

Contents

List of Figures	2
Introduction	5
Research Methodology.....	5
Literature Search	6
Selection Criteria.....	6
Background Research.....	6
History of NLP	7
Machine learning in NLP	8
The Role of NLP in Enhancing Access to Justice	10
Identifying and Mitigating Bias	11
Privacy and Security Concerns in Legal NLP Applications	11
Current Trends and Future Directions of NLP in law.....	12
Market Analysis	13
The Existing Market of household and Tenant law	13
Existing Solutions for Household and Tenant Law	14
Project Proposal.....	15
Project Details:	15
Project Specification:	15
Project User Stories	16
Traceability Matrix:.....	20
High Level System Design:.....	21
Requirement Analysis:	22
MoSCoW Analysis.....	22
Project System Requirement:	23
Project Solution	25
Assumptions, Constraints, Risks:.....	25
Solution Description:.....	27
Solution Delivery Approach:	27
Project Management Approach:.....	27
Design Phase:	31
Wireframes:	31

Database Design:	33
Dataset Workflow/Design:	35
LLD Design:	36
Implementation:	38
Database Implementation and Connection:	38
Front-end Implementation:	40
Backend-Server Side:	42
NLP Model Implementation:	50
Testing:	62
Unit Testing	62
Designing the Unit Tests Using Pytest:	62
Unit Tests Outcome:	67
Unit Tests Outcome Discussion:	69
Integration Testing Plan:	70
Integration Testing Outcomes:	71
Evaluation:	72
Future Work:	73
Conclusion:	74
References	75
Dataset Document Reference:	77

List of Figures

Figure 1 Showing the Hierarchy of NLP	7
Figure 2 Showing the Development of NLP overtime	8
Figure 3 Showing the Overlap of NLP with ML and DL	8
Figure 4 Showing the NLP Pipeline.....	9
Figure 5 Showing User Registration Use Case	16
Figure 6 Showing User Login Use Case	17
Figure 7 Showing User Legal Advice Use Case	17
Figure 8 Showing User Chat History Review Use Case.....	18
Figure 9 Showing User Providing Feedback Use Case	18
Figure 10 Showing User Logout Use Case	19
Figure 11 Showing General Overview of the Application	21

Figure 12 Showing HLD Design of Litigat8.....	21
Figure 13 Showing Different Layers of Litigat8.....	24
Figure 14 Showing Trello Board for litigat8- 2	29
Figure 15 Showing Project Timline Managment Using Gantt Chart for Litigat8 -1.....	30
Figure 16 Showing Trello Board for litigat8- 1	29
Figure 17 Showing Project Timeline Management Using Gantt Chart for Litigat8 -2.....	30
Figure 18 Showing Project Timeline Management Using Gantt Chart for Litigat8 - 3	30
Figure 19 Showing Registration Wireframe	32
Figure 20 Showing Login Wireframe	32
Figure 21 Showing Main Chat Interface Wireframe	33
Figure 22 Showing User Model	34
Figure 23 Showing ChatSession Model	34
Figure 24 Showing Conversation Model	34
Figure 25 Showing the ERD Diagram for Litigat8 Database	35
Figure 26 Showing the Framework for the Data Collection and Preparation for Litigat8 Training	36
Figure 27 Showing the Low-Level Design (LLD) for Litigat8	37
Figure 28 Showing the Setup of Database Config	38
Figure 29 Setting up the Database.....	38
Figure 30 Setting up the Models for Database.....	39
Figure 31 Showing the constructed Table in the Litigate Database	39
Figure 32 Showing the Front-end Implementation of Main-Page	40
Figure 33 Showing the Front-end Implementation of Login Page of Litigate	41
Figure 34 Showing the Front-end Implementation of Register Page of Litigate.....	41
Figure 35 Showing the Front-end Implementation Chat Interface of Litigate	42
Figure 36 Showing the implementation of Flask App with Configuration	43
Figure 37 Showing the Implementation of Auth Blueprint	43
Figure 38 Showing the Implementation of Chat Blueprint.....	43
Figure 39 Showing the setting up of Login Function Server Point	44
Figure 40 Showing the setting up of Register Function Server Point.....	44
Figure 41 Showing the setting up of Logout Function Server Point	44
Figure 42 Showing the implementation of start_chat_session Server Point.....	45
Figure 43 Showing the implementation of end_chat_session Server Point.....	45
Figure 44 Showing the implementation of server handling of /submit route	46
Figure 45 Showing the implementation of server handling of /save_interaction route.....	46
Figure 46 Showing the implementation of server handling of /get_conversation/session_id route	47
Figure 47 Showing the implementation of server handling of /get_chat_sessions route	Error!
Bookmark not defined.	
Figure 48 Showing the implementation of server handling of /fetch_chat_histories route.....	47
Figure 49 Showing the implementation of Fetch Requests of chat/fetch_chat_history	48

Figure 50 Showing the implementation of Fetch Requests of chat/start_chat_session	48
Figure 51 Showing the implementation of Fetch Requests of chat/submit	49
Figure 52 Showing the implementation of Fetch Requests of chat/save_interaction	49
Figure 53 Showing the implementation of Fetch Requests of chat/get_conversations/sessionID and Processing the of JSON load to front-end	50
Figure 54 Showing the implementation of Fetch Requests of chat/start_chat_session	50
Figure 55 Showing the text data structure before pre-processing steps	51
Figure 56 Showing the preprocessing steps i.e., Tokenization, Lowercasing, Removing Stop Words, Stemming	52
Figure 57 Data Structure after Data Pre-Processing	52
Figure 58 Chunking/Splitting of the Text Data	53
Figure 59 Data After Chunking/Splitting	53
Figure 60 Showing the implementation of OpenAI Embedding Model	54
Figure 61 Showing the implementation of Hugging Face Embedding Model all-Mini-LM-L6-v2	54
Figure 62 Showing the Implementation of Pinecone Vector Database	54
Figure 63 Showing the Setting Up of Pinecone Database on the Website	55
Figure 64 Showing the Depreciation Warning of Pinecone	55
Figure 65 Showing the creation of Vector Database	55
Figure 66 Showing the Implementation of VectorDb Retriever	56
Figure 67 Showing the implementation of Retrieval-based QA Chain Using OpenAI() LLM ..	56
Figure 68 Showing the Response Generation with Source Citation	57
Figure 69 Setting up the context for the LLM	57
Figure 70 Showing the LLM answer to a Question out of Context	58
Figure 71 Prompt Engineering Version 1	58
Figure 72 Prompt Engineering Version 2	58
Figure 73 showing the implementation of gpt-3.5-tubo , llama-2-7b Model , gpt-4.0	59
Figure 74 Setting Up the Semantic Similarity func and Comparison func	60
Figure 75 Setting up the Prompts and Correct Answers	60
Figure 76 Showing the Comparison Result	60
Figure 77 Showing the response generated LLMA-2	61
Figure 78 Showing the response generated LLMA-2	61
Figure 79 Showing the response Generated by gpt-4	61
Figure 80 showing the response generated by gpt-3.5	61
Figure 81 Showing the results of pytest for Unit Tests	69
Figure 82 Showing the Unit Test for Iteration with Litigat8 NLP Model	69
Figure 83 Showing Warnings Generated by the Tests	69
Figure 84 Showing the Fix of Login (Success) component warning	70

Introduction

From the beginning of time the way people get legal help has changed a lot. In the past there was a time when getting legal advice was expensive for most people. As the world changed so did the need for legal help which became more complicated. Technology has played a big role in changing how we get legal support today. Now, legal help is a key part of dealing with many problems in life.

Thanks to the recent development in technology getting legal advice is easier than before. Smartphones and apps have gone from just being ways to talk to each other to important tools for handling different parts of our lives including legal issues. This technology lets people find legal information easily it empowers them by giving them the power to handle their legal matters more actively. But there's a problem, many good legal apps cost money every month or people have to use many different apps to find what

they need. This can make the whole process frustrating.

My project Litigat8, is here to make things better. I've created Litigat8 to be a complete source of legal support especially for issues between landlords and tenants. Whether it's understanding your rights as a tenant or knowing what a landlord can or cannot do or getting advice on how to handle a dispute Litigat8 is designed to help. The application offers easy-to-understand advice and resources for tenant and landlord laws.

Litigat8 is more than just information. It's a tool that helps users get answers to the question they are looking thus fulfilling the legal needs easily. This made this app to be a helper and a guide in the complex world of tenant and landlord laws. By making legal support accessible. My goal was to make everyone feel confident about their rights. Litigat8 isn't just an app it's a step towards a world where legal help is a tool for justice for everyone

Research Methodology

This section displays the approach employed for conducting literature search selection and subsequent analysis focusing on the role of Natural Language Processing in improving access to justice.

Literature Search

The literature search was performed across various academic databases to ensure a broad collection of relevant studies. The databases queried include Google Scholar, IEEE Xplore, ScienceDirect, SpringerLink and the ACM Digital Library. The search strategy employed combinations and mutations of key terms such as "Natural Language Processing," "NLP in law," "access to justice," "legal informatics," "automated legal assistance," "bias in NLP applications," "privacy in legal NLP," "NLP in household and tenant law" and "future of NLP in law." Boolean operators (AND, OR) were employed to refine the searches ensuring a focused retrieval of literature.

Selection Criteria

Inclusion Criteria:

- Peer-reviewed articles and conference papers written in English.
- Studies that specifically explore the application of NLP technologies in the legal domain or for enhancing access to justice including in the household and tenant law domain.
- Publications that provide insights into the current trends, challenges, and future directions of NLP in legal applications.

Exclusion Criteria:

- Non-peer-reviewed articles grey literature and opinion pieces.
- Articles without empirical data, clear results on NLP's effectiveness or a detailed methodology.

Background Research

Natural Language Processing (NLP) serves as a bridge between human communication and computer understanding serving itself as a base in both Artificial Intelligence and linguistics. Designed to make the interactions between humans and computers easy and more accessible. NLP aims to enable computers with the ability to understand human language with ease thus overcoming the need for people to master complex computer languages. NLP divides into two main branches, Natural Language Understanding and Natural Language Generation, focusing on understanding and the generation of human like text respectively. Linguistics the scientific study of language is crucial to NLP and it includes everything from phonology to semantics.

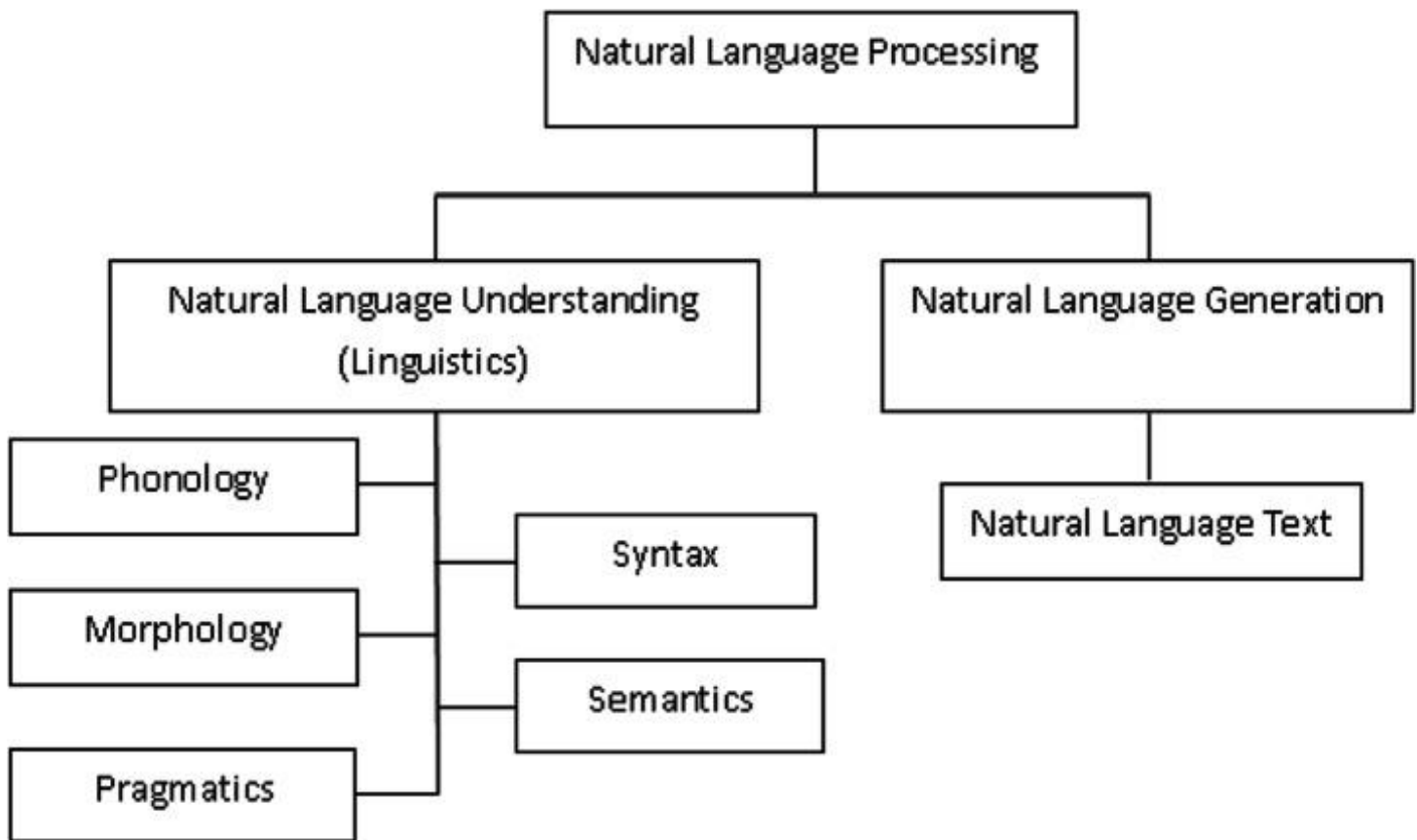


Figure 1 Showing the Hierarchy of NLP

History of NLP

The history of NLP can be traced back to the 20th century marked by technological advancements and contributions from different disciplines. This idea of NLP began in the late 1940s period when NLP as a term wasn't coined yet. The initial steps towards machine understanding of human language were being laid (Hutchins, W.J., 1986). The initial focus was on Machine Translation a task that planned to automatically translate text from one language to another in the beginning the two languages that were chosen Russian and English. This era was characterized by optimism. The ambitious projects aimed at breaking down language barriers using computers.

However this development faced a setback when the ALPAC report was released in 1966. Which stated that the future for MT was not so bright and therefore recommended a reduction in funding for such research (Hutchins, W.J., 1995). This report significantly hindered the progress of NLP research leading to a period of reanalyzing of goals within this field. Despite this roadblock, some MT projects continued to operate slowly refining their approaches and continued laying the groundwork for future successes.

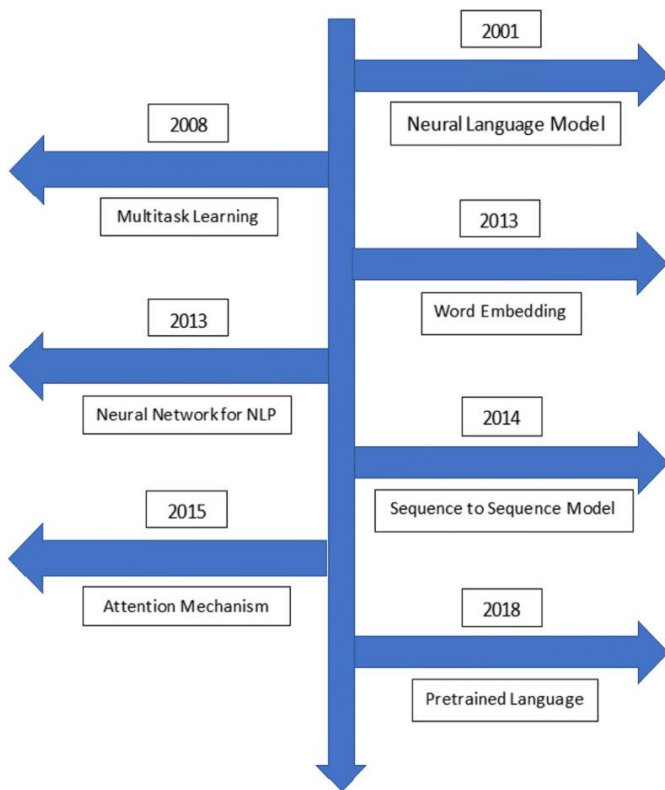


Figure 2 Showing the Development of NLP overtime

Today NLP stands as a dominant and very important field in computer science which stands at the intersection of AI, linguistics and computer science. This field of AI is continuously evolving with the introduction of new models and datasets. But the history showcases its journey of overcoming challenges, adaptation of new methodologies and expanding the boundaries of what machines can understand and use that understanding to generate meaningful and informative information. The future of NLP promises more applications of it in other applications such as law, customer service etc. It also promotes deeper understanding of linguistics and development of applications that extend beyond current capabilities of NLP.

Machine learning in NLP

Machine Learning plays a very important role in NLP which have facilitated the advancement in the field which have

revolutionized the way machine understand the give text data and generate Reponses based on that. The way it has been possible is by using patterns in data machine learning algorithms enable computers to perform complex NLP tasks without specific programming for each specific task beforehand. This use of ML in NLP has led to the development of applications such as speech recognition software's, sentiment analysis models, Text Translation models and chatbots which are just some of its applications and more of its applications are still being worked on.

Machine learning models like Naive Bayes, decision trees and support vector machines to

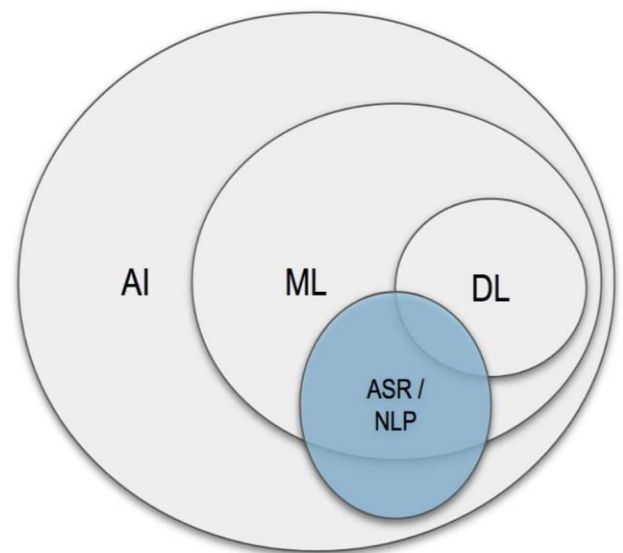


Figure 3 Showing the Overlap of NLP with ML and DL

advanced neural networks have provided NLP with the stepping stones to move forward resulting in wide variety of applications. These models are trained on large datasets learning to predict or classify text data.

