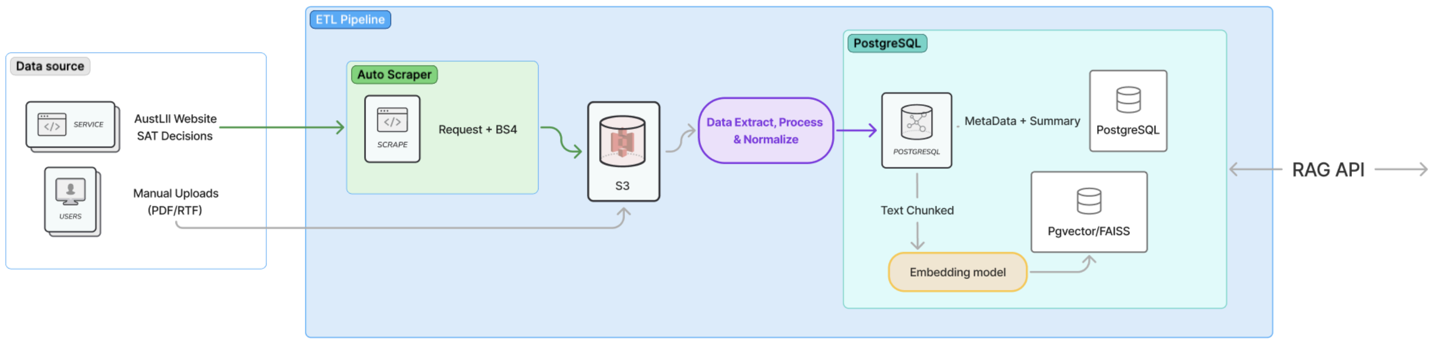






ETL pipeline design

Diagram



ETL pipeline overview

We intend to develop a data pipeline that can be quickly retrieved to support our RAG system.

Extract Layer

We will use AWS S3 as our data lake to store all raw information from two primary sources:

- Automated scraping from the SAT website (HTML documents)
- Manual uploads by authorized users (PDF/RTF/DOCX files)

These raw documents will be stored with appropriate naming conventions and metadata to maintain traceability and lineage.

Transform Layer

Then, in the transformation layer, before the 'REASONS FOR DECISION' section, we will save it into a structured database. Afterwards, the data is stored in a vector database for fast retrieving.

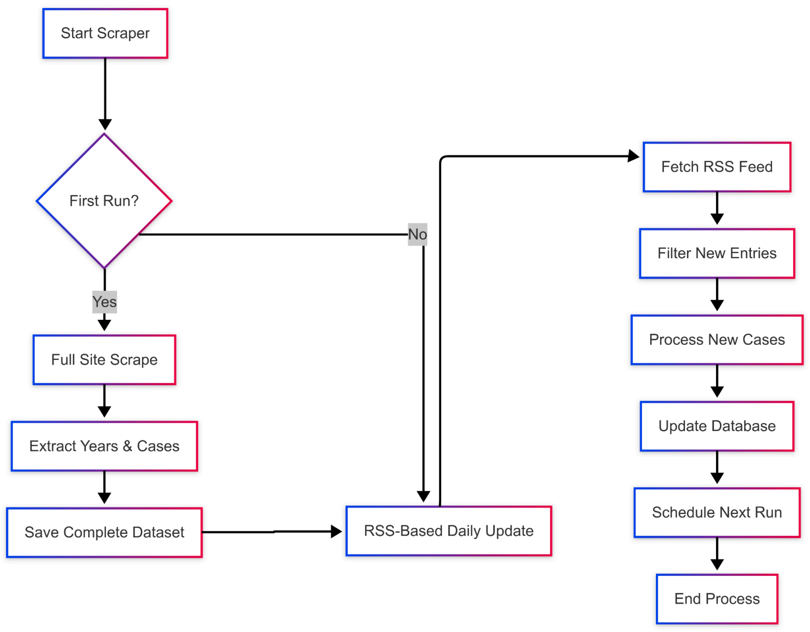
Load

Finally, in the load layer, for structured schema (PostgreSQL), metadata is stored in normalised tables (cases, members, parties, catchwords, legislation, and results). Vector schema (PostgreSQL with pgvector or using FAISS) stores full decision text.

This approach enables both precise filtering through SQL queries and semantic search through vector similarity, creating an optimal foundation for our RAG system. The Appendix A provides more comprehensive design diagrams.

WASAT Auto-Scraper System

Diagram



System overview

Our auto-scraping system consists of two parts.

- First is to extract data from all relevant web pages (if this is the first time running on our system).
- Afterwards, we will check the WASAT RSS feed to identify any changes.

This robust system ensures that we receive the latest data daily, ensuring we receive the most updated data and maintaining consistency of our scraping system. More detailed development process diagram is in appendix B.

Web Scraping

Web Scraping is the process of automatically extracting information from websites. The information can then be utilized by the user as input for their application. Web scrapers work by loading the HTML code of a webpage, parsing its content, and then extracting the relevant sections which is output to the user. Popular programming languages which can perform such tasks include Python, JavaScript (Node.js), Ruby, Java, PHP and C++. The discussion of the use of the aforementioned languages is detailed by Parsehub [3] and is summarized below.

1. **Python:** Simple language to code with many available libraries that can be used to achieve many programming tasks. In particular, the Python library called *BeautifulSoup* is specifically designed to parse HTML or XML webpages and can easily be implemented. *Scrapy* is a popular Python framework widely used for web scraping and web crawling.
2. **JavaScript (Node.js):** This language can be used to scrape live data and can support API and socket-based implementation.
3. **Ruby:** Easy to use with libraries such as Nokogiri which can parse HTML and others such as HTTParty and Pry that make the web scraping implementation simple.
4. **Java:** Widely used programming language suitable for web scraping with libraries such as JSoup which allows for extraction and manipulation of data from HTML and XML.
5. **PHP:** Typically used for web development hence has limited support for web scraping for complex project requirements.
6. **C++:** Complex language to learn; however is high performance, allows for memory management and access to many libraries. Despite these advantages, C++ is not an optimal choice for web scraping as it may have difficulty in parsing complex HTML.

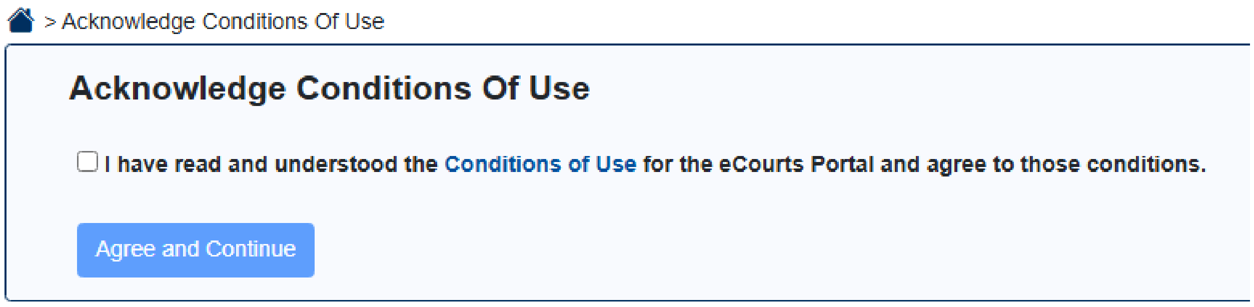
Since Python has most of the functionalities required for web scraping, beginner friendly, easy to use, and is relatively fast in comparison to its competitors, it has been selected for the web scraping portion of this project.

SAT Decision databases

As per project requirements, the application is required to automatically scrape, process and index SAT (State Administrative Tribunal) decisions. These SAT decisions can be readily accessed from Western Australia's government SAT Decisions database [5], and *Austlii* [6] which contains all decisions relevant to SAT across Australia. Comparisons between these two datasets are performed (with respect to website format and ease of access to relevant data) in order to select the website to extract the data from.

Western Australia's SAT Database [5]

The database first requires the user to acknowledge the websites Condition of Use which can complicate the development of the automatic web scraper.

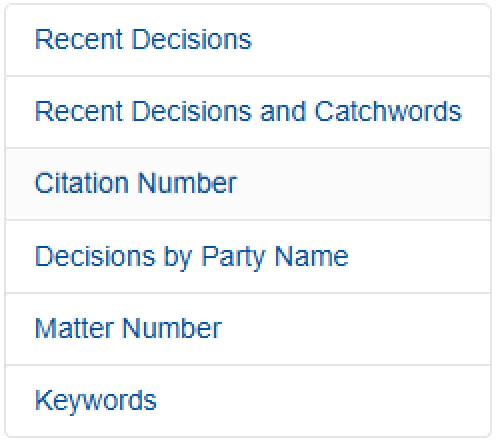


> Acknowledge Conditions Of Use

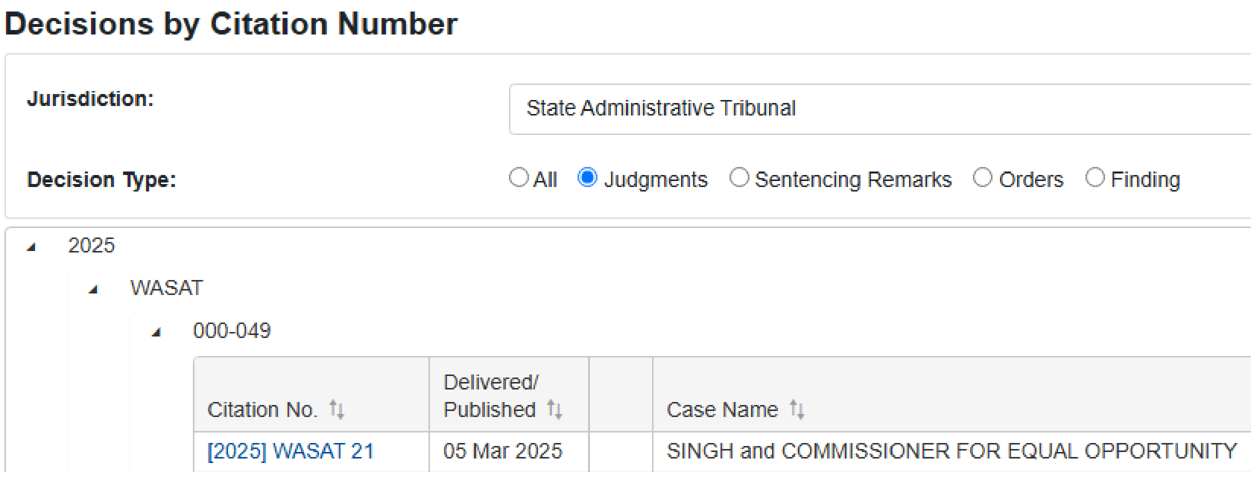
Acknowledge Conditions Of Use

☐ I have read and understood the Conditions of Use for the eCourts Portal and agree to those conditions.

The website offers many options to sort and filter decisions. Users can sort decisions based on ‘Recent Decisions’, ‘Recent Decisions and Catchwords’, ‘Citation Number’, ‘Decision by Party Name’, ‘Matter Number’ and ‘Keywords’ (as shown below).



The presented results can then be filtered based on ‘Decision Type’, where users can select from the following options: ‘Judgements’, ‘Sentencing Remarks’, ‘Orders’ and ‘Findings’. An example is shown below, where decisions are sorted by ‘Citation Number’ and the results are filtered to only include ‘Judgements’.



Decisions by Citation Number

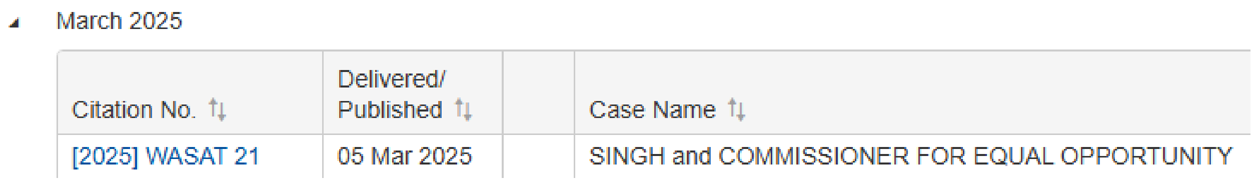
Jurisdiction: State Administrative Tribunal

Decision Type: All [✓ Judgments] Sentencing Remarks Orders Finding

2025
WASAT
000-049

Citation No.	Delivered/ Published		Case Name
[2025] WASAT 21	05 Mar 2025		SINGH and COMMISSIONER FOR EQUAL OPPORTUNITY

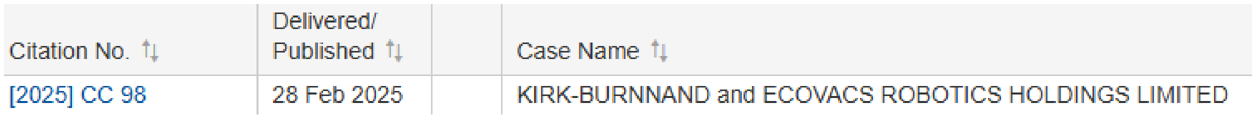
By inspection, the ‘Citation Number’ of a case follows a different format depending on the ‘Decision Type’ of the case. In general, ‘Citation Number’ is formatted as ‘[Year] [Type] [Number]’. In particular ‘Judgements’ follow the format of [Year] WASAT [Number]



March 2025

Citation No.	Delivered/ Published		Case Name
[2025] WASAT 21	05 Mar 2025		SINGH and COMMISSIONER FOR EQUAL OPPORTUNITY

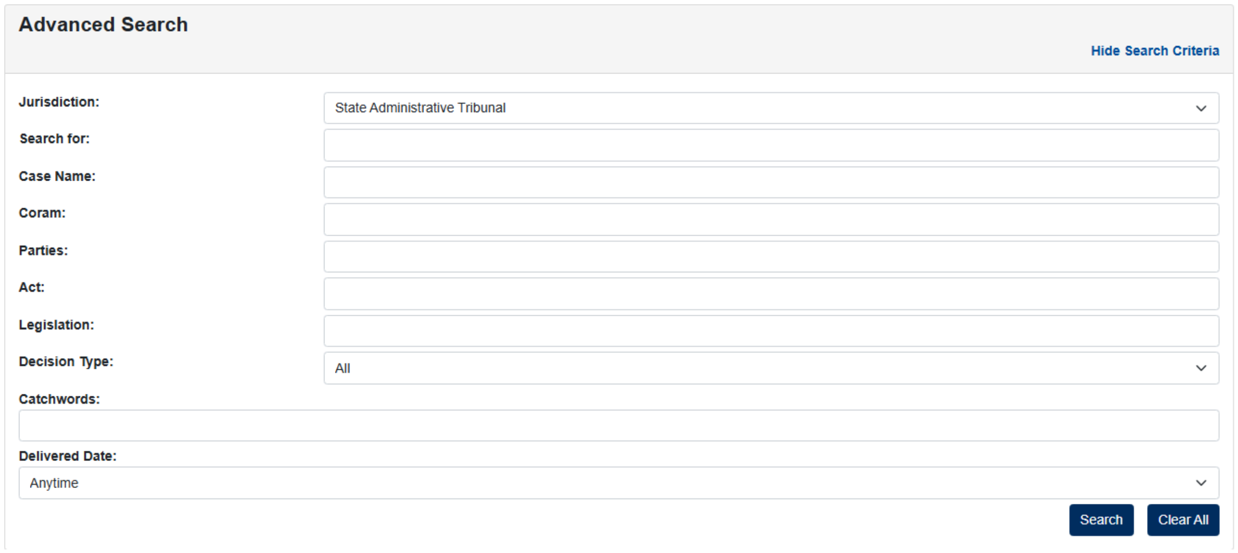
and ‘Orders’ are categorised as either ‘CC’, ‘DR’, ‘EOA’, ‘SAT Act’ and ‘VR’ which follows a similar format to the above.