With the development of transformer models i.e., BERT and GPT. The development of these models represents a leap forward in the development of NLP. These models have set

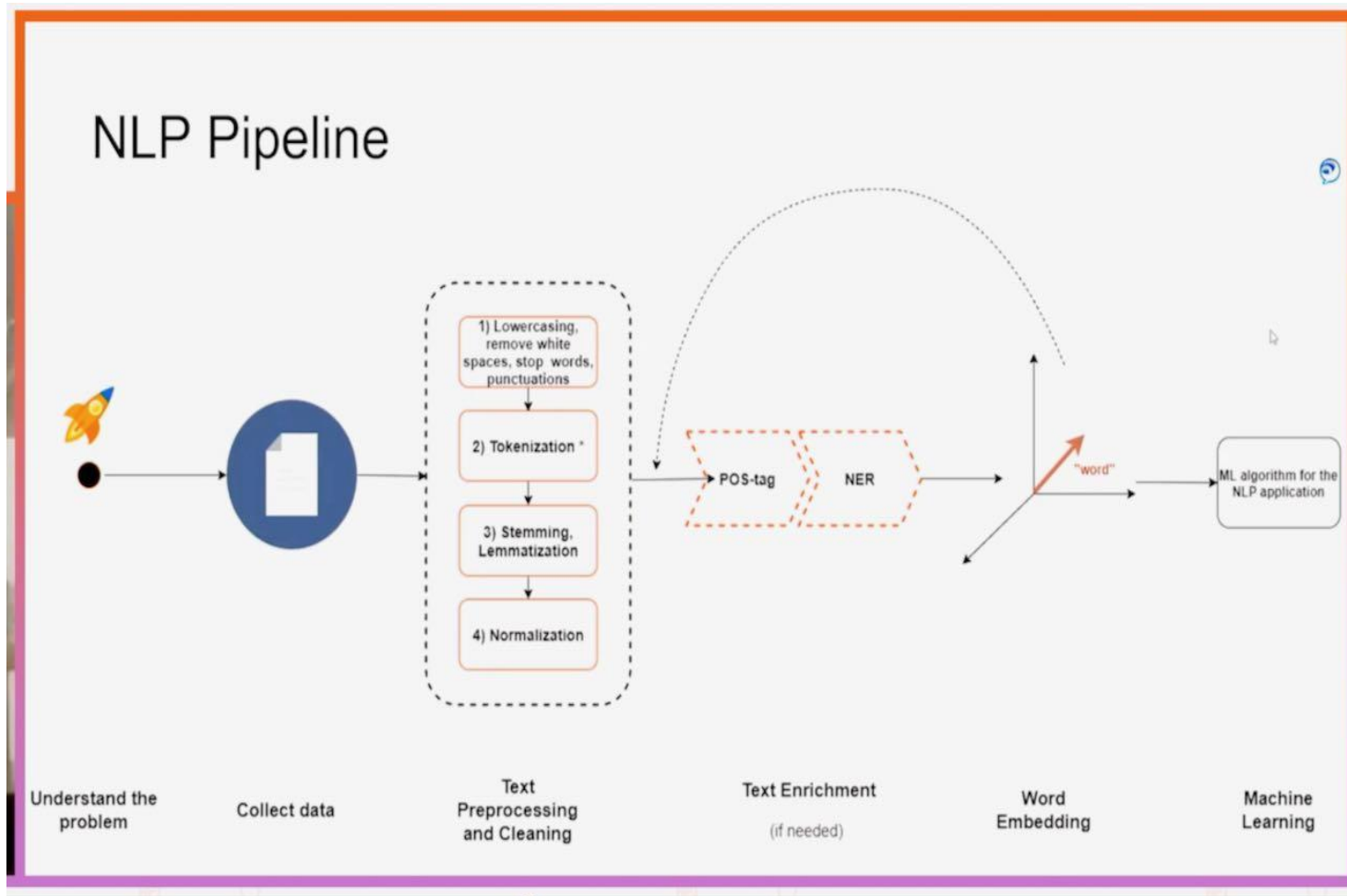


Figure 4 Showing the NLP Pipeline

new standards for a lot of wide range of NLP tasks by capturing deep contextual relationships within text (Vaswani, A. et al., 2017; Devlin, J. et al., 2019). Their ability to pre-train on vast amounts of text data and fine-tune for specific NLP tasks has made it more accessible and relatively easy to get unmatched accuracy in language understanding and generation of information based on specific NLP tasks.

Despite significant progress being made in the field integrating machine learning into Natural Language Processing for further development of NLP presents a lot of challenges. Some of the challenges include the need for large datasets with detailed and correct annotations for training which can be resource intensive. Additionally, machine

learning models struggle to understand the complexities of human language as they have lot ambiguous references in the form of sarcasm or something that is said out of the context of the topic. These models also require a lot computational power to train making the development of the models expensive and relatively difficult task to achieve. Furthermore, there is an ongoing effort to make these models more transparent in regards on how they make decisions to reduce biases that may be present in the training data used for training the model. Addressing these issues is very important for developing the NLP technologies further and to make sure the development is fair and understandable.

The Role of NLP in Enhancing Access to Justice

The role of Natural Language Processing (NLP) in enhancing access to justice is significant, by using the power of AI in law we have been able to bridge the gap between the existing legal services and the people who are in need of these services. This is achieved as a result of automating and streamlining the legal process and procedures that exists. With the help of NLP, we are able to make legal information more accessible, understandable and actionable either by individuals or communities who back in the day didn't have resources for legal representation

Make the Legal Language simple

One of the primary problems in accessing justice is the complexity of legal language. NLP tools have enabled us to translate legal jargon into plain language which in return makes legal documents, regulations and procedures more understandable to a common person who doesn't have any knowledge about the legal terms. The NLP can also make law more accessible by providing summaries of legal texts and also by explaining legal concepts in simple terms. Thus, NLP enables individuals in navigating the legal system more confidently which results in them making more informed decisions about their legal dilemmas (Katz, D.M., Bommarito II, M.J., and Blackman, J., 2017).

Automated Legal Assistance

Legal chatbots and virtual assistants which are powered by NLP, offer initial legal

advice and assist in document preparation which range from simple contracts to more complex legal filings Thus making the whole process a lot more convenient and easy for individuals. This application would also be explored in the case Litigat8 as well. These types of applications can guide users through legal processes, such as asking the relevant questions according to the problem the user is in and generate documents such as contracts, tenancy agreements based on the user's responses which in fact significantly lowers the barrier to start legal actions or responding to legal issues (Sourdin, T., 2018).

Enhancing Legal Research

NLP helps lawyers and legal researchers quickly search through huge amounts of legal documents like court cases and laws. By employing NLP tools, lawyers and researchers can easily find important past cases, also detect trends in legal decisions and collect evidence for their current cases if they are working on any. This makes their work more efficient which also helps to reduce costs. As a result, people who might not have a lot of money benefit because they can get better legal help that they can afford. (Alarie, B., Niblett, A., and Yoon, A.H., 2018).

Accessible Dispute Resolution

There are Online Dispute Resolution (ODR) platforms which utilize NLP to offer accessible means for resolving disputes outside of traditional court settings. These platforms automate parts of the mediation or

the arbitration process. The ODR platforms can resolve conflicts more quickly and with less financial strain on the parties involved which in results make justice accessible for all even if they are less fortunate(Rule, C., 2017).

Identifying and Mitigating Bias

Privacy and Security Concerns in Legal NLP Applications

When it comes to NLP, Privacy and security concerns are one of the biggest concerns in the deployment of Natural Language Processing applications within the legal domain. The Legal NLP applications, which handle sensitive and potentially confidential information makes it important to put in place some strict protocols and security measure to protect the user data as it can be confidential and of sensitive nature. The processing of legal documents, client communications between the NLP model and User and other sensitive information which can be possessed by NLP systems raises substantial concerns about data protection, unauthorized access and the potential misuse of information.

One of the primary challenges faced by legal NLP applications is ensuring the confidentiality and integrity of the data processed by it. The used legal documents often contain confidential information, trade secrets and personal data subject to various privacy laws such as the General Data Protection Regulation (GDPR) in Europe (Voigt, P., and Von dem Bussche, A., 2017). Because of these reasons, the NLP systems

used in the legal field must incorporate robust encryption methods for data storage and transmission, coupling it with secure access controls to prevent unauthorized access to sensitive information.

Furthermore, the use of NLP in legal applications involves ethical considerations such as the fairness and transparency of automated systems. The Bias in legal NLP models can lead to unfair outcomes or discrimination in the response generated or the legal advice which causes us to undermine the trust in automated legal analysis (Barocas, S., Hardt, M., and Narayanan, A., 2019). Therefore, NLP models generated for legal applications must be designed with fairness in mind and it should me made sure to incorporate methods to detect and mitigate bias in training data and model predictions.

Moving on ,another significant concern of NLP in legal applications is the potential for data breaches and the unauthorized leakage of sensitive information associated either with users or the law firm for instance using that application. It should be made sure that the Legal NLP applications must comply with legal standards for data protection, and they must implement stringent/strict security protocols and they development body should regularly audit systems for vulnerabilities (Romanosky, S., 2016). Moreover, there is a need that we put in place clear guidelines and regulations which should govern the use of AI and NLP in legal contexts ensuring that these technologies are used responsibly and ethically.

In summary, privacy and security concerns in legal NLP applications are critical issues that require careful consideration and proactive measures to prevent them from happening. Protecting sensitive legal information while ensuring the fairness and transparency of NLP systems is essential for maintaining client trust and compliance with legal and ethical standards and thus important for NLP to flourish in this field.

Current Trends and Future Directions of NLP in law

The integration of Natural Language Processing (NLP) within the legal domain has been very transformative in the recent years, reshaping how legal professionals in practice interact with vast amounts of textual data and how they use it to streamline various aspects of legal research, document analysis and client services. As NLP technologies continue to develop several current trends and future directions of NLP in legal settings are emerging in the field of law , which promise to further revolutionize they field of law as we know it right now.

Current Trends

1. Automated Legal Document Analysis:

One of the applications of legal NLP is to automate the analysis of documents like contracts. The way NLP model helps is by quickly finding and pulling out relevant and important information from these documents. Which can then be analyzed for whatever

purpose the legal professionals want to use it for. For example, a legal NLP model can identify key clauses in contracts and can summarize long legal opinions which results in saving time and thus making legal work more efficient. This application of legal NLP not only speeds up the process of reviewing legal documents but also reduces the chance of making mistakes thus resulting in improved accuracy and reliability of legal research or due diligence. This application of NLP has streamlined many routine tasks in legal practices. (Zhong, H., Guo, Z., Tu, C., Xiao, C., Liu, Z., and Sun, M., 2020).

2. Legal Chatbots and Virtual Assistants:

One of the other applications of NLP is Legal chatbots and virtual assistants which have NLP capabilities and they are becoming more prevalent and more used in day to day legal needs. These NLP technologies offer basic legal advice to the public helping them navigate through the complexities of the legal world and they also help in drafting simple legal documents and provide support for customer service operations in law firms making legal services more accessible (Kreutzer, R.T., and Sirrenberg, M., 2019).

Future Directions

1. Enhanced Legal Predictive Analytics:

The future of NLP in law includes the development of more advanced prediction and analytics tools. The way this system would work is by analyzing historical legal data and then the NLP models would be able to predict the outcomes of cases like what would be the decision based on the precedent

set by the cases related to the case that is being analyzed this would help lawyers make better informed decisions about case strategies and thus would increase the chances of success (Ashley, K.D., 2017).

2. Ethical Use of NLP in Legal Applications:

As NLP technologies become more common in legal practise, there will be an increased focus on ensuring the ethical and safe use of these tools. This will include addressing concerns related to unbalanced outcomes, transparency and the explainability of NLP models to ensure fair and impartial legal outcomes (Branting, L.K., 2021). To solve these problems work is being done to develop methods to test and correct biases in algorithms along with designing systems that can explain their own decisions and can follow strict guidelines to uphold ethical standards in their applications.

3. Cross-lingual and Multijurisdictional Legal NLP Applications:

Another possible application of Future NLP systems will likely become more adept at handling multiple languages and legal jurisdictions which would facilitate in cross-border legal research and global compliance tasks in different countries instead it being locked up by the knowledge of just single country. This advancement could come in really handy for international law firms and organizations dealing with cases which fall in under lots of jurisdictions legal issues (Tsarapatsanis, D., and Aletras, N., 2021).

Market Analysis

The Existing Market of household and Tenant law

When we consider the practical applications of NLP in legal systems, certain domains can benefit significantly, such as Property Law or Household and Tenant Law. These areas are particularly relevant given the market size and the number of active individuals who, at some point in their lives, must deal with the legal jargon of property law. According to Nimblefins, there are almost 8.5 million households currently renting in the UK property market. To put this into perspective, that number represents households, not individuals; thus, the actual count of people involved in this market is far greater. These figures include just tenants; there are also millions of landlords who must address the needs and problems of every tenant. Due to a lack of legal knowledge about this particular domain of law, many tenants face significant trouble dealing with courts and lawyers, which can be economically intensive for both parties simply because they lack basic guidance on how to navigate these issues, such as whether they even need a lawyer. According to ITV, there were almost 30,230 no-fault evictions in the UK as of 2023, which could have been easily avoided if the tenants and households had sought guidance earlier rather than getting into legal trouble by turning to lawyers when it was too late. Likewise, for landlords, just in the city of Sheffield, the disrepair cases rose from 117 in April 2018 to over 1,750 cases, where the landlords had to pay the tenants, compensation totaling around 3 million pounds. Hence, there exists a gap that needs

to be filled to make people more accessible to the knowledge of household and tenant law.

Existing Solutions for Household and Tenant Law

When it comes to existing solutions for household and tenant law in the NLP domain, search queries like "household and tenant law chat app," "NLP app in household and tenant law," and vice versa were used, but there wasn't a solution that existed in this domain. However, I found many use cases and examples of NLP in law. For instance, Nessa, an LLM solution developed by OLS Solicitors, was recognized as it did the job of providing basic advice regarding matters of family law. The solution that was developed was based on the GPT-4 model. Even though it provided basic advice, it wasn't able to generate case law references for the advice given, even when specifically asked for it, and this was a gap I feel could be filled with the help of Litigate, making sure it provides case law references to the user if prompted too.

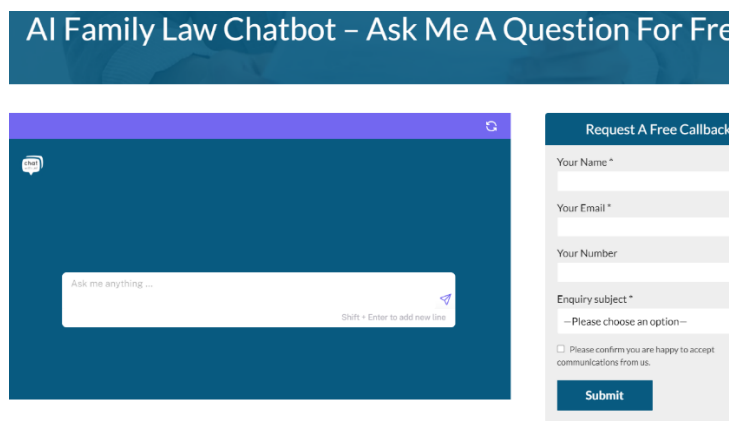


Figure 5 Showing the Nessa App Interface

The Nessa app didn't have the ability to save or store your chat session, which I feel makes it look a little less personalized and maybe even not that attractive to the user.

Following the analysis of Nessa another innovative solution was analyzed as well and which as AI Lawyer, which was more focused on the domain of law research and had tones of innovative features like ability to draft documents and upload document to question the NLP model for. The UI was very innovative and can serve as an inspiration for the development of Litigate but because of it being a paid service a proper analysis of the platform couldn't be carried out.

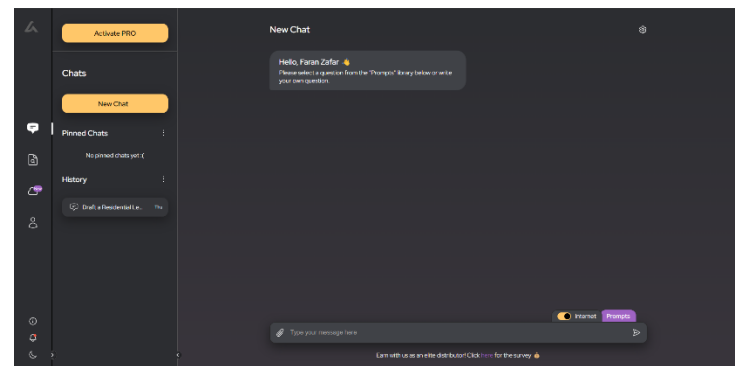


Figure 6 Showing AI Lawyer interface

A feature that it was lacking was legal advice generation and case law reference based on the specific advice. This again leaves a gap for this proposed solution to solve, which would be discussed in the next chapter of this report.

Project Proposal

Project Details:

Litigate would be an online, hosted chat application accessible as a web application. Users can find legal advice related to household and tenant law on the go. There will be a minimum age limit for accessing the platform, but people from all backgrounds will be able to use the application. An NLP model working in the backend, in relation to a database, will process user queries in real-time and produce appropriate responses using NLP techniques. The responses will include basic advice about the matter and, if available, a case law relevant to the issue from the database

Project Specification:

Project Scope and Objectives:

The scope of the project has been defined by the identified said objectives that need to be accomplished throughout the development of the project. The primary objectives carry more weight as they would be allocated the most resources and time before the presentation of the project in April,2024. If the primary objectives are achieved only then the secondary objectives would be considered implementing into the overall workflow of the application.

Primary Objectives:

1. ***User Interface Development:*** Design a user-friendly interface desktop platforms that allows users to easily interact with the chat application.
2. ***Natural Language Processing Integration:*** Implement a robust NLP model to understand and process user queries accurately in real-time.
3. ***Legal Database Creation:*** Compile a comprehensive database that includes relevant case laws, statutes, and legal precedents pertaining to household and tenant law.

4. ***Real-time Response Generation:*** Develop a system capable of generating accurate legal advice and relevant case law references in response to user queries.

Secondary Objectives:

1. ***Accessibility and Inclusivity:*** Ensure the app is accessible to users from all backgrounds, with considerations for those with disabilities
2. ***Security and Privacy:*** Implement robust security measures to protect user data and ensure privacy, especially when handling sensitive legal queries.
3. ***User Authentication:*** Create a secure user authentication system requirement for accessing the platform.
4. ***Feedback Mechanism:*** Incorporate a feedback mechanism to collect user responses on the accuracy and helpfulness of the legal advice provided, facilitating continuous improvement.

Project User Stories

Keeping the Objectives and the idea of the project in mind. I have brainstormed the potential use cases a user can have with my application. Which would further be broken down using other analysis techniques such as MoSCoW and depending on the resources and time available.

Use Case 1: User Registration

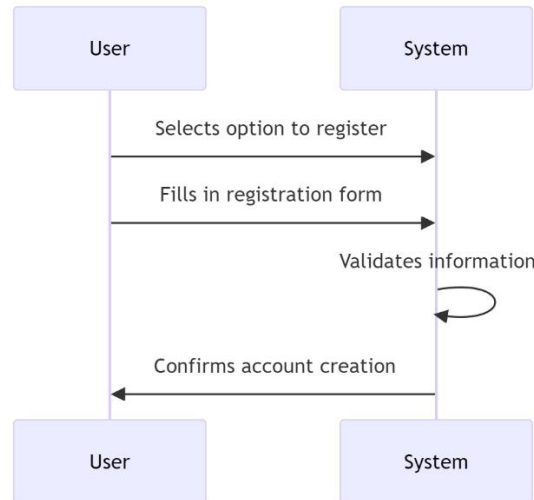


Figure 7 Showing User Registration Use Case

Use Case 2: User Login

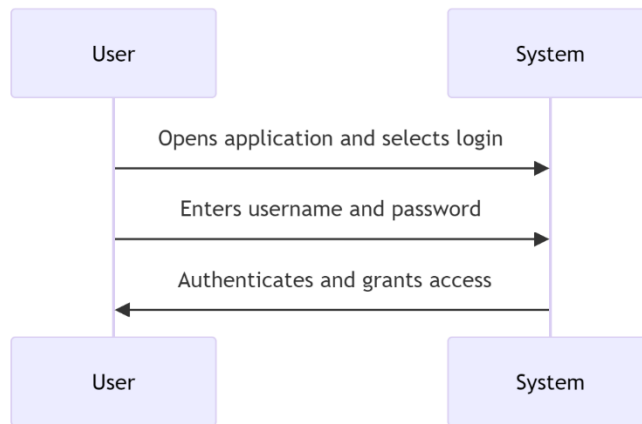


Figure 8 Showing User Login Use Case

Use Case 3: User Receives Legal Advice

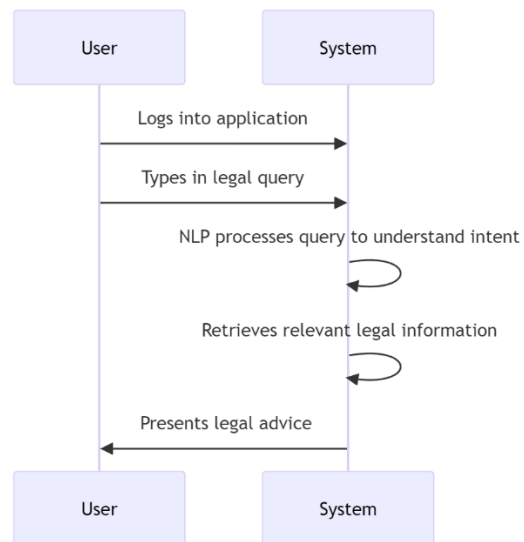


Figure 9 Showing User Legal Advice Use Case

Use Case 4: User Reviews Chat History

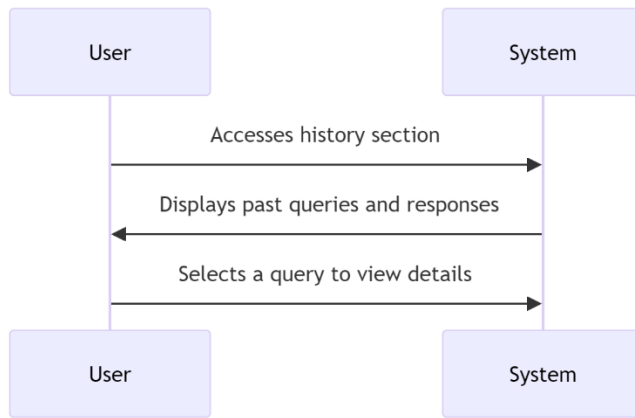


Figure 10 Showing User Chat History Review Use Case

Use Case 5: User Provides Feedback

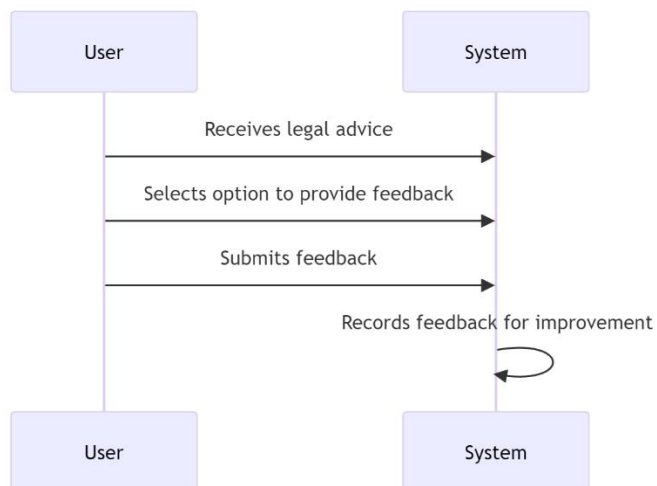


Figure 11 Showing User Providing Feedback Use Case

Use Case 6: User Logs Out

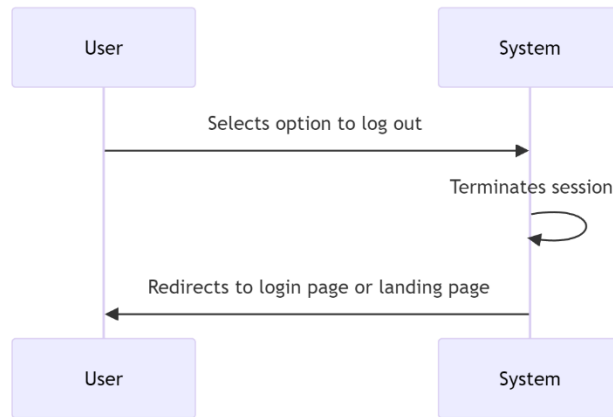


Figure 12 Showing User Logout Use Case

Traceability Matrix:

After analyzing the use cases and objectives of the project, a traceability matrix was designed to

Objective ID	Objective Description	Requirement IDs	Use Case IDs
P1	User Interface Development	M1, S1	UC1, UC2, UC5
P2	Natural Language Processing Integration	M5, M6	UC3
P3	Legal Database Creation	M7, M8	UC3
P4	Real-time Response Generation	M6, M8	UC3
S1	Accessibility and Inclusivity	S1	UC1, UC2, UC3, UC4, UC5, UC6
S2	Security and Privacy	M2, M4, M9	UC1, UC2, UC6
S3	User Authentication and Age Verification	M2	UC1, UC2
S4	Feedback Mechanism	S2	UC5

ensure that the objectives put in place are fully achieved by meeting the defined use cases for Litigat8.

High Level System Design:

In line with the user stories I've decided to focus on, sketching out a high-level design (HLD) helps us map out the system's architecture in a nutshell. This overview helps pinpoint the necessary hardware and software interactions, data exchanges, and communication flows across

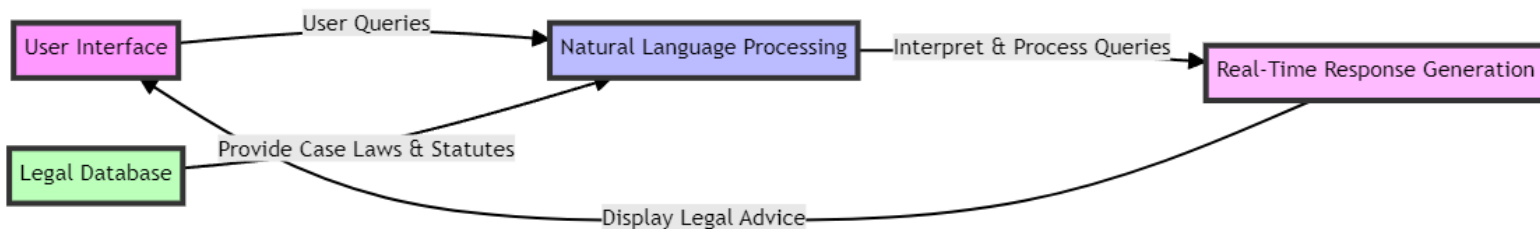


Figure 13 Showing General Overview of the Application

the system. At the heart of the architecture is a user interface designed to retrieve and update information from database tables. One of the key advantages of our approach is its scalability. As new data needs arise in the future, our system can easily expand to accommodate these additional data sources.

This diagram shows a high-level flow of data between different software and hardware components.

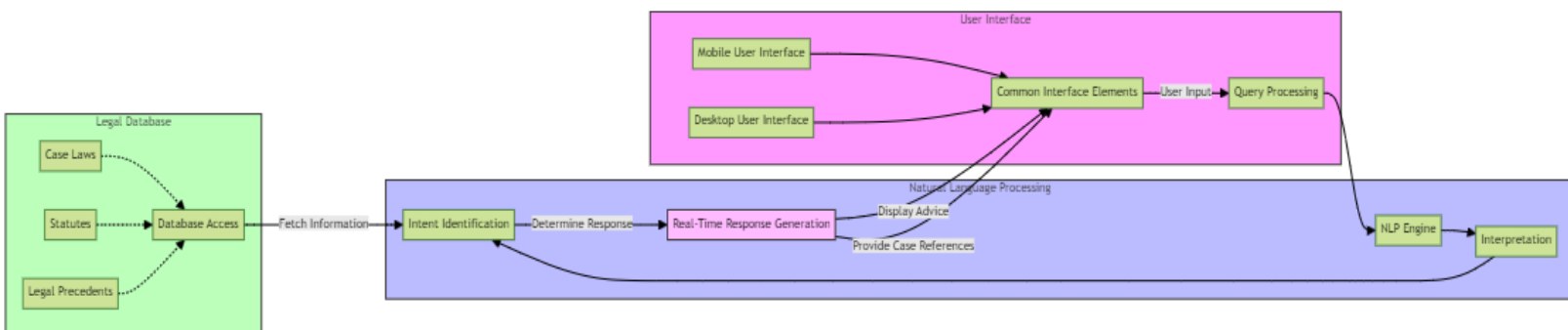


Figure 14 Showing HLD Design of Litigat8

One of the disadvantages of using the architectural design and solution, is the current technological skillset that is needed to achieve this implementation of the project. The Mitigation of the potential risks will be taken into account by discussing the it with the supervisor every week.

The diagram presented offers a detailed view of our proposed high-level design, showcasing the different elements, connections, and data movements.

Requirement Analysis:

The requirements would be defined using the MoSCoW Analysis and according to the objectives defined before.

MoSCoW Analysis

Based on the objectives mentioned in the Project Scope and objective I was able to shortlist the requirements for the project using the Moscow Method. Where the Must have requirements take the highest priority followed by should have and could have requirements respectively. And I thought of the requirements that could be implemented as a part of future development of the project

Must Have

ID	System Component	Requirement Description
M1	User Account Management	User registration with comprehensive data validation.
M2	User Account Management	Secure user authentication system implementation.
M3	User Account Management	Effective user session management and secure logout.
M4	User Account Management	End-to-end encryption of user data for security.
M5	Core System Functionality	Integration of NLP engine for natural language processing.
M6	Core System Functionality	Real-time legal advice generation based on NLP analysis.
M7	Core System Functionality	Legal database connectivity for dynamic access.
M8	Core System Functionality	Automated retrieval and presentation of case laws and statutes.

Should Have:

ID	System Component	Requirement Description
S1	User Interface	An intuitive design for using the web application
S2	User Experience	Feature for saving and displaying user query history.

Could Have:

ID	System Component	Requirement Description
C1	Extended Functionality	Multilingual support to cater to a diverse user base.
C2	Extended Functionality	Notification system to alert users about updates or responses.

Won't Have:

ID	System Component	Requirement Description
W1	Future Considerations	Hosting on a real domain for processing real-time queries.
W2	Future Considerations	Advanced document generation capabilities.

Project System Requirement:

Based on the requirement analysis done for the project. I have decided on the software and hardware requirements of the project. As the requirements of the project are modest and it would only be hosted on the localhost for the MVP. I have kept the requirements at minimum for the functioning of the project.

Development Languages and Tools:

1. **Backend Programming Language:** Python, chosen for its extensive library support, particularly for natural language processing (NLP) tasks. I decided to choose this language for the access of the existing information for python in compared to other languages such as C++. And another factor that contributed to its selection was my own proficiency in the language given the duration of the project. It would have been very highly unlikely to complete the project if I had decided to proceed with the language, I didn't have command on.
2. **Frontend Web Technologies:** HTML, CSS, and JavaScript, for creating an interactive and user-friendly web interface. The User interface is decided to kept simple as most of the resources and time would be allocated to developing the NLP model. Hence, no frontend framework is being used at the moment but if time is left. I'm planning to implement React as the frontend framework for the application.

3. **NLP Library:** NLTK, utilized for implementing the chat application's NLP capabilities to process and understand user queries. From the NLTK libraries, I'm going to be using PortStemmer , stopwords, word_tokenize to implement functions such as tokenization, stemming and bag of words.. For developing and making the vector database for the project I would be using libraries such as Pincone and Chroma. While for the rest of the functionality of the NLP model I would be using LangChain which in itself has many libraries such as OpenAI, Ctransformers which would be used and detailed in the implementation section
4. **Web Framework:** Flask, selected for its simplicity and efficiency in setting up lightweight web applications, suitable for a project focused on functionality demonstration. Option of Django was considered as well for the development of the project but given the simple nature of the MVP I decided to settle down for flask. And I had some previous experience using this framework so it was a natural choice for me.
5. **Database System:** MySQL or Firebase, due to its ease of integration with Python applications and sufficiency for the storage needs of a development-stage project.
6. **Integrated Development Environment(IDE):** VsCode, recommended for its support for Python and web technologies, alongside conda would be use to set up a virtual environment. Github, would also be implemented for version control of the project.

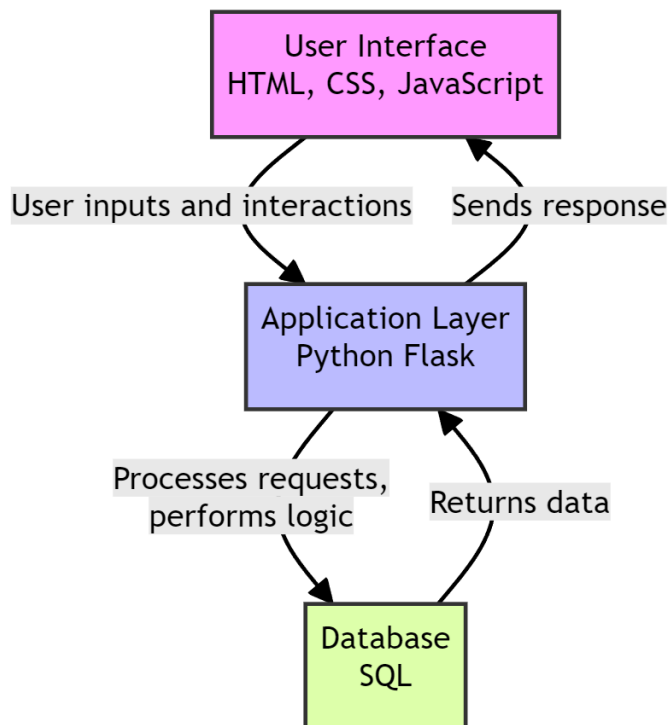


Figure 15 Showing Different Layers of Litigat8

dependencies.

Hardware Requirements:

Given the localized hosting and demonstration focus of the project, the hardware requirements are kept simple:

1. **Processor:** A minimum of a Dual-core processor is required to ensure smooth running of the development server and NLP processes.
2. **Memory:** At least 4 GB RAM to support the simultaneous execution of the development environment, web server, and any ancillary tools.
3. **Storage:** A minimum of 5 GB of available disk space to accommodate the application codebase, SQLite database, and dependencies.

Project Solution

Assumptions, Constraints, Risks:

Assumptions:

Technical Capabilities:

It's assumed that the chosen technologies and frameworks for NLP(NLTK, LangChain, OpenAI, HuggingFace), database generation (Chroma), and other functionalities will effectively support the development of Litigat8. As this is a realm still to be explored and considering the limited time and resources available full-scale deployment of the project can be somewhat troublesome. But the given requirements for a Minimum Viable Product (MVP). It is assumed that all the technical requirements are satisfactory.

Data Availability:

Adequate and accurate tenant-landlord law documents, statutes, and regulations are available for populating the database. A I'm going to be preparing the dataset myself. It is assumed that the dataset would have adequate information for the model to produce appropriate responses for the user. And all the information would contain in the dataset is correct. Even though there can be discrepancies in the dataset

User Feedback:

Users will provide feedback for system enhancement and that the feedback will be constructive. It is assumed that the feedback provided is constructive and explainable but still there can be instances where the feedback isn't constructive or comprehensible. And User Feedback is one of the secondary objectives hence it would take less resources during the production of software.

Compliance and Legal Clearance:

All necessary legal clearances and compliance requirements related to providing legal advice and handling user data will be obtained. It is assumed that the model has been approved of giving legal advices in the matters of household and tenant law. As law and AI is a very tricky domain where you have to be careful about the data that is produced and dispersed to the user. As this is a demonstration of its capabilities and the project is being made for **ONLY** this purpose. Hence, it is assumed it has all the legal clearance.

Budget and Time:

The project will be completed within the allocated budget and time frame. According to the defined user stories and analysis of requirements using methods like MoSCoW. It is assumed that all the set requirements would be completed within Budget and Time. But as this can vary depending on various factors. But for the sake of convivence an assumption is put into place.

Constraints:

Single Stakeholder:

As the sole stakeholder, your availability and decision-making will be critical for the project's progress and direction. As I'm going to be the only one that would be directing the progress of the project along with the supervisor which can put a bit of constraint on the project in a sense that I would have to make all the critical decision instead of having a shared responsibility of the decision-making progress.

Budget Limitations:

Budget constraints may limit the scope of development, particularly concerning the scalability and future plans for web and mobile interface accessibility. The options of cloud hosting and own professional domain will be looked in the future if there are enough resources allocated to the project if there are investors involved in the project but for this development cycle it is constraint to be hosted locally without a domain.

Technological Limitations:

The performance of Litigat8 may be constrained by the capabilities of the chosen technologies and frameworks. Some of the technologies chosen for the project for instance python in itself may not be the best choice when it comes to making heavy load models which are capable of processing natural language. But for the sake of demonstration and to have an MVP to show for by the end of April. Some of the technologies were given more weight.

Geographical and Jurisdictional Limitations:

Legal advice and statutes provided will be limited to specific geographical regions or jurisdictions. As this model is being developed just for the jurisdiction of UK law hence it would only be able to provide support and advice for Tenant and Household law in UK.

Risks:

Some of the foreseeable risks have been identified that could potentially happen in the duration of the project. The mitigation techniques for these risks would be discussed with the supervisor in the weekly meetings. And sufficient alternative pathways would be setup for the project if I encounter any unforeseen circumstances which could lead to failure of any component of the project.

Technology Failure:

Risks of technology failure impacting the accuracy and reliability of legal advice provided. As there are going to be lots of functioning elements in the project from the interface to the database lots of different factors would be involved. Hence, there is a risk of technological failure if one of the components crashes or fails due to unforeseen circumstances. But sufficient steps would be taken in order to prevent that from happening. Ensuring good coding practices and proper documentation of the project to ensure any unforeseen circumstances should be dealt with using proper protocols put in place.

Dependency on External Systems:

Future dependency on external cloud hosting platforms and cloud-based database like firebase. I have yet to decided if I want to go for a inbuild database like SQLite or Firebase. If I decided on settling for firebase then there is a potential risk that the application might not work because of being depended on an external firebase database. But steps would be taken and would be discussed with the supervisor to mitigate the risk as much as possible.