Citation No.	Delivered/ Published		Case Name
[2025] CC 98	28 Feb 2025		KIRK-BURNNAND and ECOVACS ROBOTICS HOLDINGS LIMITED

It is observed that there is no data for ‘Sentencing Remarks’ and ‘Finding’ decision types.

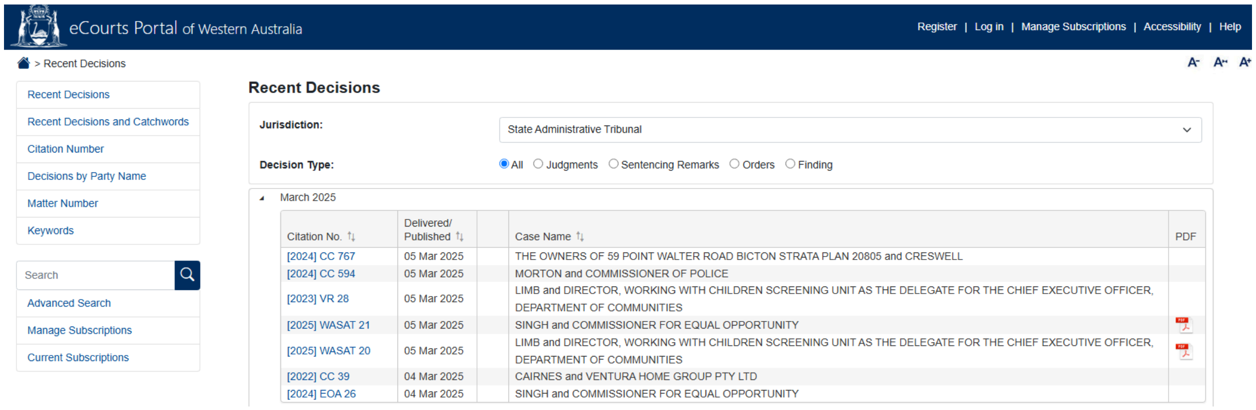
Furthermore, an advanced search is also available on the website to refine the search parameters as shown in the Figure below:



Advanced Search

Jurisdiction:	State Administrative Tribunal
Search for:	
Case Name:	
Coram:	
Parties:	
Act:	
Legislation:	
Decision Type:	All
Catchwords:	
Delivered Date:	Anytime

With respect to the presentation of the data, hyperlinks to the individual SAT decisions are shown to the user which can be directly accessed. Accessing a SAT decision presents all information regarding that case.



Recent Decisions

Jurisdiction: State Administrative Tribunal

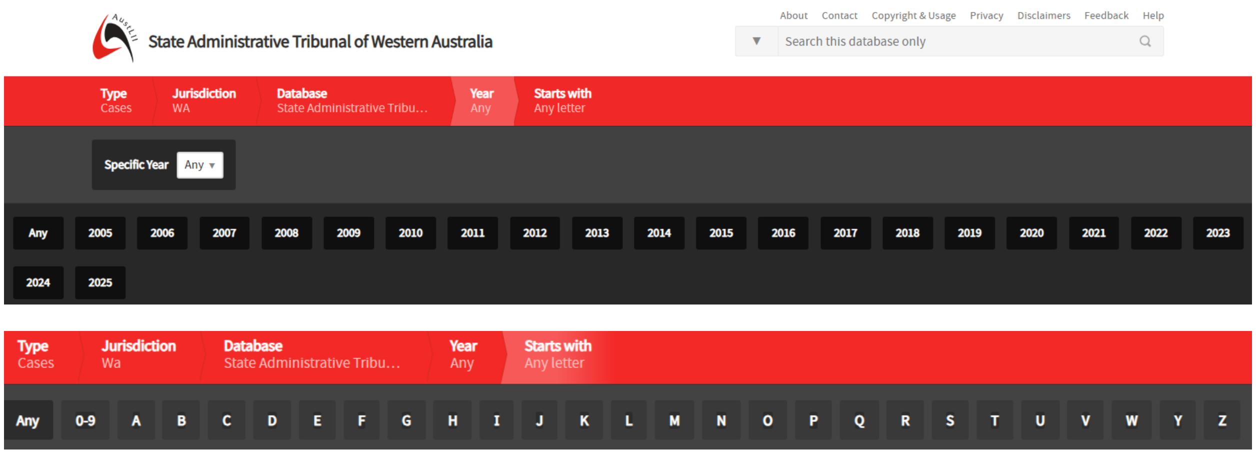
Decision Type: [✓ All] Judgments Sentencing Remarks Orders Finding

March 2025

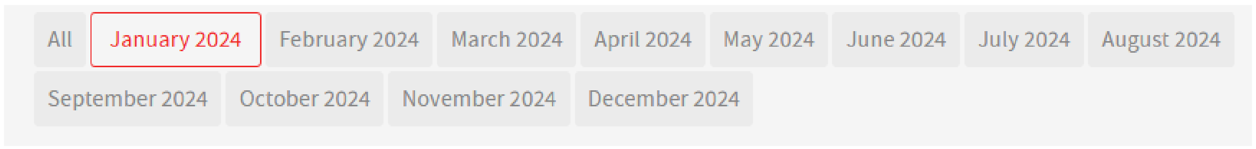
Citation No.	Delivered/ Published	Case Name	PDF
[2024] CC 767	05 Mar 2025	THE OWNERS OF 59 POINT WALTER ROAD BICTON STRATA PLAN 20805 and CRESWELL	
[2024] CC 594	05 Mar 2025	MORTON and COMMISSIONER OF POLICE	
[2023] VR 28	05 Mar 2025	LIMB and DIRECTOR, WORKING WITH CHILDREN SCREENING UNIT AS THE DELEGATE FOR THE CHIEF EXECUTIVE OFFICER, DEPARTMENT OF COMMUNITIES	
[2025] WASAT 21	05 Mar 2025	SINGH and COMMISSIONER FOR EQUAL OPPORTUNITY	📄
[2025] WASAT 20	05 Mar 2025	LIMB and DIRECTOR, WORKING WITH CHILDREN SCREENING UNIT AS THE DELEGATE FOR THE CHIEF EXECUTIVE OFFICER, DEPARTMENT OF COMMUNITIES	📄
[2022] CC 39	04 Mar 2025	CAIRNES and VENTURA HOME GROUP PTY LTD	
[2024] EOA 26	04 Mar 2025	SINGH and COMMISSIONER FOR EQUAL OPPORTUNITY	

Austlii Database [6]

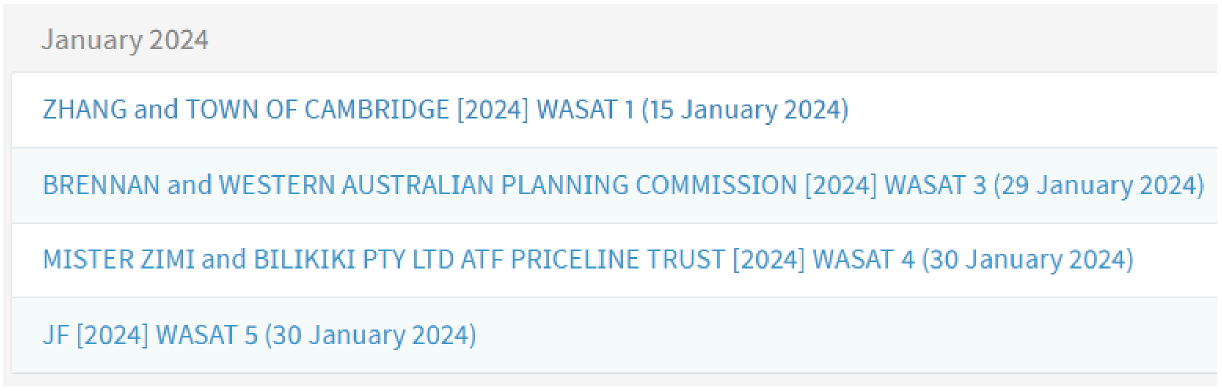
The Austlii database offers a simpler representation of the data with less filtering capabilities but is shown in a more user-friendly way which is more intuitive but also still contains all the relevant information. SAT decisions can be filtered by year and by letter.