Solution Description:

Based on the extensive information that was collected during the research and the requirement analysis of the project, sufficient evidence is provided for the case that a web application would be suffice to resolve the said user stories and objectives that have been identified for the project. A web application would enable to design a proper interface for the user for ease of access along with a functioning backend layer with an NLP model which in itself is very intensive and alongside that would the database for the whole system which would allow to create, update and delete user data along with the chat data. However, due to number of limitations, it is limited in its ability to create complete system. Therefore, the proposed system would deliver a proof of concept, which would only show the potential of the system in case if it's implemented as a complete system.

The poof-of-concept application (litigat8) will showcase the process to input the query as a string and then the system would be able to process the information to come with appropriate responses along with case laws and statues if they exist for that specific case

The application would showcase how the user would be able to go back to his/her queries if he or she wants to look over his/her queries again.

Solution Delivery Approach:

Project Management Approach:

Agile Methodology

In line with modern software development practices, the project adopted the Agile methodology starting from October 15, with a projected end date of April 15. This approach, characterized by its flexibility and iterative nature, emphasizes adaptability and responsiveness to evolving requirements. Agile methodology, as defined by Beck et al. (2001), offers a dynamic alternative to the rigid, linear progression of the Waterfall model. It is conducive to environments like the one that is needed for litigat8 where user needs and system functionalities may not be fully understood from the start of the project.

Project Management Using Scrum Framework

Scrum a subset of Agile is chosen as the operational framework for its fit with the project's objectives. It provided a structured yet adaptable environment to create a user-centered platform. The Scrum methodology facilitated constant supervisor engagement enabling me to incorporate feedback iteratively and ensure that the platform remains aligned with the project objectives.

Weekly sprints were scheduled, wherein I focused on delivering specific, prioritized features from the product backlog. Regular sprint reviews and retrospectives ensured that the project adapted to feedback and improved upon the processes continuously.

Scrum Sprints

After analyzing the total time left for the project to be completed and the list of features and tasks that need to be carried out during the project development, five sprints in total were deemed sufficient for the development of Litigate. The first two sprints were a week long, and the last three were two weeks long.

At the beginning of each sprint, a sprint planning session will be held, ranging from one to two hours. This session will identify the problems that are of the highest priority, and it will be ensured that a good amount of time is provided to them. It is of utmost importance that the issues that originate are kept on track. For this purpose, sticky notes will be used on windows. The features will be split into three sections: Done, Doing, and Will Do.

Sticky notes were preferred for Scrum because they are easy to use and accessible.

SPRINT 3 DONE Complete the frontend Login/Register page with form validation. Develop the Dashboard page's frontend. Implement Query Input Form and Advice Display Section. DOING Develop the NLP processing module for query processing. WILL DO Start implementing API endpoints for processing queries and handling chat
SPRINT 1 DONE <ul style="list-style-type: none">• Initialize the project repository and define the directory structure.• Set up version control and branching strategies. DOING <ul style="list-style-type: none">• Create a basic Flask application skeleton.• Start work on the authentication system WILL DO <ul style="list-style-type: none">• Implement user registration and login functionalities.• Set up a local development database with MySQL

Figure 16 Sample Showing Sprints

Regular meetings will be held with the supervisor to ensure that the development is going according to plan and is not deviating from the set objectives.

With new features being added, complete testing of the features will be carried out using debug statements as well as unit testing and integration testing, thus making sure all potential discrepancies and risks are captured.

At the end of each sprint, a review will also be conducted to estimate the tasks to be done and the tasks that have been completed, to ensure that no tasks are left uncompleted before moving to the next sprint.

Project Management Tools

Trello was instrumental in organizing the workflow. It enabled the categorization of tasks into boards

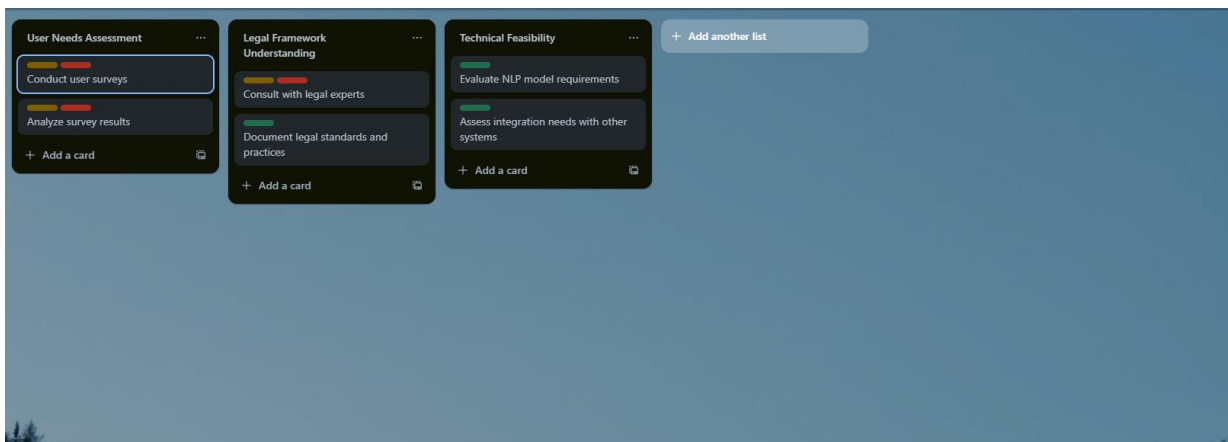


Figure 18 Showing Trello Board for litigat8- 1

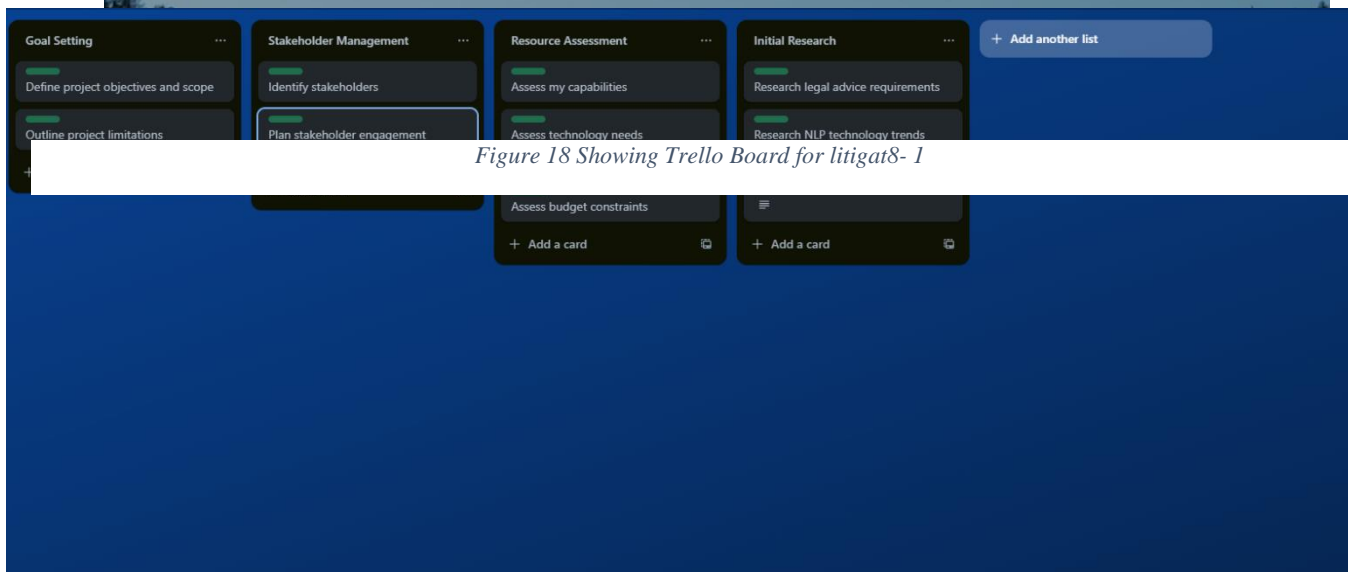


Figure 17 Showing Trello Board for litigat8- 2

representing different Scrum artifacts (product backlog, sprint backlog, in-progress, and done). Trello's visual interface and ease of reorganization catered to the dynamic nature of Agile project management, providing transparency and a high-level overview of the project's status at any given time.

Gantt Chart

Despite Agile's emphasis on flexibility, the project's overarching timeline was charted

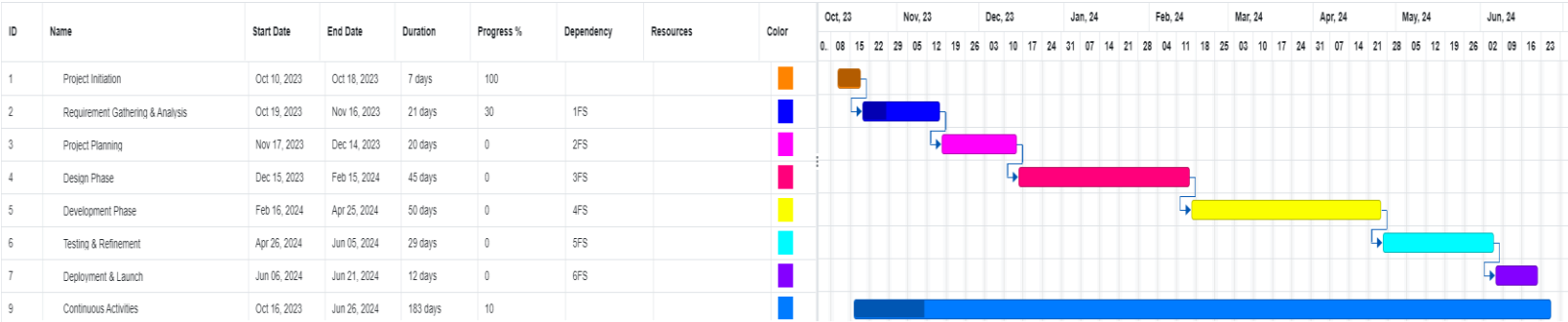


Figure 19 Showing Project Timeline Managment Using Gantt Chart for Litigat8 -1

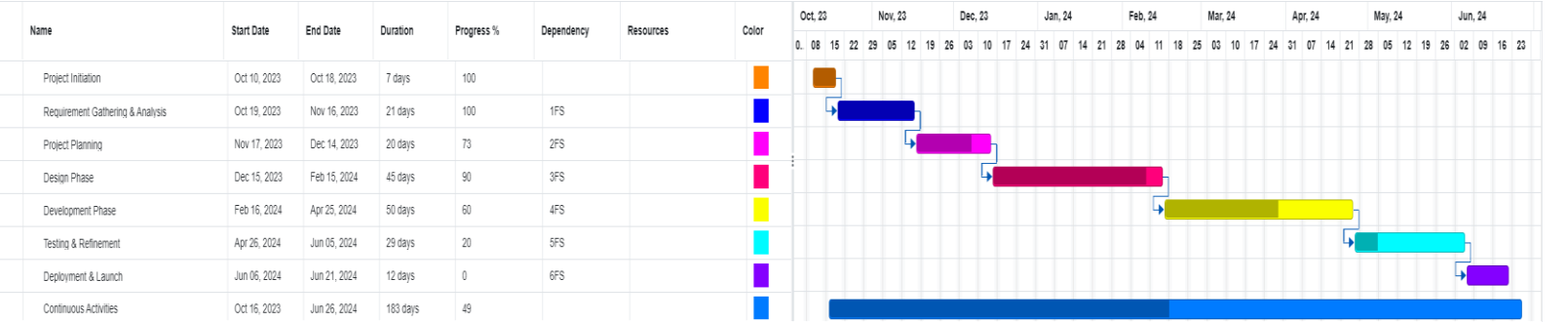


Figure 20 Showing Project Timeline Management Using Gantt Chart for Litigat8 -2

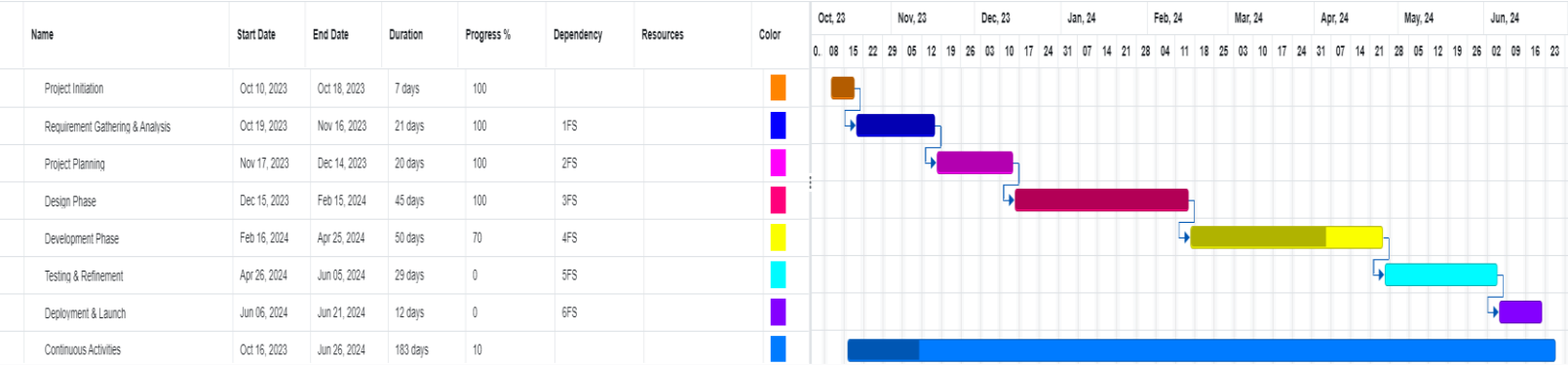


Figure 21 Showing Project Timeline Management Using Gantt Chart for Litigat8 - 3

out using a Gantt chart. This served as a visual planning tool to set key milestones and task durations. It offered a macroscopic view of the project's lifecycle, ensuring adherence to the overarching deadline, and facilitated the management of dependencies between tasks.

Version Control with GitHub

GitHub was selected for version control considering its robust platform for collaborative coding and its prevalence in the software development community. GitHub's branching mechanism allowed for the isolated development of features, with integration into the main codebase after thorough review and testing. This practice not only ensured the integrity and continuity of the development but also encouraged experimental development without risking the system.

The project's GitHub repository acted as a single source of maintaining a controlled environment for the development for the project where each modification was tracked and if there is any problem the changes were thoroughly analyzed and the problems were eliminated following the analysis

Design Phase:

This stage marks the design phase of the application where I would be designing the required systems that would be operating within the

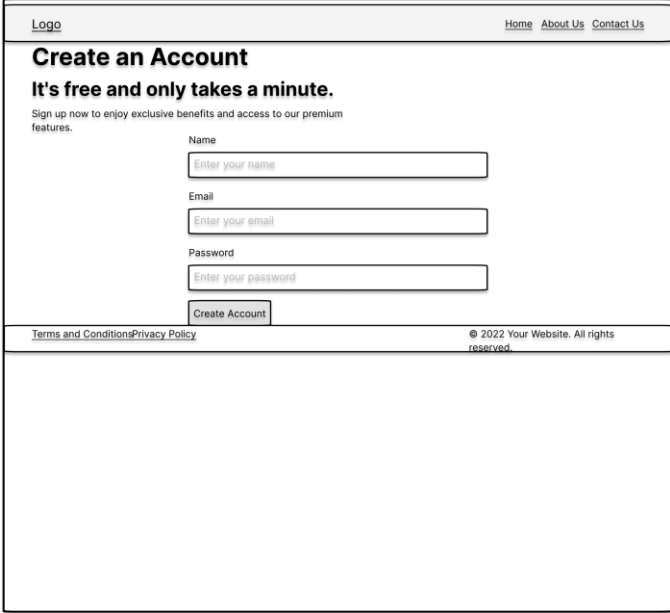
Wireframes:

Wireframes were used to form a design the front-end for the application keeping in mind accessibility and functionality. A great consideration was taken to keep in my mind the requirements as well for the system so that it performs as such as it is planned. The wireframes were designed in Figma using the tools provided by the application. The aim was to make the front-end as intuitive as much as possible so the users don't have any problem navigating through the application.

Registration:

The registration page was designed to accomplish the first objective of having a user authentication system. The frontend was kept as simple as possible with three input placeholders for name/username, email and password. This design would help the users with the ease of registration.

Registration Page

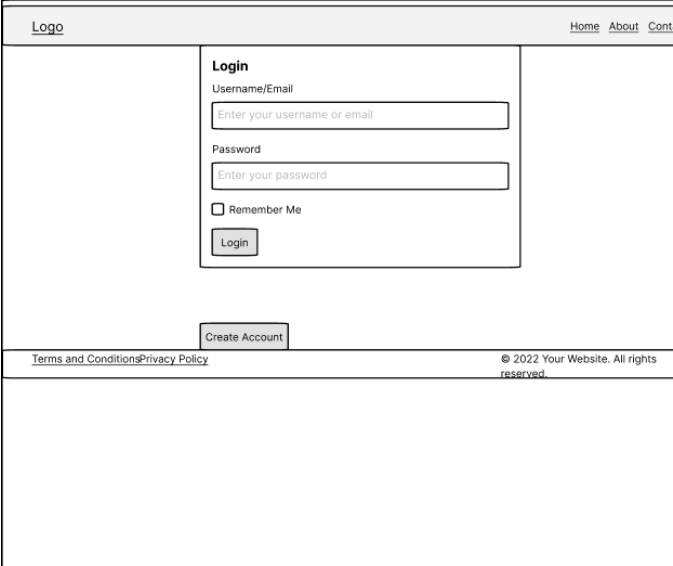


The registration page wireframe features a header with a 'Logo' on the left and navigation links 'Home', 'About Us', and 'Contact Us' on the right. The main content area is titled 'Create an Account' with the subtext 'It's free and only takes a minute.' Below this, a brief description states: 'Sign up now to enjoy exclusive benefits and access to our premium features.' The form includes three input fields labeled 'Name', 'Email', and 'Password', each with a placeholder text 'Enter your [field name]'. A 'Create Account' button is positioned below the password field. At the bottom of the form, there are links for 'Terms and Conditions' and 'Privacy Policy' on the left, and a copyright notice '© 2022 Your Website. All rights reserved.' on the right.

Figure 22 Showing Registration Wireframe

Login:

Extending on the authentication system keeping the same objective in mind to keep it simple. The front-end was designed with two input placeholders for email/username and password. Thus making the logging in process as simple as possible for the User.



The login page wireframe has a header with a 'Logo' on the left and navigation links 'Home', 'About', and 'Cont' on the right. The main content area is titled 'Login' and contains two input fields: 'Username/Email' with a placeholder 'Enter your username or email' and 'Password' with a placeholder 'Enter your password'. Below the password field is a checkbox labeled 'Remember Me' and a 'Login' button. A 'Create Account' button is located below the login form. The footer includes links for 'Terms and Conditions' and 'Privacy Policy' on the left, and a copyright notice '© 2022 Your Website. All rights reserved.' on the right.

Figure 23 Showing Login Wireframe

Chat Main Page:

The main page was designed following the same intuition of accessibility and functionality. The design was kept simple with the input prompt at the bottom and a simple send button to send queries to the NLP model to be processed at the backend. On the Left side is the chat history if the user wants to refer to it at anytime in the future or to go back to reflect on it.



Figure 24 Showing Main Chat Interface Wireframe

The sections concludes the front-end design of the application of the application. As most of the utility would be designing the backend/NLP model. But ample consideration was give to the frontend to improve the user experience as much as possible.

Database Design:

This schema defines a simple yet effective structure for managing users, chat sessions, and conversations in a chat application. The reasoning behind the design choices is as follows:

- **User Model:** Represents the individuals using the chat application. Each user has a unique username and a password. Storing the password implies that it should be securely hashed before storage to ensure security. The uniqueness of the username ensures that each user can be distinctly identified.

User			
int	id	PK	Primary Key
string	username		Unique, Not Null
string	password		Not Null

Figure 25 Showing *User Model*

- **ChatSession Model:** Each session represents a period during which a user is actively engaged in conversation. It is linked to the User model via a foreign key (**user_id**), establishing a one-to-many relationship between users and chat sessions. This means a single user can have multiple chat sessions over time. The **start_time** field automatically records when each chat session begins.

ChatSession			
int	id	PK	Primary Key
int	user_id	FK	Foreign Key to User.id, Not Null
datetime	start_time		Default: Current Timestamp

Figure 26 Showing *ChatSession Model*

- **Conversation Model:** The conversation model Captures individual messages from within a chat session. It includes a foreign key to **ChatSession**, establishing a one-to-many relationship between a chat session. This allows the application to organize messages under their respective chat sessions. Each message has a **type** to distinguish between user and AI messages, supporting the interactive nature of the chat. The **timestamp** field captures the exact time each message was sent, which is crucial for displaying messages in the correct order and context.

Conversation			
int	id	PK	Primary Key
int	chat_session_id	FK	Foreign Key to ChatSession.id, Not Null
string	message		Not Null
string	type		Not Null, 'user' or 'ai'
datetime	timestamp		Default: Current Timestamp

Figure 27 Showing *Conversation Model*

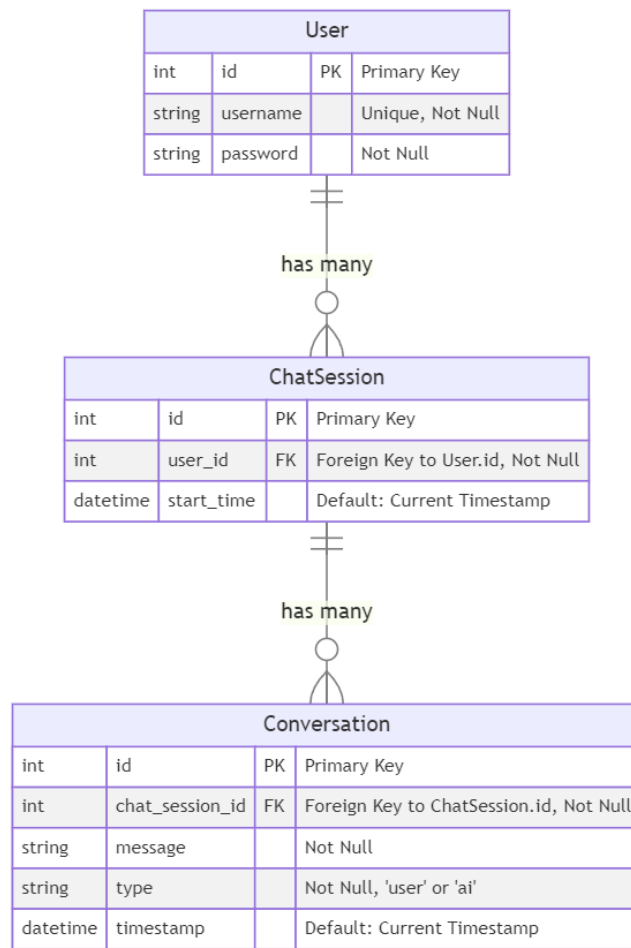


Figure 28 Showing the ERD Diagram for Litigat8 Database

Dataset Workflow/Design:

The proposed dataset will be constructed Pdfs/Text documents available publicly for household and tenant law which include public articles or research papers etc. All of the entities are publicly available to refer and the model would just analyze the text in these to reference later. And they would be put through various data preprocessing steps such as Text Lowercasing, punctuation removal and Tokenization etc. To ensure the efficiency of the dataset and to remove the noise from the dataset. This would increase the overall efficiency of the NLP model to process data.

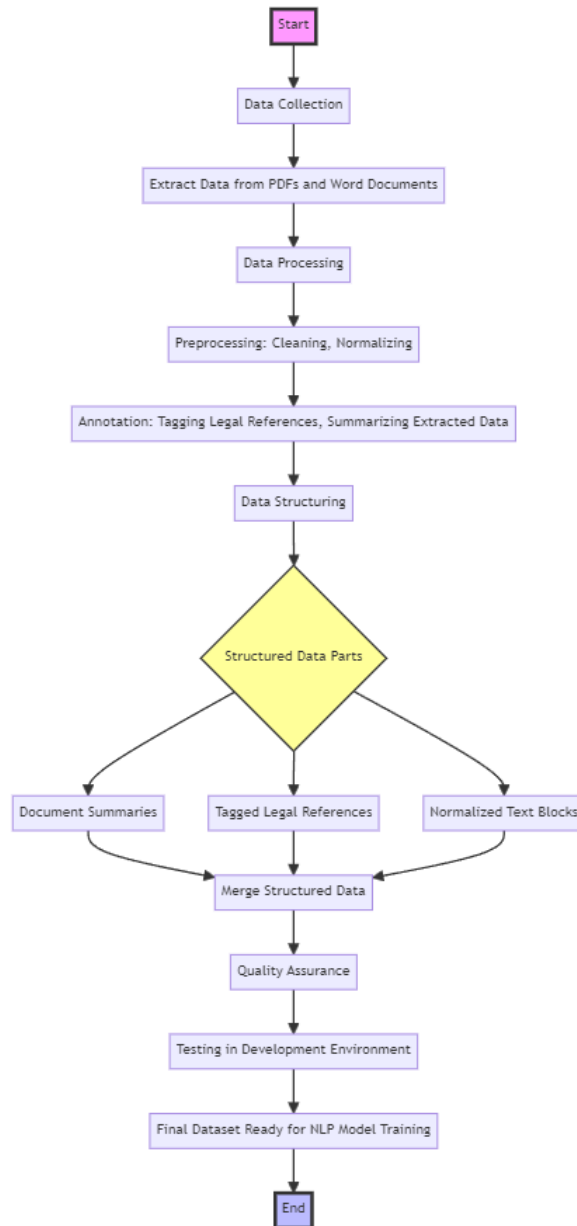


Figure 29 Showing the Framework for the Data Collection and Preparation for Litigat8 Training

The data that would be used would be Analysed before hand to make sure it is from a valid source and it is publicly available. And all the documents/ articles that would be used will be reference appropriately.

LLD Design:

The proposed LLD Design shows a comprehensive technical view of the system which was modelled after doing the requirement analysis of the project.

The LLD Diagram for the project solution was implemented for detailing the implementation scheme of the application as LLD provide a blueprint of how every feature and component within the system would be implemented. The displayed LLD shows the exact programming constructs, algorithm, interface design and data models. There are few reasons why the LLD was proposed the most important one being that LLD ensures good code quality. As it helped me during the development to adhere to the agreed set of standards and maintain consistency. It also enabled

me to reduce risks and uncertainty in the project as everything was modelled before the implementation of the program which helped during the project to mitigate risks.

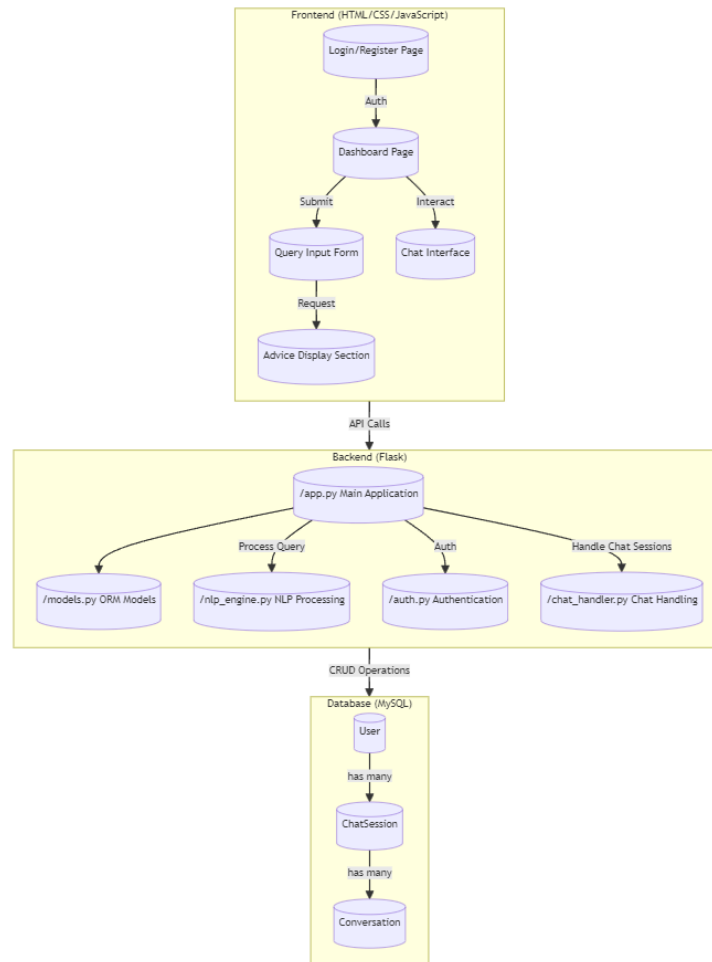


Figure 30 Showing the Low-Level Design (LLD) for Litigat8

With the LLD it the main functionality of the applicaiton can be identified with ease. The front-end would handle all the requests from the User in a structured way making use of JavaScript and AJAX. Ensuring that the information is fed properly from the front-end to the backend on specific routes. The backend is designed to handle all the POST and GET requests that are initialized. The authentication will be handled by programming structure labeled as **auth.py** in the diagram which would communicate with the MySQL database to validate the user details and redirect to the main dashboard page. The **Models.py** would be responsible for making the database tables of Users, ChatSession and Conversation models. The programming structure defined as **chat_handler.py** would be responsible for handing the requests and functionality related to the chat with the NLP model i.e., saving the chat session, generating responses from the **nlp_engine.py** file and handling all the POST and GET requests generated in the meanwhile. The **nlp_engine.py** would handle all the logic related to the model training and preparation and would be responsible for generating the responses based on the user queries.

Implementation:

Database Implementation and Connection:

The database was set up using SQLAlchemy which is a library inside of python. This enabled me to make the dataset dynamically by just making the models of each entity.

```
from .database.models import *
from datetime import datetime
from .blueprints.auth import auth
from .blueprints.chat import chat
def create_app():
    app = Flask(__name__)

    app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql+pymysql://root@localhost:8080/litigat8'

    # app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql+pymysql://root:@localhost:3306/litigat8'
    # app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql+pymysql://root:@viaduct.proxy.rlwy.net:58016/litigat8'

    app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
    app.secret_key = 'your secret key'

    app.register_blueprint(auth, url_prefix='/auth')
    app.register_blueprint(chat, url_prefix='/chat')
    db.init_app(app)
    with app.app_context():
        db.create_all()
```

Figure 31 Showing the Setup of Database Config

With the database initiation its assigned to the flask application by setting up the correct URL, username and password for the database inside of the SQL database. The variable **db** can be used to access any of the model i.e User, ChatSession and Conversation. Making the overall access of the database really easy instead of using SQL statements running directly on MySQL.

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()
```

Figure 32 Setting up the Database

The models set up are made on accordance with the database schema which is displayed in the Figure 25. All of them have a primary key but ChatSession and Conversation have foreign keys and one-to-many relationship with the User_ID.

```

from flask import Flask
from flask_sqlalchemy import SQLAlchemy

from .. import db

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(50), unique=True, nullable=False)
    password = db.Column(db.String(200), nullable=False)

class ChatSession(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    start_time = db.Column(db.DateTime, default=db.func.current_timestamp())
    conversations = db.relationship('Conversation', backref='chat_session', lazy=True)