About Contact Copyright & Usage Privacy Disclaimers Feedback Help
State Administrative Tribunal of Western Australia
Search this database only
Type Cases
Jurisdiction WA
Database State Administrative Tribu…
Year Any
Starts with Any letter
Specific Year Any
Any 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020 2021 2022 2023 2024 2025
Type Cases
Jurisdiction Wa
Database State Administrative Tribu…
Year Any
Starts with Any letter
Any 0-9 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

For a select year, data can be further filtered based on the month.



With respect to the presentation of the data, hyperlinks to the individual SAT decisions are shown to the user which can be directly accessed. Accessing a SAT decision presents all information regarding that case. Information for each of the SAT decisions is also available in PDF or RTF format which can be readily downloaded.



January 2024

ZHANG and TOWN OF CAMBRIDGE [2024] WASAT 1 (15 January 2024)

BRENNAN and WESTERN AUSTRALIAN PLANNING COMMISSION [2024] WASAT 3 (29 January 2024)

MISTER ZIMI and BILIKIKI PTY LTD ATF PRICELINE TRUST [2024] WASAT 4 (30 January 2024)

JF [2024] WASAT 5 (30 January 2024)

Western Australia's SAT Database [5] versus Austlii Database [6]

From the above analysis of Western Australia's SAT database and the Austlii database, it is noted that the former contains many features to refine search parameters that are useless for the scope of the web scraper. It also presents data in the form of nested tables making the HTML of the dataset difficult to interpret. This will make the implementation of the web scraper more difficult since SAT decisions are hidden behind many filters and layers of HTML tags. Additionally, accessing the database requires the user to confirm the website's terms and conditions which makes the implementation of the web scraper more complex. Furthermore, as the project requirements pertain to SAT decisions, Western Australia's SAT database includes other decision types such as 'Orders' (as described previously). Based on observation, these types of case data generally do not provide any contextual information of the case and the corresponding decision. Since, we desire to build an application which can gain insight from similar case studies, this data which provides limited information is not very useful for our design and is excluded from our data set. In light of this, we are only interested in accessing the 'WASAT' type decisions.

In comparison, the HTML of the Austlii database is less complex and easy to understand. Information is presented much more simply which will make the web scraping process simpler. Furthermore, the database exclusively only contains 'WASAT' decisions and presents a simpler organisation of data that can easily be filtered by year. The simpler design of the web page allows for easier implementation of the web scraper and hence has been selected for our dataset. Furthermore, since SAT decisions can be filtered by year, this enables flexibility in the amount of data to web scrape.

Scraping options

As discussed, the Austlii database will be used to scrape the desired SAT decision data. Now, since the data is available in different formats namely HTML, PDF and RTF, consideration is made on the most optimal way to scrape the data:

1. **HTML**: The structure of HTML is well defined and relevant elements can be directly accessed by their tag, class or ID. Furthermore, Python offers many libraries specially built for parsing and extracting information from HTML, thus simplifying the web scraping implementation.
2. **PDF:** Scraping information from a PDF is relatively more difficult in comparison to scraping a HTML since they inherently do not follow a defined structure. Also, since these PDFs will need to be downloaded, this will introduce an additional step in the scraping process which could worsen performance of the web scraper.
3. **RTF:** Similar reasoning to that of PDF.

Due to the predictable structure of HTML and ability to readily reference elements based on their tag, class or ID, data can be extracted more efficiently in comparison to the other methods and hence is the main method of extracting information.

Scrapy versus BeautifulSoup:

Comparison is made between the available options to extract the available data. For the purposes of this task, available tools such as BeautifulSoup and Scrapy are compared. Such tools are discussed and evaluated by *Zenrows* [7] which is summarised below.

1. **BeautifulSoup:**
    a. Simple to scrape static HTML content from a single web page quickly. However, multiple web pages will need to be scraped one at a time.
    b. Relies on other libraries such as pandas to export data.
    c. Simpler to implement and easy to learn
2. **Scrapy:**
    a. Ideal for scraping information from multiple web pages concurrently due to its asynchronous design.
    b. Supports exporting data into CSV, JSON or XML format.
    c. Difficult for beginners.

Since there are approximately 4000 documents in the Austlii database, using BeautifulSoup to extract the relevant data from each web page will be a very time consuming process. Hence, for this project, Scrapy will be used due to its ability to scrape multiple web pages concurrently and native functionality to export data into useful file formats.

Data acquisition and processing

As per project requirements, the web scraper is required to scrape and store SAT decision metadata. The extracted data should be informative and easily categorised. As per project specifications, the following are extracted for each of the cases:

1. Date/time of extraction
2. Case URL
3. Date of case
4. Case title
5. Involved parties
6. All sections under ‘Reasons for decision of the Tribunal’

We understand that due to the large amount of text available for each of the cases, the storage of metadata may become a challenge as more data is stored in our chosen databases. However, such textual information is needed for the application to retrieve the most relevant cases and generate the most appropriate responses for the intended users.

Databases

As mentioned previously, metadata, document references and other structured data extracted during the web scraping process will be stored in a database. Below are comparisons between some databases that are commonly used.

MongoDB versus PostgreSQL

AWS [10] provides a comparison between MongoDB and PostgreSQL which is summarised below .

MongoDB	PostgreSQL
Non-relational database	Object relational database
Stores data as JSON documents	Stores data as tables with rows and columns
Allows for fast retrieval replication and analysis	Supports many data types, scalability, concurrency, and data integrity for structured data
Can store unstructured and dynamic data	Stores structured data following a predefined schema
Uses MongoDB Query Language (MQL) for querying the database	Uses SQL and PostgreSQL for querying the database.

Supports indexing for fast retrieval of data.	Supports indexing for fast retrieval of data.
Data available even during server outage by duplicating data.	Data available even during server outage by duplicating data.
High scalability	High scalability
No relationships between collections. Querying the data is already optimised.	Relationships between tables can be made so that queries can be made across multiple tables.
ACID compliant for later versions	Offers ACID compliance for high levels of data consistency.

Based on the comparison between these different databases, since the SAT decision database contains structured metadata, PostgreSQL will be used due to its capability to store this structured data. Furthermore, PostgreSQL offers ACID compliance meaning that data will remain consistent and will handle issues regarding duplicate data as per project requirements. It also supports efficient searching of documents by indexing words and phrases [11].

FAISS versus Pinecone versus Pgvector

Vector databases store data represented as vectors which serve as mathematical representation of features. These databases are typically for AI applications due to its ability to handle complex information such as text and images as well as its efficient similarity search capabilities [12]. Discussion on the use of vector database is covered by Tejashri R.P [13] and *MyScale* [12], in particular they discuss and compare the use of vector databases FAISS and Pinecone which is summarised below. Furthermore, we have found that Pgvector is a built-in database in PostgreSQL, hence we also investigate its properties.

FAISS	**Pinecone**	**Pgvector**
Open source	Cloud vector database	PostgreSQL extension for vector search
Supports indexing for fast retrieval of data	Suitable for applications requiring natural language processing	Supports efficient vector operations directly in PostgreSQL
Can utilise GPU to accelerate search operations	User-friendly and is highly scalable.	Integrates seamlessly with existing PostgreSQL setup
Less user-friendly and requires more technical expertise.	Greater performance for vector search	Low overhead for small to medium datasets

Highly optimised algorithms for exact search and nearest neighbour search for large datasets. Optimised for speed and memory		Efficient for vector search within relational data
Able to store large volumes of high-dimensional data		Suitable for embedding models integrated with relational queries

Database summary

Below is a summary of the adopted databases for the efficient storage of metadata and embeddings for this project.

- Solution Plan: Since there are two types of data (metadata and text) that we need to deal with, we will implement a hybrid database architecture that includes a relational database (PostgreSQL) for structured data and either FAISS or Pgvector for unstructured data to ensure efficient storage, retrieval, and querying of SAT tribunal decisions.
- Alternatives:
    - NoSQL / MongoDB- More suitable for unstructured data.
    - Elasticsearch - Not optimized for high-dimensional embeddings.
- Justification: Metadata is highly structured, requiring relational queries for case filtering, making PostgreSQL a better choice. FAISS and Pgvector are efficient for fast and scalable retrieval of similar cases based on AI-driven vector search.
- Novelty: Hybrid storage for legal case retrieval and allows users to find similar cases with AI rather than just relying on keyword search.