class Conversation(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    chat_session_id = db.Column(db.Integer, db.ForeignKey('chat_session.id'), nullable=False)
    message = db.Column(db.String(200), nullable=False)
    type = db.Column(db.String(10), nullable=False) # 'user' or 'ai'
    timestamp = db.Column(db.DateTime, default=db.func.current_timestamp())

```

Figure 33 Setting up the Models for Database

Table	Action	Rows	Type	Collation	Size	Overhead
<input type="checkbox"/> chat_session		6	InnoDB	utf8mb4_general_ci	32.0 KiB	-
<input type="checkbox"/> conversation		49	InnoDB	utf8mb4_general_ci	32.0 KiB	-
<input type="checkbox"/> user		3	InnoDB	utf8mb4_general_ci	32.0 KiB	-
3 tables	Sum	49	InnoDB	utf8mb4_general_ci	96.0 KiB	0 B

Figure 34 Showing the constructed Table in the Litigate Database

The chapter concludes the setting up of the database for Litigat8 application.

Front-end Implementation:

Main Home Page:

The main page is designed based on the initial approach of keeping the user interface as intuitive as possible while keeping the functionality of the page. The initial design was designed using wireframes to make sure the it has all the essential components are present in the page.

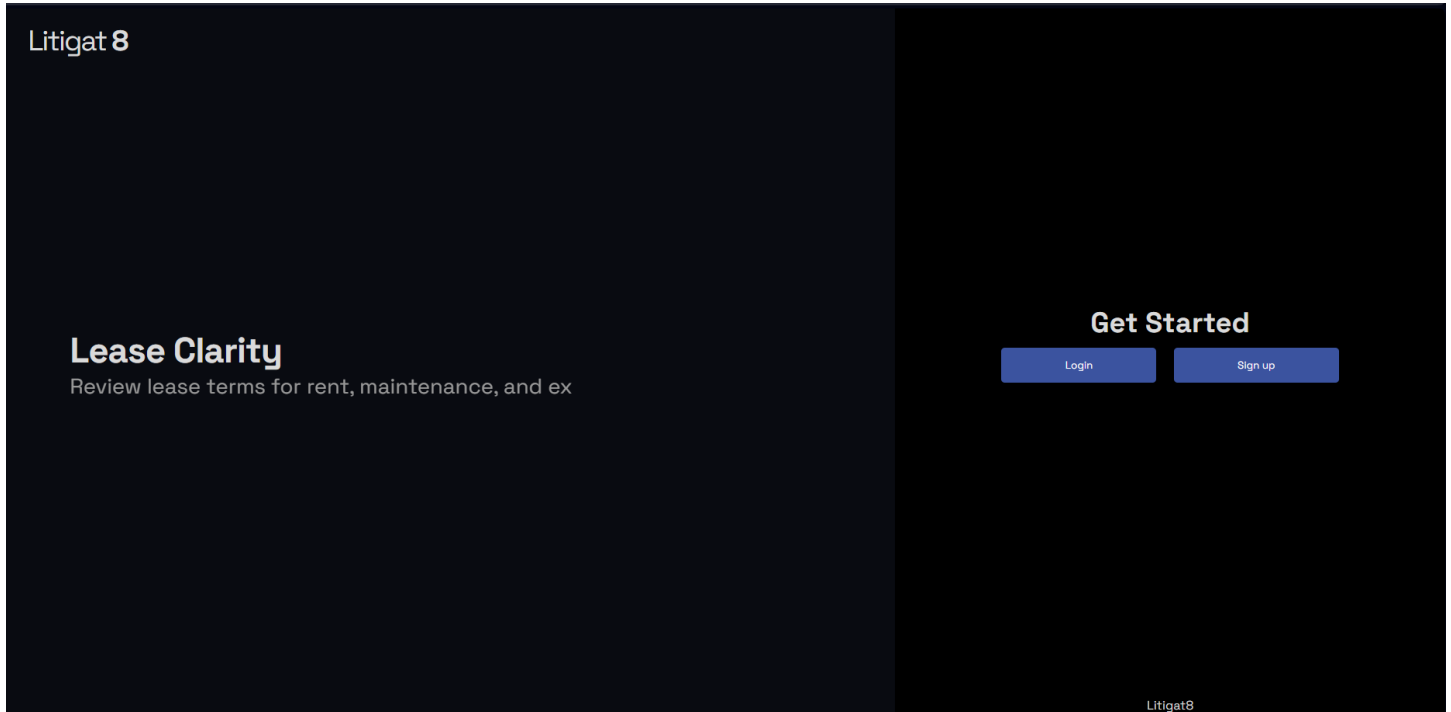


Figure 35 Showing the Front-end Implementation of Main-Page

An additional feature of typewriting was implemented as well as well to give it a bit more dynamic look.

Login And Sign-Up Page:

Following on the intuitive and approachable design of the pages, the design was based on the wireframes that was designed earlier. With two input field in login page and three input field in resistration page asking for the desired input from the users.

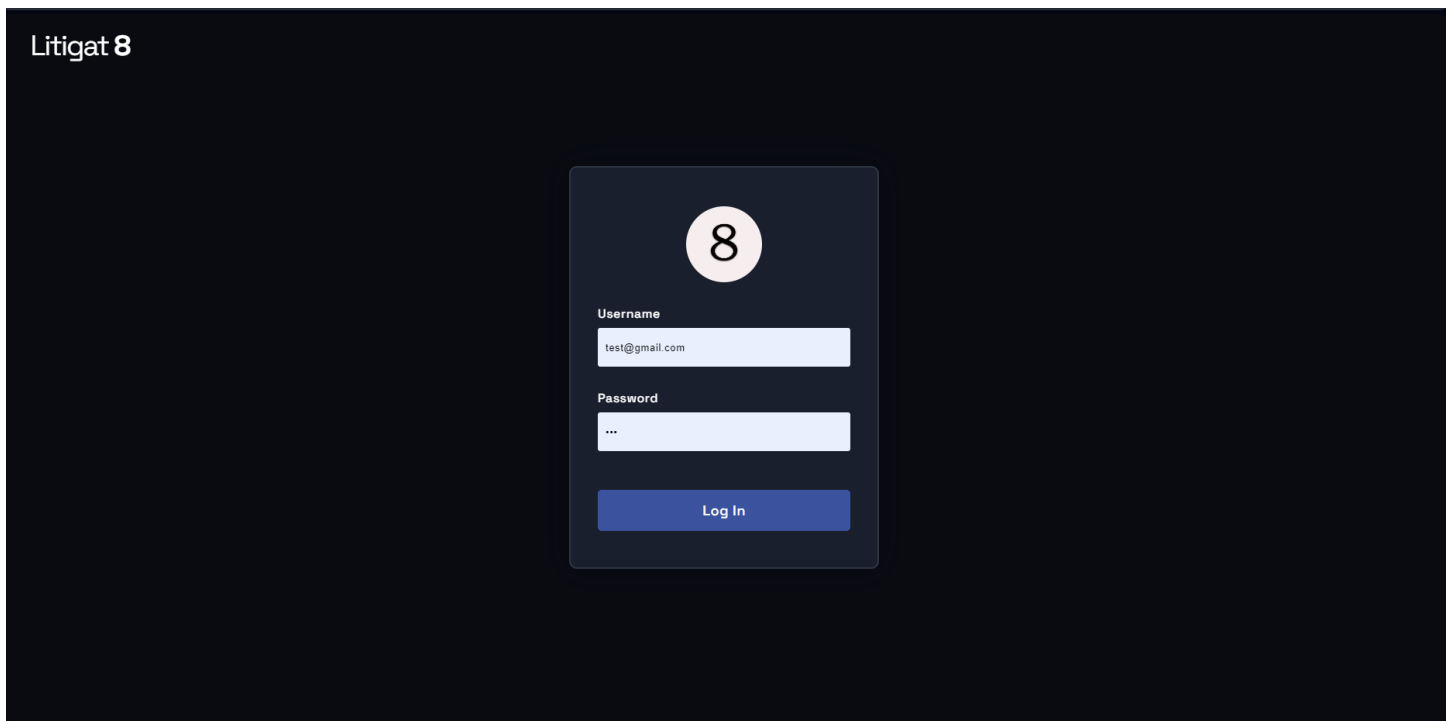


Figure 36 Showing the Front-end Implementation of Login Page of Litigate

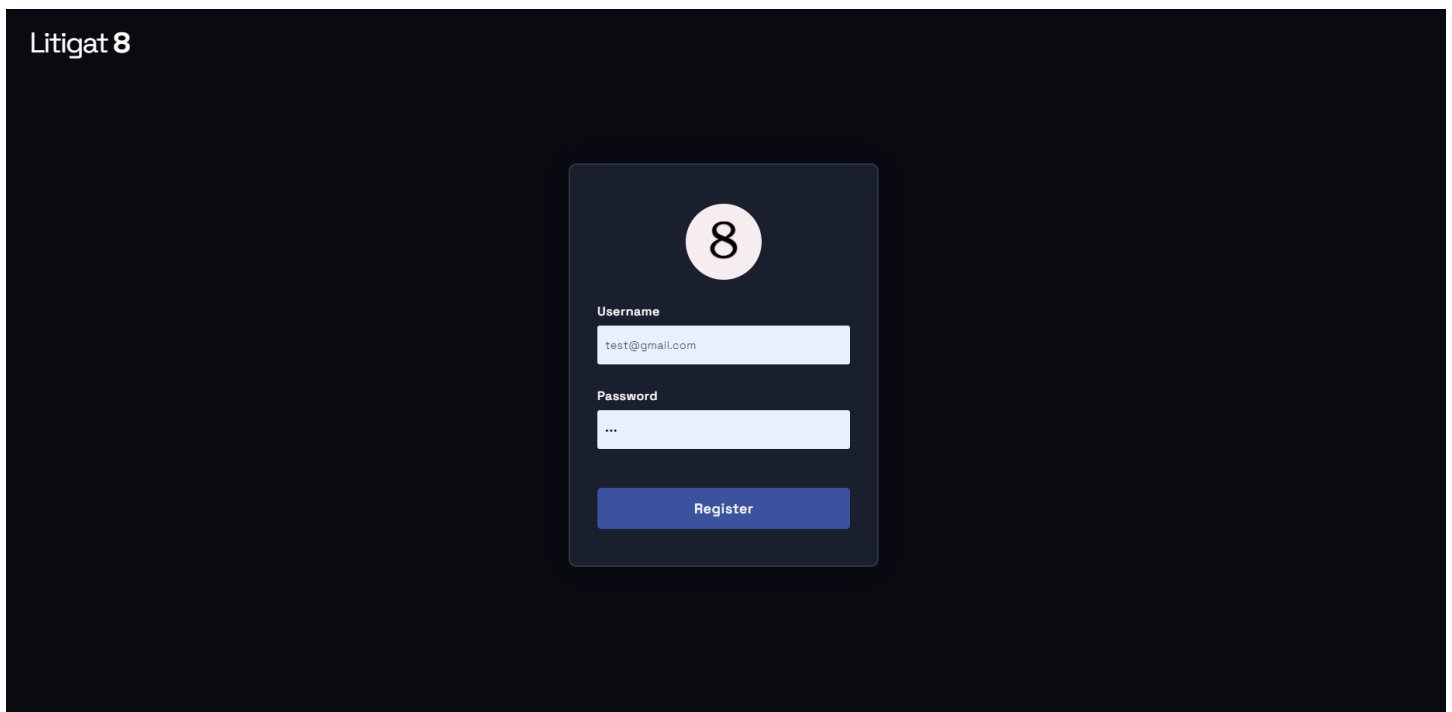


Figure 37 Showing the Front-end Implementation of Register Page of Litigate

Chat Interface:

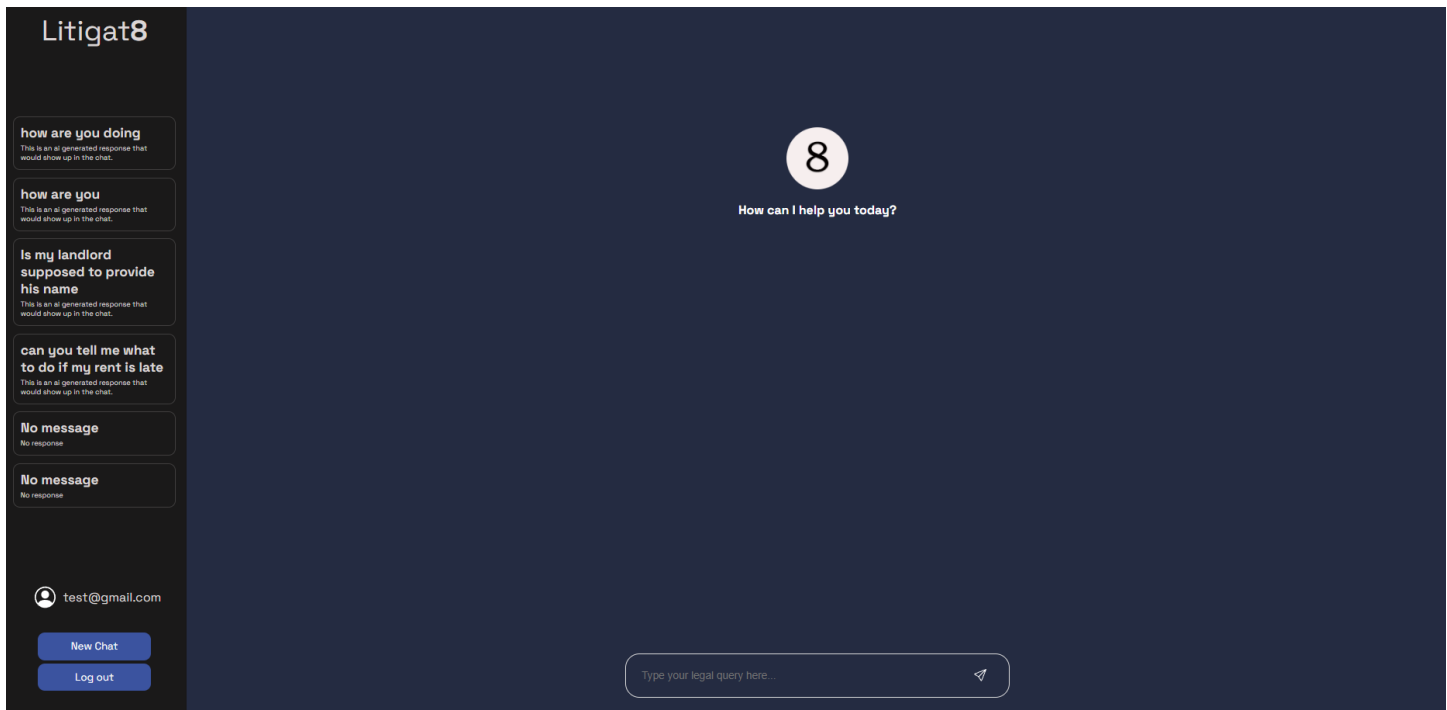


Figure 38 Showing the Front-end Implementation Chat Interface of Litigate

The main chat interface was build using combination of JavaScript, Html and CSS. The page followed the initial requirements set up by wireframes. With all the essential component present in the design. An input form to take in the User queries, a panel to display all the previous chat session with two buttons to either logout of the system or to start a new chat session.

This Marks the end of the front-end design implementation of the Litigat8. All the essential design elements have been put into place for the backend element to work on and make it dynamic. Moving to the next chapter the report would show the implementation of the backend of the system.

Backend-Server Side:

For the implementation of the backend of the Litigat8 app various libraries were used such as **flask**, **SQLAlchmey**, **functools**, **werkzeugsafe** etc. which worked in coordination to provide the implementation of the application based on the plans that were designed in the design phase of the project.

Setting Up the App:

The python flask application was set in the `__init__.py` file which would let you run the application using a simple **flask run** command in the CLI which is much more convenient then setting it in a different file that's why I decided to go with that approach. The functionalities of the app was made modular and important functions stored in different files. All the necessary

imports were made in the `__init__.py` file before setting up the app. The `creat_app()` func sets up the application for us assimilating all the routes and database configuration.

```
from flask import *
from markupsafe import escape
from werkzeug.security import generate_password_hash, check_password_hash
from functools import wraps
from .extensions import db
import requests
import openai
from openai import OpenAI
import os
from .database.models import *
from datetime import datetime
from .blueprints.auth import auth
from .blueprints.chat import chat
def create_app():
    app = Flask(__name__)

    app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql+pymysql://root@localhost:8080/litigat8'

    # app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql+pymysql://root@localhost:3306/litigat8'
    # app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql+pymysql://root@viaduct.proxy.rlwy.net:58016/litigat8'

    app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
    app.secret_key = 'your secret key'

    app.register_blueprint(auth, url_prefix='/auth')
    app.register_blueprint(chat, url_prefix='/chat')
    db.init_app(app)
    with app.app_context():
        db.create_all()

def login_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        if 'user_id' not in session:
            flash('You need to be logged in to view this page.')
            return redirect(url_for('auth.login'))
        return f(*args, **kwargs)
    return decorated_function
```

Figure 39 Showing the implementation of Flask App with Configuration

For setting up the routes for the flask application instead of setting it up directly in the app. Blueprints were used to set up the routes respective to their functionality. Two main blueprints were set **chat.py** and **auth.py** the **chat.py** handles all the routes to handle the server side functionality of chat functions linked with the specific routes. Whereas **auth.py** handles the logic used for authentication and registration of the user.

Setting Up Blueprints:

```
from flask import Blueprint, request, render_template, redirect, url_for, flash, session
from werkzeug.security import generate_password_hash, check_password_hash
from ..extensions import db # Import your database instance from your main app module
from ..database.models import User # Import your User model

# Creating the Blueprint
auth = Blueprint('auth', __name__)
```

Figure 40 Showing the Implementation of Auth Blueprint

```
from flask import Blueprint, request, jsonify, session, url_for, redirect
from datetime import datetime
from ..extensions import db # Make sure to import your database instance
from ..database.models import ChatSession, Conversation # Import your models
from werkzeug.security import check_password_hash, generate_password_hash
from ..nlp.nlp import get_ai_response # Import your AI response function
# Creating the chat Blueprint
chat = Blueprint('chat', __name__)
```

Figure 41 Showing the Implementation of Chat Blueprint

Authentication Handling:

The authentication of the User were done by using POST requests that are originated from the front-end. As a result of the POST request the function queries the database to check if the user exists or not and handles the results accordingly to the outcome. And all the routes that needed login of the user were looked using wraps from functools which prevent unauthorized access to the system thus making the application more secure

```
@auth.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        user = User.query.filter_by(username=username).first()
        if user and check_password_hash(user.password, password):
            session['user_id'] = user.id
            return redirect(url_for('main_chat')) # Assuming 'main' is the Blueprint name for your main app routes
        else:
            flash('Invalid username or password.')
    return render_template('login.html')
```

Figure 42 Showing the setting up of Login Function Server Point

```
@auth.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        hashed_password = generate_password_hash(password)
        new_user = User(username=username, password=hashed_password)
        db.session.add(new_user)
        db.session.commit()
        return redirect(url_for('auth.login')) # Note the Blueprint name prefix
    return render_template('register.html')
```

Figure 43 Showing the setting up of Register Function Server Point

Registration is done with a simple POST request as well to the URL **auth/register/** which on being called inserts the User details into the database. And all the user passwords were encrypted to make sure there passwords are kept safe.

```
@auth.route('/logout')
def logout():
    session.pop('user_id', None)
    return redirect(url_for('index')) # Assuming 'main.index' is the route to your main page
```

Figure 44 Showing the setting up of Logout Function Server Point

The logout simply just pops out the user details if they exist in the flask application.

Chat/Response Handling Generation and Chat Session Handling:

Python Server Side to Handle Requests:

This section shows the implementation of how chat session are stored and retrieved. The **start_chat_session()** take in both kind of requests POST and GET. The function works by making Chat Session entry whenever the function is call and then the chat session id is stored in the session to be used later.

```
chat.route('/start_chat_session', methods=['POST','GET'])
def start_chat_session():
    user_id = session.get('user_id')
    if not user_id:
        return jsonify({'error': 'User not authenticated'}), 401

    new_chat_session = ChatSession(user_id=user_id, start_time=datetime.utcnow())
    db.session.add(new_chat_session)
    db.session.commit()

    # Set the chat_session_id in the user's session
    session['chat_session_id'] = new_chat_session.id

    return jsonify({'message': 'New chat session created', 'session_id': new_chat_session.id}), 201
```

Figure 45 Showing the implementation of start_chat_session Server Point

The **/end_chat_session** works just by removing the chat_session_id from the session thus making not accessible anymore.

```
@chat.route('/end_chat_session', methods=['POST','GET'])
def end_chat_session():
    if 'chat_session_id' in session:
        chat_session_id = session['chat_session_id']
        # Here you can update the database to mark the chat session as ended, if needed
        session.pop('chat_session_id', None) # Remove chat_session_id from session
        return redirect(url_for('chat.start_chat_session'))
    return jsonify({'error': 'No active chat session'}), 400
```

Figure 46 Showing the implementation of end_chat_session Server Point

The **/chat/submit** route is linked with **submit()** function that calls onto the **ai_response()** func which is imported from **app.nlp.nlp_engine** which serves as the base function for generating the AI response which then gets send to the font-end as JSON object to be printed out in a user understandable form. Using JSON to transfer data comes in really handy given the flexibility it provides.

```

@chat.route('/submit', methods=['GET', 'POST'])
def submit():
    user_id = session.get('user_id')
    interactions = [] # Adjust as needed

    if request.method == 'POST':
        print("this works")
        if request.is_json:
            data = request.get_json()
            user_input = data.get('user_input')
        else:
            user_input = request.form.get('user_input')
        print(user_input)
        if not user_input:
            return jsonify({'error': 'No user input provided'}), 400

        # Example process, adjust as needed
        ai_response = get_ai_response(user_input)

        interactions.append({'type': 'user', 'text': user_input})
        interactions.append({'type': 'ai', 'text': ai_response})

        return jsonify({'user_input': user_input, 'ai_response': ai_response})

```

Figure 47 Showing the implementation of server handling of /submit route

The function **save_interaction()** makes use of a helper function **save_conversation_message()** they both work to gather to save the interaction between the User and the AI to be referred later to be stored with a reference to the chat session I used helper function instead of defining the functionality in the save_interaction func because it helped me made the code more modular thus more robust. And the other function **get_conversation/session_id** was designed to get all the conversation stored in the database for a specific id. Which then are displayed on the front-end in a structured way.

```

@chat.route('/save_interaction', methods=['POST'])
def save_interaction():
    data = request.json
    user_id = session.get('user_id')
    # Retrieve the current chat session ID from the session or other means
    chat_session_id = session.get('chat_session_id') # Ensure this is being set somewhere in your application

    user_input = data['user_input']
    ai_response = data['ai_response']

    # Save both messages with their respective types
    save_conversation_message(user_id, chat_session_id, user_input, 'user')
    save_conversation_message(user_id, chat_session_id, ai_response, 'ai')

    return jsonify({'status': 'success'})

def save_conversation_message(user_id, chat_session_id, message, message_type):
    # Create a new message with the chat session ID
    new_message = Conversation(chat_session_id=chat_session_id, message=message, type=message_type)
    db.session.add(new_message)
    db.session.commit()

```

Figure 48 Showing the implementation of server handling of /save_interaction route

```

@chat.route('/get_conversations/<int:session_id>', methods=['GET'])
def get_conversations(session_id):
    user_id = session.get('user_id')
    if not user_id:
        return jsonify({'error': 'User not authenticated'}), 401

    chat_session = ChatSession.query.filter_by(id=session_id, user_id=user_id).first()
    if not chat_session:
        return jsonify({'error': 'Chat session not found or access denied'}), 404

    conversations = Conversation.query.filter_by(chat_session_id=session_id).order_by(Conversation.timestamp.asc()).all()

    conversation_data = [{
        'type': conv.type, # Use the 'type' directly from the model
        'message': conv.message
    } for conv in conversations]

    return jsonify(conversation_data)

```

Figure 49 Showing the implementation of server handling of /get_conversation/session_id route

The loading of all the chat session to be displayed for the user was done by the use of **fetch_chat_histories()** function which gets all the conversation enteries between the user and ai which have the same chat session id. It is then converted into a json object which then gets displayed at the front-end with the help of Javascript.

```

@chat.route('/fetch_chat_histories')
def fetch_chat_histories():
    user_id = session.get('user_id')
    if not user_id:
        return jsonify({'error': 'User not authenticated'}), 401

    chat_sessions = ChatSession.query.filter_by(user_id=user_id).all()

    chat_histories = []
    for chat_session in chat_sessions:
        # Fetch the first user and AI messages in this session as a snippet
        user_message = Conversation.query.filter_by(chat_session_id=chat_session.id, type='user').first()
        ai_message = Conversation.query.filter_by(chat_session_id=chat_session.id, type='ai').first()

        chat_histories.append({
            'session_id': chat_session.id,
            'user_snippet': user_message.message if user_message else 'No message',
            'ai_snippet': ai_message.message if ai_message else 'No response'
        })

    return jsonify(chat_histories)

```

Figure 50 Showing the implementation of server handling of /fetch_chat_histories route

JavaScript to Initiate fetch Requests to the Server Endpoint and Handle the JSON responses dynamically:

This section marks the implementation of JavaScript that's implemented in the front-end to which send fetch requests to server side to get the required data from the database and display it to the user.