Backend

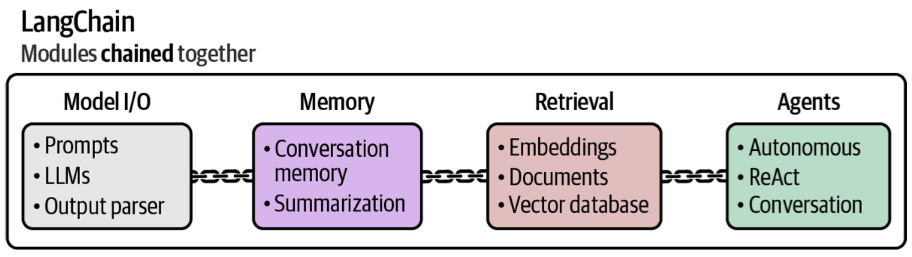
Handle authentication, search queries, and API endpoints.

- Solution Plan: We will use FastAPI as the backend framework to handle API requests, manage user authentication, and facilitate communication between the frontend, database and RAG module.
    - Process user queries - Receive search requests from frontend, search relevant metadata and text from PostgreSQL and FAISS and return AI generated summaries/insights.
    - Manage document uploads - Accept and preprocess SAT decision documents, store metadata into PostgreSQL and generate vector embedding for FAISS retrieval.

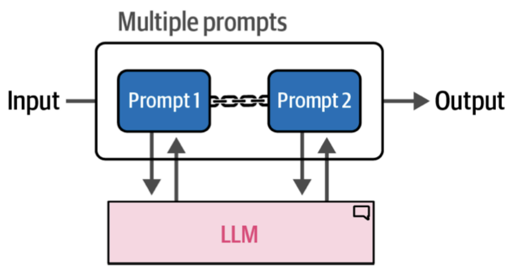
  - Expose APIs for frontend - Provide endpoints for search, document uploads, authentication and AI generated response.
  - Optimize performance - Handle multiple requests efficiently and ensure low latency response.
  - Integrate with AI services - Forward search queries to RAG pipeline, retrieve response and send AI generated summaries/insights back to users.
- Alternatives:
  - Flask - Does not natively support asynchronous operations.
  - Django - Heavyweight and higher cost.
  - Node.js - Requires Javascript/TypeScript backend development and Python-based AI/ML libraries are not natively supported.
- Justification: FastAPI is built for high-performance API development, supporting asynchronous processing. Optimised for machine learning and AI integration, making it a great fit for RAG-based AI retrieval. Lightweight yet scalable – allows for easy expansion without unnecessary overhead.
- Novelty: Unlike traditional REST APIs, our backend is designed for AI-driven retrieval, integrating vector search (vector DB) and AI summarization seamlessly.

RAG(Retrieval-Augmented-Generation)

- LangChain (Connect different components):



LangChain provides modularity, supports multiple embedding and model backends, and allows fine-tuning retrieval. In addition, the client highlights the model's reasoning ability and would prefer multi-agent when reasoning. LangChain has a React framework that we can use to implement multi-step reasoning.



For multi-step reasoning, we chain with multiple prompts: “With sequential chains, the output of a prompt is used as the input for the next prompt. For example, we extract key legal entities with the first prompt, and feed the result into the second prompt which can retrieve relevant cases for these legal entities” [1].

- Query Rewriting (Query Expansion + Generation) or Template Input:
  Users may input unclear or incomplete queries, making it hard to retrieve relevant cases. Implementing query rewriting techniques like synonym expansion, paraphrasing, structured query transformation can improve searching efficiency and relevance. On the other hand, the client mentioned we may define a structured template for users (judge) to input. We use this structured application file to search similar cases.
  Query rewriting can be implemented through a prompt and an API call, such as Cohere’s API, which provides a dedicated query rewriting mode for co.chat, helping users refine their queries.
- Query Routing - Simple Search versus Deep Research with Reasoning:
  Some queries require instant results like finding similar cases or a summary of some cases, while others demand deeper legal research or multiple reasoning steps. So we plan to implement a routing phase that distinguishes between simple searches and in-depth multi-step reasoning queries.
  Query routing reduces unnecessary computation by directing straightforward queries to direct search (single-step) retrieval (e.g direct to specific knowledge base which is only one specific category of SAT decisions) while engaging reasoning LLMs for complex reasoning-based searches [1].
  A query classifier determines whether to use simple search for fast retrieval and semantic search for deeper analysis, which ensures optimal balance between speed and depth.
- Hybrid Search - Keyword Search + Semantic Search:
  Keyword searches may lack contextual understanding, while pure semantic searches may retrieve loosely related cases. For both simple query and complex query, In order to combine the strengths of both, we propose to use BM25 for keyword search and vector DB for semantic search to avoid

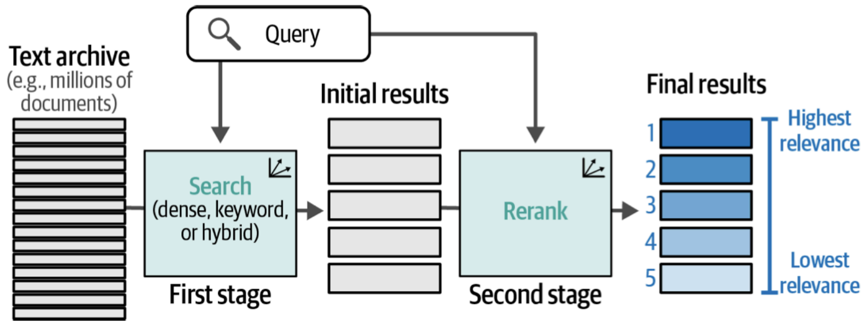
missing valuable information. More evaluations will be made to compare different search results [2].

- Prompt Engineering
  Users will input different types of queries, so well-structured prompts will be tailored accordingly. To enhance legal reasoning and accuracy, we will combine In-Context Learning (ICL), Chain Prompting, Chain-of-Thought (CoT) reasoning, and Self-Consistency Sampling to improve response quality [1].
  Since legal queries require factual accuracy, hallucination mitigation will be a priority. The model will be prompted to respond professionally and clearly, leveraging CoT prompting for step-by-step legal reasoning and self-consistency sampling to ensure more stable and reliable outputs.
  To further improve response quality, our prompts will be evaluated and optimized through iterative testing, ensuring they align with SAT decisions and maintain legal precision.

- Multi-hop RAG
  Some legal queries require multiple retrieval steps to provide accurate answers. For example, a query like: "What SAT decisions in 2023 involved environmental regulations and what were their outcomes?" To answer this, the system must first retrieve relevant cases, then extract their outcomes. Without Multi-hop RAG, a single retrieval step may return incomplete or disconnected results, making it difficult for users to get correct information [1].
  To implement Multi-hop RAG, the system first analyzes the user query to determine if multiple retrieval steps are needed, using either an LLM-based query parser or predefined rules. It retrieves initial results, generates follow-up queries to extract case outcomes, and performs secondary retrieval. Finally, the AI model synthesizes the results into a cohesive and contextualized response.
- Multi-query RAG
  Multi-query RAG improves retrieval by splitting a single query into multiple related searches when one query alone may not provide a complete answer. For example, if a user asks, "Compare SAT decisions on property disputes from 2020 and 2023," a single search might not retrieve both years' cases in one document. Instead, the system runs two separate searches:
  Query 1: "SAT property dispute decisions 2020"
  Query 2: "SAT property dispute decisions 2023"

This ensures comprehensive case retrieval, allowing the AI model to generate a well-grounded comparative summary. Additionally, multi-query RAG can determine when a search is unnecessary—if the model already has enough knowledge to generate a confident response.
To implement Multi-query RAG, the system first analyzes the query and determines if multiple searches are needed. If so, it generates parallel sub-queries, each targeting a specific aspect. The system fetches results for each query separately and provides them to the AI model for context-aware summarization and comparison [1].