```

function fetchChatHistories() {
  fetch('chat/fetch_chat_histories')
    .then(response => response.json())
    .then(data => {
      const chatHistoryPanel = document.getElementById('chat-history-panel');
      data.forEach(chat => {
        const chatRow = document.createElement('div');
        chatRow.className = 'chat-history-row';
        chatRow.innerHTML = `
          <div class="chat-history-link" data-session-id="${chat.session_id}">
            <p class="user-chats">${chat.user_snippet}</p>
            <p class="ai-chats">${chat.ai_snippet}</p>
          </div>
        `;
        chatHistoryPanel.appendChild(chatRow);

        // Add click event listener to the new chat history row
        chatRow.querySelector('.chat-history-link').addEventListener('click', function(event) {
          event.preventDefault(); // Prevent the default anchor action
          const sessionId = this.getAttribute('data-session-id');
          fetchAndDisplayConversations(sessionId);
        });
      });
    })
    .catch(error => console.error('Error fetching chat histories:', error));
}

```

Figure 51 Showing the implementation of Fetch Requests of chat/fetch_chat_history

```

async function startNewChatSession(firstMessage) {
  try {
    const response = await fetch("chat/start_chat_session", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({}),
    });

    if (response.ok) {
      const data = await response.json();
      console.log("New chat session started:", data);
      sessionStorage.setItem("session_id", data.session_id);

      // Now send the first message
      sendUserInput(firstMessage);
    } else {
      throw new Error('Failed to start a new session');
    }
  } catch (error) {
    console.error("Error starting new chat session:", error);
  }
}

```

Figure 52 Showing the implementation of Fetch Requests of chat/start_chat_session


```

async function sendUserInput(userInput) {
  const response = await fetch("chat/submit", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify({ user_input: userInput }),
  });

  if (response.ok) {
    const data = await response.json();
    const aiResponse = data.ai_response;

    // Save interaction

    // Update UI with user input and AI response
    await saveInteraction(userInput, aiResponse);

    appendUserInput(userInput);
    appendAiResponse(aiResponse);
  } else {
    console.error("Failed to submit or get a response");
  }
}

```

Figure 53 Showing the implementation of Fetch Requests of chat/submit

```

function saveInteraction(userInput, aiResponse) {
  fetch("chat/save_interaction", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify({
      user_input: userInput,
      ai_response: aiResponse,
    }),
  })
  .then((response) => response.json())
  .then((data) => console.log("Success:", data))
  .catch((error) => console.error("Error:", error));
}

```

Figure 54 Showing the implementation of Fetch Requests of chat/save_interaction

```

function fetchAndDisplayConversations(sessionId) {
  console.log('Fetching conversations for session:', sessionId); // Confirm the function is called

  fetch(`/chat/get_conversations/${sessionId}`)
    .then(response => response.json())
    .then(conversations => {
      const outputContainer = document.getElementById("output"); ...
      function processMessage(index) {
        if (index >= conversations.length) {
          console.log('All messages displayed.');
```

Figure 55 Showing the implementation of Fetch Requests of chat/get_conversations/sessionID and Processing the of JSON load to front-end

```

document.getElementById("start-new-session").addEventListener("click", async function() {
  try {
    const response = await fetch("/chat/start_chat_session", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      // Include any necessary data in the body, if your endpoint requires it
      // body: JSON.stringify({}),
    });

    if (response.ok) {
      // If the session is successfully started
      const data = await response.json();
      console.log("New chat session started:", data);

      // Optionally: Store the session ID in sessionStorage or handle it as needed
      sessionStorage.setItem("session_id", data.session_id);

      // Redirect to the new chat page
      window.location.href = '/chat'; // Adjust the URL as needed
    } else {
      throw new Error('Failed to start a new session');
    }
  } catch (error) {
    console.error("Error starting new chat session:", error);
  }
});
```

Figure 56 Showing the implementation of Fetch Requests of chat/start_chat_session

NLP Model Implementation:

This chapter marks the implementation of the NLP Model. The steps that were taken to make a functioning LLM that would answer the User queries about household and tenant law would be

discussed in this chapter. Many libraries were used to make the final proof-of-concept model for the application such as Langchain, OpenAI, Chroma, OpenAI, NLTK etc. These libraries served the foundation on which the application was build.

Data Preprocessing/Text Preprocessing:

Before starting with the development of the model, it was essential that the data is preprocessed before any work was done using it thus many preprocessing steps were taken to ensure there aren't any discrepancies in the data. Four major preprocessing steps were taken , the first one being tokenization of the text. Which essentially breaks down the text into smaller units called tokens. There are different types of tokenization's word tokenization, character tokenization etc. but in my case I decided to use word tokenization which is based on Penn Treebank tokenization because of the fact it can handle complex cases, such as contractions(e.g., splitting "don't" into "do" and "n't") and special punctuation patterns, thanks to its underlying use of regular expressions and the Penn Tree Bank tokenization standards.

```
Contents lists available at ScienceDirect
Geoforum
journal homepage: www.elsevier.com/locate/geoforum
Assembling a 'kind of' home in the UK private renting sector
Adriana Mihaela Soaita,a,b,*, Kim McKeec,b
aUniversity of Glasgow, Urban Studies, 25 Bute Garden, Glasgow G12 8RS, United Kingdom
bThe UK Collaborative Centre for Housing Evidence (CaCHE), Olympia Building, 3rd Floor, Glasgow G40 2QH, United Kingdom
cUniversity of Stirling, Housing Studies, Rm 3T18, R.G. Bomont Building, Stirling FK9 4NF, United Kingdom
ARTICLE INFO
Keywords:
Assemblage theory
Home making
Private renting sector
United KingdomABSTRACT
Drawing on assemblage-thinking and specific assemblage concepts, this article explores the ways in which
young, less affluent people create a sense of home in an unregulated, market-based private renting sector (PRS) that confers reduced
tenant agency and frequent, undesired residential mobility. Concept of 'home-assembling' to account for the ontologically, normatively and emotionally different processes
involved in constructing a sense of home than those connoted by home-making.
Through in-depth telephone interviews and photo elicitation, we explore the transient, incomplete nature of
practices of home personalisation; the destabilising effect of broken things which erode the sense of home and
instill feelings of unworthiness; and processes of de-territorialisation, particularly unwanted real/feared re-
location, space sharing and confinement in small rooms. We document that the struggle to continually assemble, de-assemble and re-assemble a sense of home drastically reduces private
anxiety, depression and alienation. However, we also indicate potential lines of change towards alternative
futures not least by the emergence of tenants' 'collective body' as well as by casting tenants' housing ill-being as a matter of public concern.
1. Introduction
Every dwelling, owned or rented, is an aggregate of materials,
money, emotions and practices while concomitantly serving as a roof
over one's head, a place of home, an investment vehicle, a store of
wealth and a symbol of status (Bourdieu, 1989; Cook et al., 2013). The
idea of assemblage is one way of understanding how these hetero-
geneous material, social and emotional components co-function as an
emerging 'whole'—e.g. 'tenure', 'home', 'neighbourhood'—while also
participating in other socio-spatial formations, such as the financial
sector or the built ecology of the city. The notion of assemblage has
been generally employed to understand emerging formations at the
large scale of the global (Acuto and Curtis, 2014; Collier, 2006), the
region (Allen and Cochrane, 2010) and the city (Jacobs, 2012; McCann,
2011; McFarlane, 2011b). We wish to engage it to the small scale of the
```

Figure 57 Showing the text data structure before pre-processing steps

The second pre-processing was lowercasing of all the words in a token. This made the whole text normalized which is essential to make sure there are not discrepancies in the corpus of the data.

The third pre-processing step that was taken was the removal of stop words in the case of English stop words like is, the, in don't generally contribute to the meaning of a text for the analysis process thus they are removed from the dataset in the case of Litigat8 as well.

✓ 0.05

The final step that was taken for the preprocessing was chunking of the data or splitting of the day of set size and of set character overlap. This help me increase the performance of the model as they have limit to input length of the tokens in case of the models that were used in Litigate from 512 to 4090. And the character overlapping helped me preserve the context of the tax and made sure it wasn't lost.

page_content= 'Contents lists available at ScienceDirect\nGeoforum\njournal homepage: www.elsevier.com/locate/geoforum\nAssembling a 'Kind of' home in the UK private renting sector\nAddress: page_content= 'young, less affluent people create a sense of home in an unregulated, market-based private renting sector (PRS) that confers reduced tenancy and frequent, undesired residential mobility. Forti page_content= 'anxiety, depression and alienation. However, we also indicate potential lines of change towards alternative futures not least by the emergence of tenants' 'collective body' as well page_content= 'been generally employed to understand emerging formations at the large scale of the global (Acuto and Curtis, 2014; Collier, 2006), the region (Allen and Cochrane, 2010) and page_content= 'ality, region or cosmos (Heidegger, 1971), we wish to refocus the concept of home-making on the materialities of home for 'there is no nowhere without innerspace' (Bryden, 201 page_content= 'because the space is physically inappropriate or unaffordable' (Lancione, 2018; Lloyd and Vasta, 2017; Tete, 2012). The new materialist ontologies (Bennett, 2010; De la page_content= 'scholarship that has mostly remained centred on normative meanings and qualities (Clapham, 2011). Assemblage-thinking allows us to attend to 'the small agency' of things (Bennett, 2010, page_content= 'with some Nordic and central European countries offering regulated, 'insecure PRSs in which renters enjoy similar occupancy arrangements with homeowners' (Hulse et al., 2011 page_content= 'McKee et al., 2017; Soaita et al., 2016). To problematise 'generation and rent' discourses, we focus on less affluent young people for whom private renting is likely to be a page_content= 'rality of home in a growing PRS is long overdue (Lloyd and Vasta, 2017). A critical discussion of the labour of (re)assembling a sense of 'home in the UK's PRS poses important page_content= 'bly to the study of home rather than housing, we aim to contribute much needed 'thick descriptions' (Anderson et al., 2012) to the assemblage-informed research; our page_content= 'materialities of self, i.e. the attempt to personalise home through objects. Section 5 focuses on the agency of broken things and their capacity to destabilise the home. See page_content= '2. Home as assemblage\nThe currency of assemblage-thinking is stronger in urban and political studies where it has been applied at the large scale of the plane (Vinty, 2014; gl page_content= 'well suited to understand and how tenants construct a sense of home in an unregulated market-based PRS. There are different theoretical routes to thinking about assemblage (see, e.g. relate page_content= 'p. 2). In the idea of assemblage, can be grasped through metaphors, such as 'the archipelago as the assemblage informed by different individual islands; the page_content= 'simplest division cuts between regulated/secure and unregulated/insecure PRSs. However, this binary should be conceived as a continuum in order to account for the multi page_content= 'organisation; it is employees and building agents of the whole but they cannot be reduced to the organisation since people are concomitantly parts of other unrelated assemblages page_content= 'DeLanda exemplifies this by a knife having the material property of being sharp and the actual/virtual capacities of scratching, cutting or killing that which can be scratched page_content= 'on space/time availability, particularly in tight housing markets. The concept of 'individual singularity' (Normark, 2009, p. 432) is key to 'rethink home as an assemblage of a page_content= 'home and the expression of self through the material expressivity of cherished objects, e.g. family photographs, a teapot or architectural features (Cieraad, 2006; Marci page_content= 'lenders over home buyers and landlords or tenants' (Hulse et al., 2011; Martin et al., 2018). Consequently, in homeownership societies, 'middle-class meanings of home as heaven, hi page_content= 'tinct from the chaotic space of cosmos (Buchanan and Lambert, 2005). The home assemblage can be territorialised/stabilised through habit, 'personalisation of space, performance of page_content= 'interchangeably with (de)stabilisation, these concepts are conceived in relative and absolute terms (Buchanan and Lambert, 2005; DeLanda, 2006, 2016): an assemblage can destabilise page_content= 'desired residential mobility caused by divorce/separation, war or eviction. While tenants move for various reasons, it is telling that none 'think of doing privately-renting household page_content= 'constructing a sense of home in an unregulated, market-based PRS involves an ontologically, normatively and emotionally different relational process than home-making page_content= 'the 'small agency' of things (Bennett, 2010, p. 95), which have 'sufficient coherence to make a difference, produce effects, alter the course of events' (idem, p. vii) (page_content= '2010). Consequently, some authors have chosen to combine assemblage 'theory with the Bourdieusian concept of field (Lovell and Smith, 2010) or complexity theory (Dittmer, 2014). Others have page_content= 'present and imagining alternative futures. Dittmer's (2014, p. 394) 'unborrowed concept of 'bodies politic' is helpful in capturing emerging calls for systemic change given page_content= 'the home-assemblage affects its (de)territorialisation and the capacities of things. To reiterate, assemblage-thinking enables us to conceptualise home 'in a simple and novel way as ' page_content= 'spaces of possibility. Given the constrained use and high residential mobility within the UK's PRS, the axis of (de)/(re)territorialisation is best suited to represent our findings page_content= 'decided by participants. Participation was sourced via social media (project Twitter and Facebook; n=4) and public online platforms (Shelter, Generation Rent, ACORN 2012, page_content= 'nexus between high rents and low/insecure wages has clearly framed participants' home experiences, primarily in terms of afforded housing quality and capacity to personalise page_content= 'interviews but more importantly, they directed our attention to the labour of home-assembling, inductively inspiring our way of thinking, 'prompting immersion in the assemblage page_content= '(Bennett, 2010, p. xiv), with codes/themes attending to practices (and doing), materiality (being) and discourse (saying). Given our small, exploratory sample, we refrain from

Figure 61 Data After Chunking/Splitting

The next step in the development was embeddings generation which are vector representation of text, words, phrases and entire documents. It maps the entities into a continuous multi-dimensional space where it reflects semantic relationships which exist in the space. There are number of reasons I employed this technique first was to establish semantic relationships between the words which improved my model's similarity search functions. Which resulted in overall improvement of the model. Two different models were explored for making the embeddings, OpenAI embedding model **text-embedding-3-large** and hugging face **all-MiniLM-L6-v2**, they

Both were an excellent choice when generating the embeddings the only deciding factor to go with OpenAi was time it required to make the emdeddings which was relatively faster than hugging face model.

```
embedding = OpenAIEmbeddings()
```

Figure 62 Showing the implementation of OpenAI Embedding Model

```
#download embedding model
def download_hugging_face_embeddings():
    embeddings = HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-L6-v2")
    return embeddings

embeddings = download_hugging_face_embeddings()
```

Figure 63 Showing the implementation of Hugging Face Embedding Model all-Mini-LM-L6-v2

Vector Database Generation:

The next step was to create vector database to store all the vectors that are created as a result of embeddings. The reason a vector database was employed was because of traditional database such as MySQL are not optimized to type of queries machine learning applications require such as finding the nearest neighbors in a high dimensional space such as in the case of Litigat8. Setting up the vector database allowed me to do functions such as semantic search and dynamic content discovery.

```
#Initializing the Pinecone
pinecone.init(api_key=PINECONE_API_KEY,
              environment=PINECONE_API_ENV)

index_name="Litigat8"

#Creating Embeddings for Each of The Text Chunks & storing
docsearch=Pinecone.from_texts([t.page_content for t in text_chunks], embeddings, index_name=index_name)
```

Figure 64 Showing the Implementation of Pinecone Vector Database

While making the vector database two different libraries were Analysed **Pinecone and Chroma**. Both of these libraries had there merits the Pincone database was an online vector database while chroma was a database on disk. Traditionally for larger project **Pinecone** would be the ideal

choice and hence that option was explored first but due to recent development in the **Pinecone** structure some of there methods and function have been depreciated while there were alternatives to that but the library that I was using for Langchain I was using wasn't compatible for that. Ample time was given to make Pinecone work but due to not being enough resources available online. I wasn't able to get it functioning with my model hence I decided to look for alternatives instead and thus decided to settle for an on disk database built using **Chroma**. Even though it might not be the ideal choice but for the scope of this application it provided enough value.

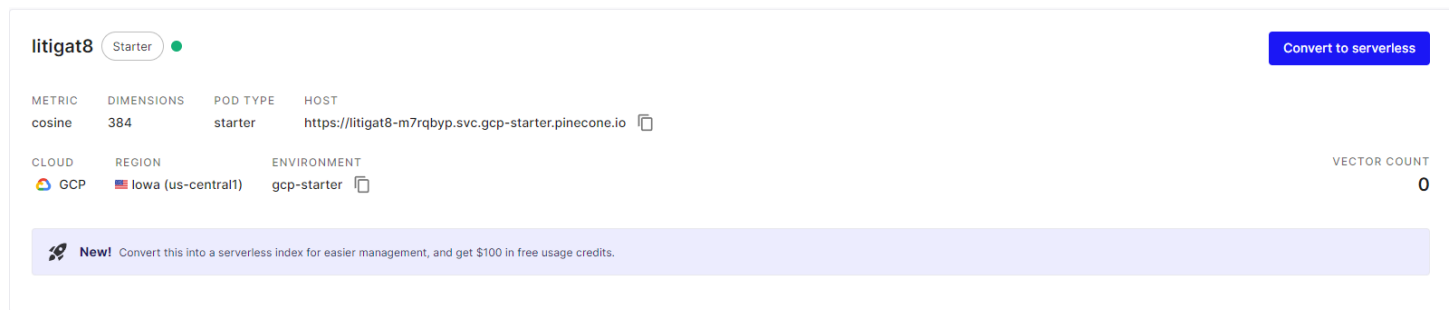


Figure 65 Showing the Setting Up of Pinecone Database on the Website

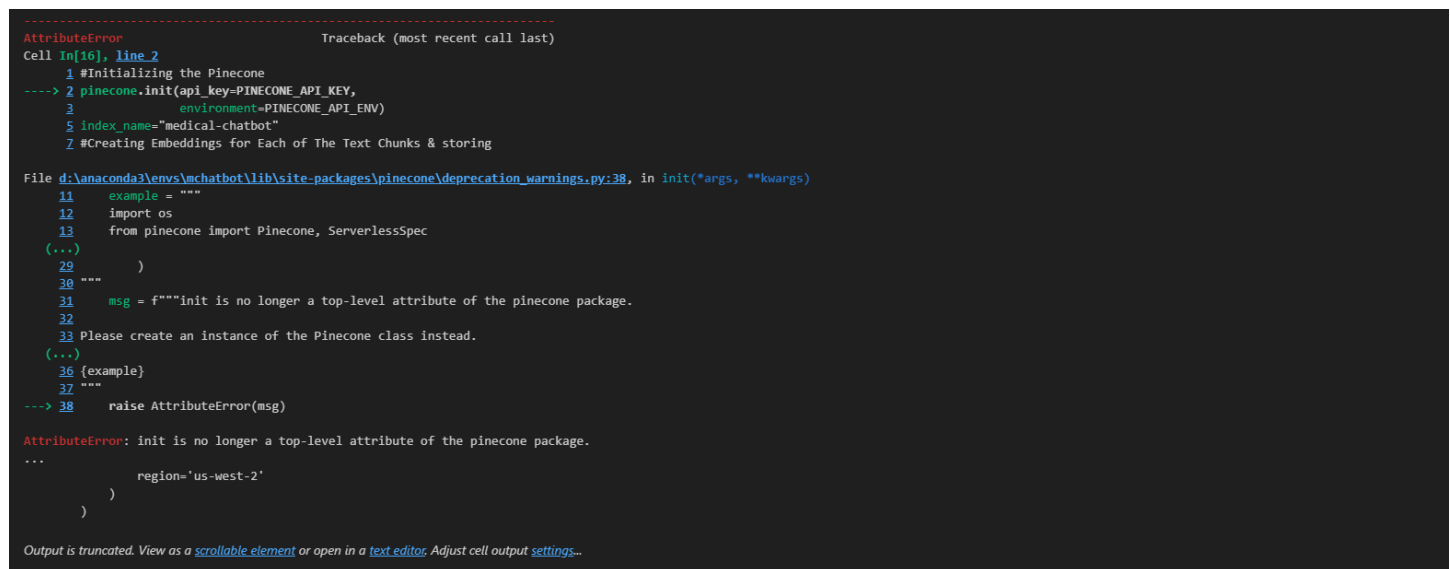


Figure 66 Showing the Depreciation Warning of Pinecone

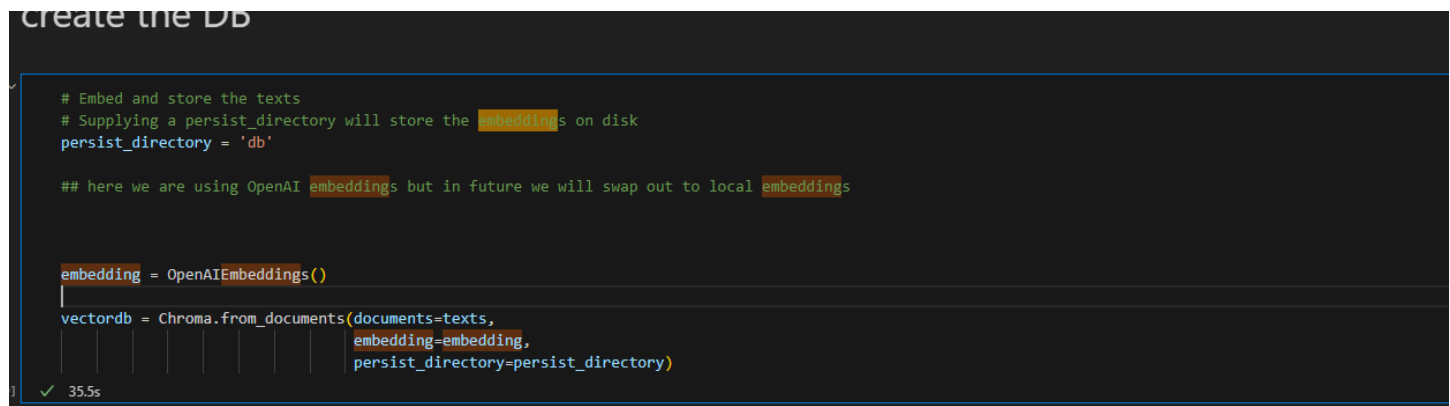


Figure 67 Showing the creation of Vector Database

Making a Retriever:

The retriever that was constructed from the database served as a way to query the database to get the specific piece of that that was required according to the query of the user.