- Reranking



Reranking is used for changing the order of the output results based on relevance to the search query. Search accuracy is crucial for this application since users rely on the system to find the most relevant tribunal decisions. Though BM25 or vector-based retrieval methods can fetch related documents, the top ranked results may not be the most useful ones. Therefore we need reranking to ensure that the most relevant and contextually important SAT decisions appear at the top [1].
To implement reranking, our two-stage ranking process includes:
  - Use BM25 (for keyword-based retrieval) or vector search (for semantic similarity) to fetch the top N relevant documents.
  - Apply ML-based reranking to reorder the top N results based on deeper understanding such as query relevance, case importance, and recency.

- Fine-tuning

Fine-tuning is the key step in adapting a model to a specific dataset or domain. Our system prioritizes RAG and prompt engineering to enhance QwQ-32B's capability in processing SAT legal case retrieval. However, if RAG fails to produce legally accurate and coherent outputs, fine-tuning QwQ-32B might be necessary.
Fine-tuning could be required under the following circumstances:

  - QwQ-32B is primarily trained on Chinese legal texts, which differ significantly from Australian SAT legal frameworks. Without adaptation, the model may misinterpret legal concepts or fail to correctly apply Australian legal precedents.
  - If retrieved case law from RAG varies in structure, legal terminology, or reasoning style, QwQ-32B may struggle to generate clear, standardized, and logically sound legal summaries. Fine-tuning can help align output format with professional legal drafting standards.

  Therefore, in cases where fine-tuning is needed, we propose adapting QwQ-32B by using low-rank adaptation (LoRA), which is one of the most widely used and effective techniques for parameter-efficient fine-tuning (PEFT) [2].

- Evaluation & Performance Metrics:

  The RAG evaluation usually focuses on how we assess the effectiveness, accuracy, and reliability of the retrieval and generation process. The key aspects to evaluate include: retrieval quality, response quality, verifiability, and efficiency [19].

  - Retrieval quality measures how well our system retrieves relevant SAT legal cases. Key metrics include Recall@K and Mean Reciprocal Rank (MRR), nDCG (Normalized Discounted Cumulative Gain). Our target benchmarks include Recall@5 ≥ 85% and MRR ≥ 0.75, ensuring highly relevant legal cases are retrieved efficiently.
  - Response quality evaluates the legal accuracy and coherence of AI-generated answers, and will be measured using BLEU, ROUGE, and factual consistency, ensuring AI-generated legal answers align with retrieved cases and expert-reviewed legal standards.
  - Verifiability and citation accuracy are critical for maintaining trust in AI-generated legal responses. We use citation recall, precision and F1 score to balance these two measures, ensuring that AI-generated legal references are both relevant and precise. Our goal is to achieve citation recall ≥ 60% and citation precision ≥ 80%.
  - Efficiency and latency determine whether the system can provide real-time legal case retrieval and response generation. We aim for query latency under 2 seconds and token generation speed of at least 20 tokens per second to maintain real-time legal retrieval.

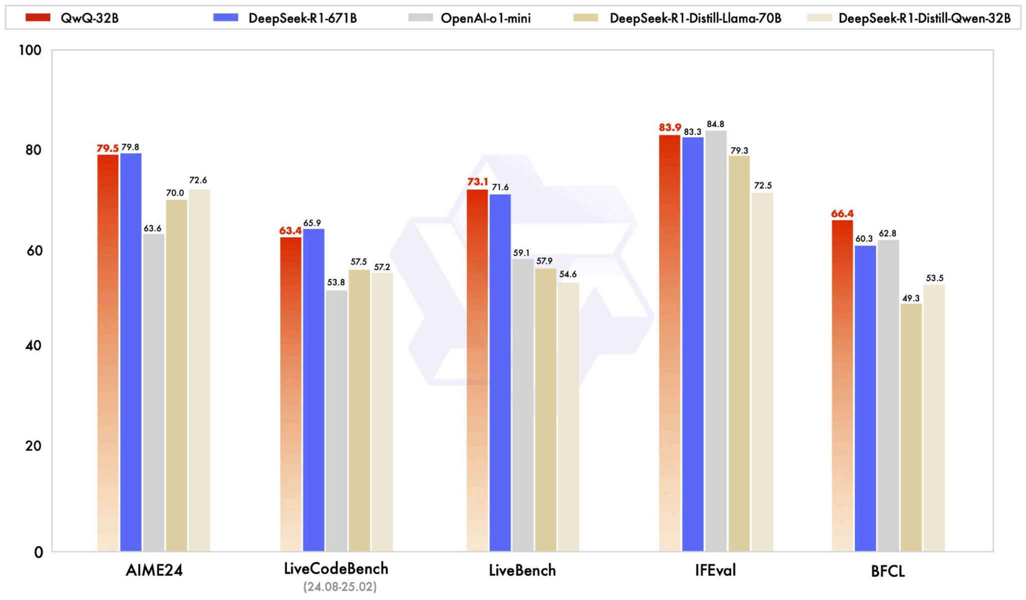
- LLMs Selection

  Regarding LLMs selection, client proposed to use three LLMs which are Llama 3.1 Model, DeepSeek R1 (R2 in the near future) Model, Legal_BERT (https://opensource.legal/projects/Legal_BERT) and QwQ-32B. We consider two main use cases which are simple search summary and complex reasoning tasks. More models are considered and compared in the table below:

Metric	**DeepSeek R1**	**Llama 3.1 405B**	**LegalBERT**	**QwQ-32B**	**Claude Sonnet 3.7**	**Claude Sonnet 3.7 Thinker**
Release Date	Jan 21, 2025	Jul 23, 2024	2020	Mar 6, 2025	Feb 24, 2025	Feb 24, 2025
Parameters	671B (37B active)	405B	110M	32B	Not publicly disclosed	Not publicly disclosed
Context Window	128K tokens	128K tokens	512 tokens	32K tokens	200K tokens	200K tokens
Open Source	Yes	Yes	Yes	Yes	No	No
Input Cost per 1M Tokens	$0.75 on Deepinfra	$0.8 on Deepinfra	N/A	$0.12 on Deepinfra	$3 on Anthropic	$3 on Anthropic
Output Cost per 1M Tokens	$2.4 on Deepinfra	$0.8 on Deepinfra	N/A	$0.18 on Deepinfra	$15 on Anthropic	$15 on Anthropic
Reasoning Model	Yes	No	No	Yes	Hybrid reasoning model	Yes

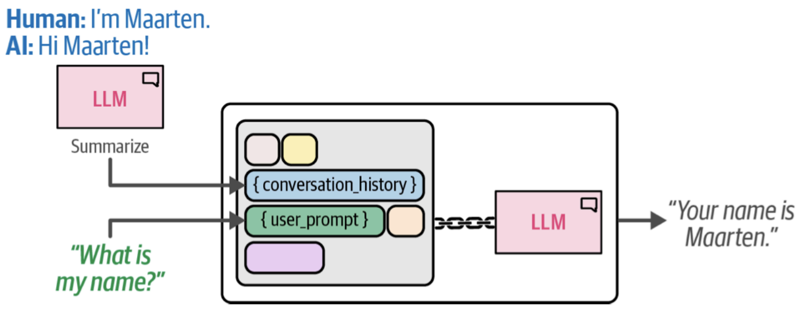
Cost references and comparison of other benchmarks can be found on llm-stats.com [17].

It should be noted that a new light reasoning model called QwQ-32b [18] was just released. It can achieve significantly enhanced performance in downstream tasks, especially hard problems. QwQ-32B is a medium-sized reasoning model which is capable of achieving competitive performance against state-of-the-art reasoning models such as DeepSeek-R1, o1-mini.



So we propose to use a general model without reasoning capability to handle simple query which is efficient while using a reasoning model to handle complex multi-steps query. Different models' performance will be evaluated and compared in order to find the optimal one.

- Conversational AI with Context Memory:



Users often conduct iterative searches and follow-up queries that require keeping prior context. To address this, we propose implementing session-based memory, enabling the system to recall user query history, search results, and LLM-generated responses.

To enhance query refinement, we integrate conversation memory, where the LLM receives a chat history summary alongside new queries. This approach allows users to refine searches progressively. For example, if a user first asks, "Please help find eviction cases from 2021" and follows up with, "Focus on those involving commercial leases," the system will recall the previous query and refine results accordingly.

If a query is a follow-up, the system will retrieve prior search results, compare them with the new query, and apply logical reasoning to generate a refined output. Maintaining conversational context enables progressive search refinement, reduces redundant queries, and improves usability.

To prevent excessive memory usage, we implement session expiration strategies, where only the last 2-5 interactions are retained. This ensures optimal performance while maintaining an effective query refinement process.

User Stories Address Project Objectives

Project Objectives	User Stories/Tasks Addressing it
Scrape SAT data and build a SAT database	• T10ACHOCOL-17 Research web scraping options • T10ACHOCOL-18 Implement basic web scraper for small data set • T10ACHOCOL-24 Define data scheme • T10ACHOCOL-27 Import extracted data into the database • T10ACHOCOL-26 Research database options • T10ACHOCOL-27 Import extracted data into the database • T10ACHOCOL-28 Proposals flowchart for data extraction & data warehousing • T10ACHOCOL-29 Develop scraping model in local machine • T10ACHOCOL-30 Coding scraping realisation on AWS
Multiple format input	• T10ACHOCOL-25 Basic file upload function development - front-end • T10ACHOCOL-22 Basic login page development - front-end
Sufficient summary	• T10ACHOCOL-19 Experiment on RAGFlow first with small data

	● T10ACHOCOL-43 Integrate AI-generated case summaries
Improve searching efficiency and relevance	● T10ACHOCOL-11 Research existing R&D projects and Analyze pros and cons ● T10ACHOCOL-23 Basic prompt page development - front-end ● T10ACHOCOL-39 Setup RAG pipeline ● T10ACHOCOL-40 Implement query rewriting (expanding, refining) ● T10ACHOCOL-41 Implement hybrid search (keyword + semantic search) ● T10ACHOCOL-42 Implement multi-hop retrieval for complex legal queries ● T10ACHOCOL-16 Implement an Evaluation component for RAG result
Convenient in case comparison	● T10ACHOCOL-36 UI Design ● T10ACHOCOL-2 As a member of the General public, I want to search SAT decisions using natural language so that I can retrieve relevant cases and generate analysis reports. ● T10ACHOCOL-4 As a lawyer, I want to be able to build claims based on past decisions and relevant legal precedents efficiently, so I can save time for more client-focused work.
Legal reasoning	● T10ACHOCOL-20 Connect RAG component to read data from database in order to retrieve relevant cases ● T10ACHOCOL-21 Improve retrieval mechanisms, refine prompt engineering, and minimize hallucinations to ensure relevant responses. Improve generation capability with reasoning LLMs.

References

1. Alammar, J & Grootendorst, M 2024, *Hands-On large language models: Language Understanding and Generation,* O'reilly Media, Sebastopol.
2. Tunstall, L, Von Werra, L & Wolf, T 2022, *Natural Language Processing with Transformers: building language applications with hugging face,* O'reilly Media, Sebastopol.
3. Perez, M 2023, 'What is Web Scraping and What is it Used For?', *ParseHub Blog*, weblog, 6 August, accessed 10 March 2025, <https://www.parsehub.com/blog/what-is-web-scraping/>.
4. Radavicius, D 2023, 'Best Programming Languages for Effective Web Scraping', *Oxylabs*, weblog, 31 march, accessed 10 March 2025, <https://oxylabs.io/blog/best-web-scraping-language>.
5. ECourts Portal of Western Australia 2025, *Acknowledge Conditions Of Use - eCourts Portal*, Western Australia Government, accessed 10 March 2025, <https://ecourts.justice.wa.gov.au/eCourtsPortal/Decisions/Filter/SAT>.
6. Austlii 2025, *State Administrative Tribunal of Western Australia*, accessed 10 March 2025, <https://www.austlii.edu.au/cgi-bin/viewdb/au/cases/wa/WASAT/>.
7. ZenRows 2024, *Scrapy vs BeautifulSoup: Which Is Better For You?*, accessed 10 March 2025, <https://www.zenrows.com/blog/scrapy-vs-beautifulsoup#similarities>.
8. PostgreSQL 2019, *PostgreSQL: About*, The PostgreSQL Global Development Group, accessed 10 March 2025, <https://www.postgresql.org/about/>.
9. LangChain 2024, *Faiss*, LangChain Inc, accessed 10 March 2025, <https://python.langchain.com/docs/integrations/vectorstores/faiss/>.
10. Amazon Web Services n.d., *MongoDB vs PostgreSQL - Difference Between Databases*, Amazon Web Services Inc, accessed 10 March 2025, <https://aws.amazon.com/compare/the-difference-between-mongodb-and-postgresql/>.
11. Neon 2024, *PostgreSQL Full-text Search*, Neon Inc, accessed 10 March 2025, <https://neon.tech/postgresql/postgresql-indexes/postgresql-full-text-search>.
12. Myscale 2024, *Faiss vs. Pinecone: Ideal Vector Database Comparison*, MOQI Singapore Pte Ltd, accessed 10 March 2025, <https://myscale.com/blog/faiss-vs-pinecone-ideal-vector-database/>.
13. Pathak, TR 2024, '*Understanding Faiss and Pinecone Vector Databases: A Comprehensive Guide'*, Medium, weblog, 2 March, accessed 10 March 2025, <https://medium.com/@tejupathak/understanding-faiss-and-pinecone-vector-databases-a-comprehensive-guide-66502279077e>.
14. Thomson Reuters n.d., *Westlaw*, accessed 10 March 2025, <https://www.thomsonreuters.com.au/en-au/products/westlaw.html>.

15. Ragflow 2025, *Get started*, InfiniFlow, accessed 9 March 2025, Available at: <https://ragflow.io/docs/dev/>.
16. Lexisnexis 2025, *Lexis+ AI: Conversational Search Platform*, LexisNexis, accessed 9 March 2025, <https://www.lexisnexis.com/en-au/products/lexis-plus-ai>.
17. LLM Stats 2025, *DeepSeek-R1 vs Llama 3.1 405B Instruct vs Claude 3.7 Sonnet*, LLM Stats, accessed 9 March 2025, <https://llm-stats.com/models/compare/deepseek-r1-vs-llama-3.1-405b-instruct-vs-claude-3-7-sonnet-20250219>.
18. Huggingface 2025, *Qwen/QwQ-32B*, accessed 9 March 2025, <https://huggingface.co/Qwen/QwQ-32B>.
19. Liu, N, Zhang, T & Liang, P 2023, *Evaluating Verifiability in Generative Search Engines*, <doi:https://doi.org/10.18653/v1/2023.findings-emnlp.467>.

Appendix

Figure 1

Detailed ETL pipeline design diagram

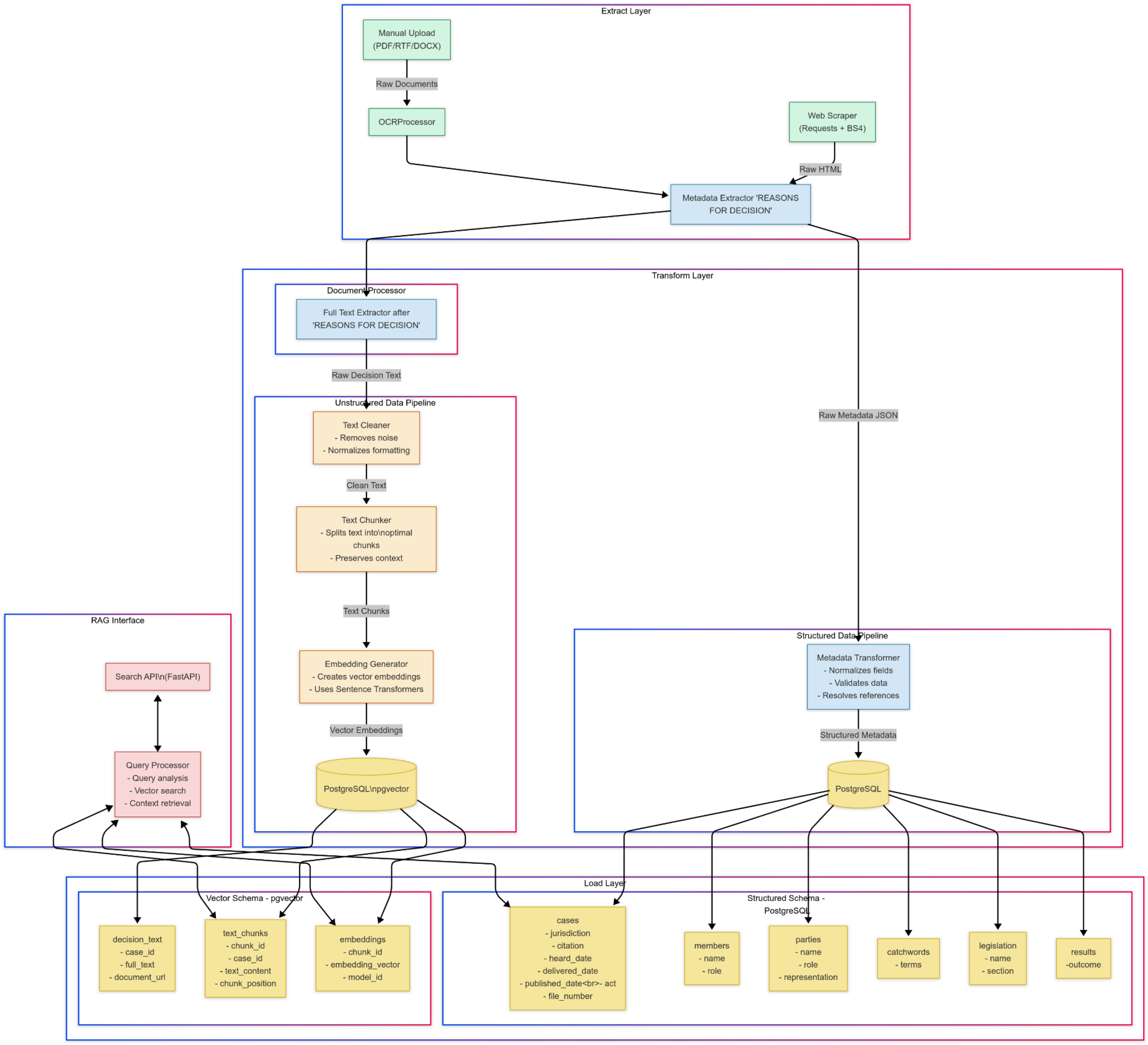
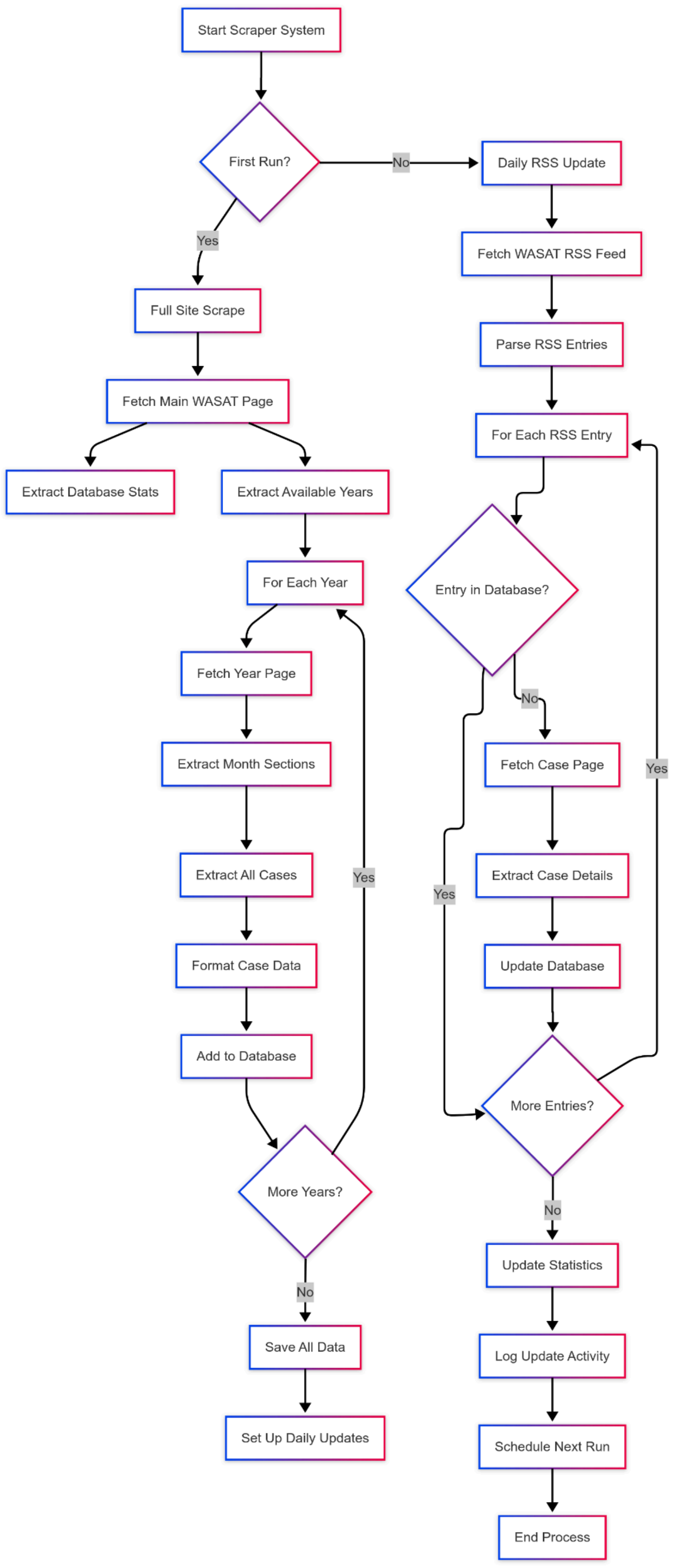

Extract Layer
Manual Upload (PDF/RTF/DOCX)
Raw Documents
OCRProcessor
Web Scraper (Requests + BS4)
Raw HTML
Metadata Extractor 'REASONS FOR DECISION'
Transform Layer
DocumentProcessor
Full Text Extractor after 'REASONS FOR DECISION'
Raw Decision Text
Unstructured Data Pipeline
Text Cleaner - Removes noise - Normalizes formatting
Clean Text
Text Chunker - Splits text into\noptimal chunks - Preserves context
Text Chunks
Embedding Generator - Creates vector embeddings - Uses Sentence Transformers
Vector Embeddings
PostgreSQL\npgvector
Raw Metadata JSON
Structured Data Pipeline
Metadata Transformer - Normalizes fields - Validates data - Resolves references
Structured Metadata
PostgreSQL
RAG Interface
Search API\n(FastAPI)
Query Processor - Query analysis - Vector search - Context retrieval
Load Layer
Vector Schema - pgvector
decision_text - case_id - full_text - document_url
text_chunks - chunk_id - case_id - text_content - chunk_position
embeddings - chunk_id - embedding_vector - model_id
Structured Schema - PostgreSQL
cases - jurisdiction - citation - heard_date - delivered_date - published_date
- act - file_number
members - name - role
parties - name - role - representation
catchwords - terms
legislation - name - section
results -outcome

Figure 2

Detailed WASAT Auto-Scraper System diagram


Start Scraper System
First Run?
No
Daily RSS Update
Yes
Fetch WASAT RSS Feed
Full Site Scrape
Parse RSS Entries
Fetch Main WASAT Page
For Each RSS Entry
Extract Database Stats
Extract Available Years
For Each Year
Entry in Database?
Fetch Year Page
No
Extract Month Sections
Fetch Case Page
Yes
Extract All Cases
Yes
Yes
Extract Case Details
Format Case Data
Update Database
Add to Database
More Entries?
More Years?
No
Update Statistics
No
Save All Data
Log Update Activity
Set Up Daily Updates
Schedule Next Run
End Process