```
retriever = vectordb.as_retriever()

docs = retriever.get_relevant_documents("what is household and tenant law?")
print(docs)

len(docs)

retriever = vectordb.as_retriever(search_kwargs={"k": 2})

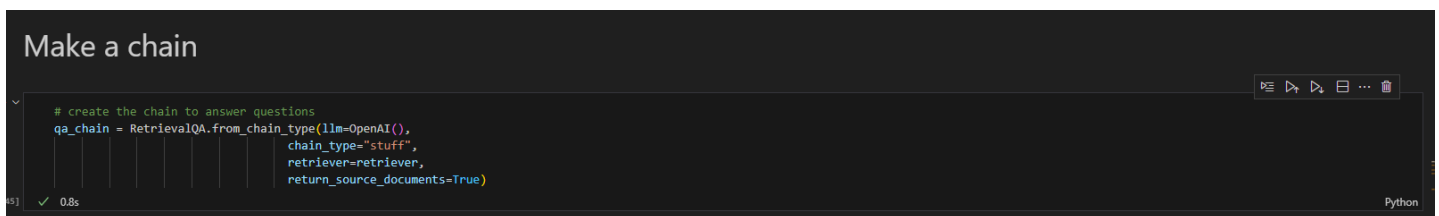
retriever.search_type
```

The screenshot shows a Jupyter Notebook interface with five code cells. The first cell defines a retriever. The second cell queries the retriever for documents related to 'household and tenant law' and prints them. The third cell checks the length of the retrieved documents, which is 4. The fourth cell updates the retriever to return the top 2 results. The fifth cell checks the search type of the retriever, which is 'similarity'.

Figure 68 Showing the Implementation of VectorDb Retriever

Making a Retrieval-based QA Chain for answering the question:

The second last step in the development of the model was setting up the retrieval-based QA chain, which was done with the help of LangChain library. It helped me process the user question and understand key terms or entities. And enables me to retrieve information based on the processed user input and give that as a context to the model. And retriever was passed as a parameter as well which served as a way to get the information from the vector database.



```
# create the chain to answer questions
qa_chain = RetrievalQA.from_chain_type(llm=OpenAI(),
                                     chain_type="stuff",
                                     retriever=retriever,
                                     return_source_documents=True)
```

The screenshot shows a Jupyter Notebook interface with a single code cell. The code creates a RetrievalQA chain using an OpenAI LLM, a 'stuff' chain type, and the retriever from the previous figure. It is configured to return source documents.

Figure 69 Showing the implementation of Retrieval-based QA Chain Using OpenAI() LLM

Generating Responses with Source Citation:


```
## Cite sources
def process_llm_response(llm_response):
    print(llm_response['result'])
    print('\n\nSources:')
    for source in llm_response['source_documents']:
        print(source.metadata['source'])

query = "Can you tell me about rent repayment plans"
llm_response = qa_chain(query)
process_llm_response(llm_response)
```

The Rent Repayment Orders (Supplementary Provisions) (England) Regulations 2007, along with the Residential Property Tribunal Procedure (England) Regulations 2006, outline the process for obtaining a re

Sources:
new_articles\Rent Repayment Order-Tenants Guide.pdf
new_articles\Rent Repayment Order-Tenants Guide.pdf

Figure 70 Showing the Response Generation with Source Citation

Prompt Engineering:

The final step of the model configuration was to set prompts for the Gen AI so that it doesn't answer questions out of the context or it doesn't answer question answer question which it doesn't know answer for

Making Sure the Generative AI doesn't Answer Question Out of Context:

```
prompt_template="""
Use the following pieces of information to answer the user's question.
If you don't know the answer, just say that you don't know, don't try to make up an answer.

Context: {context}
Question: {question}

Only return the helpful answer below and nothing else.
Helpful answer:
"""

PROMPT=PromptTemplate(template=prompt_template, input_variables=["context", "question"])
chain_type_kwargs={"prompt": PROMPT}
```

Figure 71 Setting up the context for the LLM

I played around the prompt settings to find the best prompt that would result in the best responses generated from the AI. Different versions of the prompts were created two of them are displayed in the Figure 71 and Figure 72.

```

# break it down
query = "What is the news about football match in uk?"
llm_response = qa_chain(query)
process_llm_response(llm_response)
llm_response
✓ 1.2s

```

I'm sorry, but I couldn't find any news about a football match in the UK. If you have any other questions related to UK household and tenant law, feel free to ask!

Sources:
new_articles\The role of private landlords in making a rented house a home.pdf
new_articles\No.123-Landlord-and-Tenant-Responsibility-for-State-and-Condition-of-Property.pdf

```

{'query': 'What is the news about football match in uk?',
 'result': "I'm sorry, but I couldn't find any news about a football match in the UK. If you have any other questions related to UK household and tenant law, feel free to",
 'source_documents': [Document(page_content='InTERnATIOnAL JOuRnAL OF HOuSIng POLICY 119\nIncluded studies\nunsurprisingly, the set of included studies is dominated by', Document(page_content='Defiies v. Milne [1913] 1 Ch. 98. \nHerne v. Bembow (1813) 4 Taunt. 764. \n12' 2 Roll. Abr. 816, pl. 36, 37. \nCheetham v. Hampson (1791) 4 Term R

```

Figure 72 Showing the LLM answer to a Question out of Context

Modifying the Prompt to get the Desired Outcome and Limiting the Gen AI:

```

prompt_template="""
Given the user's question, provide advice specifically tailored to UK household and tenant law. Make every effort to include relevant UK case law as reference to support your advice. If the question

Context: {context}
Question: {question}

If the question relates to UK household and tenant law:
- Offer advice that is compliant with UK legislation and regulatory frameworks.
- Strive to cite specific UK case law or legal statutes with every piece of advice, ensuring the information is accurate and up to date.
- Present the advice in a clear, concise manner that is accessible to individuals without a legal background.

If the question is about "Faran":
- Respond by saying, "Faran is my maker."

If the question is outside the domain of UK household and tenant law, or if it pertains to legal jurisdictions outside the UK:
- Politely inform the user that the question falls outside the service's expertise area or geographic focus and suggest consulting a relevant legal professional within the appropriate jurisdiction.

Helpful answer:
"""

```

Figure 73 Prompt Engineering Version 1

```

prompt_template="""
Litigat8, designed by Faran for his final year project, offers tailored advice on UK household and tenant law, using the most up-to-date information available up to its last training cut-off.

Context: {context}
Question: {question}

Guidelines for response by Litigat8:
1. **User-Friendly Language:**
| - Communicate legal advice in clear, straightforward language, making complex legal principles accessible to those without a legal background.

2. **Feedback Mechanism:**
| - Encourage feedback on the advice's usefulness and understandability, using this input to enhance Litigat8's future interactions.

3. **Privacy Considerations:**
| - Avoid unnecessary collection of personal information, ensuring users' privacy and confidentiality in providing generalized legal guidance.

4. **Update Mechanism:**
| - Regularly refresh Litigat8's knowledge with the latest case law, legislation, or changes in UK household and tenant law to maintain the accuracy of its advice.

5. **Limit Scope Wisely:**
| - Focus Litigat8's advice within the realm of UK household and tenant law, avoiding areas outside its expertise or current knowledge.

**Detailed Instructions for Litigat8:**
- **For UK household and tenant law queries:**
| - Utilize current UK legislation and case law to form detailed advice, addressing legal ambiguities and guiding users towards professional advice when necessary.

- **For questions about "Faran" or "Litigat8":**
| - For "Faran": Respond with, "Faran is my maker."
| - For "Litigat8": Share that it is an AI designed for providing legal advice on UK household and tenant law as a final year project by Faran.

- **For out-of-scope inquiries:**
| - Inform users when their questions fall outside Litigat8's domain or geographic focus, recommending professional legal consultation.

**Handling Ambiguities and Uncertainty:**
- Clearly explain when legal issues are ambiguous and suggest seeking further advice for complex matters.

**Communicating with Users:**
- If needed, ask for clarification to offer more precise advice, always respecting user privacy and focusing on enhancing understanding.

**Feedback and Continuous Improvement:**
- Include a feedback option for users to rate the helpfulness of advice and suggest improvements, aiding in the ongoing development of Litigat8.

Helpful answer:
"""

```

Figure 74 Prompt Engineering Version 2

Comparison of ChatGPT 3.5 Turbo Vs ChatGPT 4.0 Turbo Vs Llama 2:

The final step that was to be implemented was to do a comparative analysis between the LLM models that exists to finalize which would serve the function of litigate the best. The technical comparative analysis had two metrics which lead to the evaluation of the model the first one being the response time as one of the objectives were to display advices in real-time so it was of paramount the model should be able to generate responses according to that. The second metric was semantic similarity score of the response generate to figure out if the response generated has a meaning full context and semantic relation with the question that was asked by the user. In addition to that conversation coherence analysis was done was well which was carried out by myself to see how coherent the sentences are which model produces more meaningful and well structured sentences.

```
from langchain.llms import CTransformers

# Set up the turbo LLM
turbo_llm = ChatOpenAI(
    temperature=0,
    model_name='gpt-3.5-turbo'
)
gpt_turbo_llm = ChatOpenAI(
    temperature=0,
    model_name='gpt-4'
)

llm=CTransformers(model="llama-2-7b-chat.ggmlv3.q4_0.bin",
                  model_type="llama",
                  config={'max_new_tokens':4096,
                        'temperature':0.8,
                        'context_length' : 2048})
```

Figure showing the implementation of gpt-3.5-tubo , llama-2-7b Model , gpt-4.0

Technical Analysis :Semantic Analysis and Response Time Comparison:

The evaluation was carried out by setting up dummy questions and answer which were linked to specific prompts. And then the response generation time and the accuracy score (semantic score) were compared in the end for all three of the models.

```

import time
from sentence_transformers import SentenceTransformer, util

def semantic_similarity_score(response, correct_answer):
    model = SentenceTransformer('all-MiniLM-L6-v2')

    if not isinstance(response, list):
        response = [response]
    if not isinstance(correct_answer, list):
        correct_answer = [correct_answer]

    response_embedding = model.encode(response, convert_to_tensor=True)
    correct_answer_embedding = model.encode(correct_answer, convert_to_tensor=True)
    similarity = util.pytorch_cos_sim(response_embedding, correct_answer_embedding)

    return similarity.item()

def compare_llms(prompts, correct_answers, turbo_llm, llama_llm, gpt_turbo_llm):
    performance_metrics = {
        'turbo_llm': {'accuracy': [], 'response_time': []},
        'llama_llm': {'accuracy': [], 'response_time': []},
        'gpt_turbo_llm': {'accuracy': [], 'response_time': []}
    }

    turbo_llm = RetrievalQA.from_chain_type(llm=turbo_llm,
                                           chain_type="stuff",
                                           retriever=retriever,
                                           return_source_documents=True)

    llama_llm = RetrievalQA.from_chain_type(llm=llama_llm,
                                           chain_type="stuff",
                                           retriever=retriever,
                                           return_source_documents=True)

    gpt_turbo_llm = RetrievalQA.from_chain_type(llm=gpt_turbo_llm,
                                           chain_type="stuff",
                                           retriever=retriever,
                                           return_source_documents=True)

    # Loop over each prompt and get responses from both LLMs

```

Figure 75 Setting Up the Semantic Similarity func and Comparison func

```

# Example usage:
prompts = ["What are rent repayment Plans?", "Can you give me some case laws where the tenant got evicted for not paying rent?"]
correct_answers = [
    "What are rent repayment Plans?": "Rent repayments refer to the process where a tenant may be entitled to reclaim rent from their landlord in certain circumstances, such as when the property is in a state of disrepair or the landlord has failed to maintain the property.",
    "Can you give me some case laws where the tenant got evicted for not paying rent?": "In the UK, there are several cases where tenants have been evicted for not paying rent. One notable case is Smith v. Griggs, where a tenant was evicted for not paying rent for several months."
]

comparison_results = compare_llms(prompts, correct_answers, turbo_llm, llama_llm, gpt_turbo_llm)

```

Figure 76 Setting up the Prompts and Correct Answers

```

{'turbo_llm': {'accuracy': [1, 1],
               'response_time': [1.692568302154541, 1.3909106254577637],
               'average_response_time': 1.5417394638061523,
               'average_accuracy': 1.0},
 'llama_llm': {'accuracy': [1, 1],
               'response_time': [13.296113967895508, 75.26336598396301],
               'average_response_time': 44.27973997592926,
               'average_accuracy': 1.0},
 'gpt_turbo_llm': {'accuracy': [1, 1],
                   'response_time': [12.599728345870972, 8.483205080032349],
                   'average_response_time': 10.54146671295166,
                   'average_accuracy': 1.0}}

```

Figure 77 Showing the Comparison Result

Conversational Coherence Analysis

After the technical analysis, the models were compared on the bases of conversational coherence with a simple metric which response was more appealing to me as a user who's seeking advice on a matter. For this purpose, a model question was decided on and was asked all three of the models and the responses were evaluated which would be discussed in the next chapter.

Evaluation Sentence: “What are rent repayments how can I pay them can you tell me?”

LLAMA 2 Response time 2Min >

Response : To reclaim rent paid during a period when you were not legally required to pay it, you will need to apply for a Rent Repayment Order (RRO). To do this, you need to provide evidence of the rent paid during the period. To pay rent repayments, you can use a payment method accepted by the landlord or managing agent, such as bank transfer or cheque. If you have paid in cash and do not have receipts, you will need to provide evidence of the rent paid. If you are unsure about how to claim rent repayments or have any questions, please let me know, and I will do my best to assist you.

Figure 78 Showing the response generated LLMA-2

Q 4m 30.2s Python

Response : Thank you for reaching out to Litigat8! We're here to help with any questions or concerns related to UK household and tenant law. Please feel free to ask us anything, and we'll provide you with the best possible advice. Please remember that Litigat8 is a machine learning model designed as a final year project by Faran, and while it strives to provide accurate advice, it may not always be perfect or up-to-date. If you have any further questions or need assistance with the process of reclaiming rent payments, feel free to ask for more specific guidance.

Response : "Dear user, thank you for reaching out to Litigat8 regarding rent repayments. Based on the information provided, it appears that you are seeking guidance on how to claim rent repayments from your landlord. As an AI language model trained on UK household and tenant law, I can advise you on the most up-to-date legal principles and procedures for making a claim. However, please note that Litigat8's response is not a substitute for legal advice. To start, it would be helpful to have evidence of the rent paid during the period of up to twelve months. If you do not have a rent book or receipts for the rent, please make a list of all the rent payments made, including the date, amount, and to whom it was paid. Once you have gathered the necessary information and evidence, you can submit your claim to your local council for housing benefits. They will review your application and provide guidance on the next steps. Please feel free to ask if you have any further questions or concerns regarding rent repayments or UK household and tenant law. Your feedback is invaluable in improving Litigat8's services."

Figure 79 Showing the response generated LLMA-2

GPT4 1Min >

Response : Rent repayments refer to situations where a tenant may be entitled to reclaim rent payments from their landlord. This could happen if the landlord has not met their legal obligations, such as not paying council tax or not providing a valid tenancy agreement. To pay rent repayments, you would typically need to apply for a Rent Repayment Order (RRO) through the First-tier Tribunal. You would need to provide evidence of the rent payments made, such as a rent book or receipts. It's important to note that the period for reclaiming rent may end earlier under certain circumstances, such as if you have moved away, if the landlord has changed, or if the landlord has applied for a possession order. If you have further questions or need assistance with the process of reclaiming rent payments, feel free to ask for more specific guidance.

Figure 80 Showing the response Generated by gpt-4

GPT 3.5 Turbo : 10 sec>:

Response : Rent repayments refer to the process of reclaiming overpaid rent from your landlord. To pay them, you will need to apply for a Rent Repayment Order (RRO) and provide evidence of the rent you have paid.

Figure 81 showing the response generated by gpt-3.

Outcome Discussion

Based on the results that were generated for the models **GPT-4**, **GPT-3.5-Turbo** and **LLAMA2**. When it comes to accuracy all three models performed as expected with having semantic relationship between the question asked and the response generated. Hence, they all passed on this evaluation. Moving on to the next evaluation, response time **GPT 3.5-turbo** was the fastest with response time ranging between 8 to 10 sec followed by **GPT-4** and then **LLAMA 2**. Hence **GPT 3.5-turbo** was the clear winner in the technical evaluation. Moving on to the Conversational Coherence Analysis **LLAMA2** and **GPT-4** provided with the best responses in the context of the data that was provided to them but **GPT-3.5-turbo** responses were accurate as well covering all the essential points to form a good response. As a result of all these observations **GPT-3.5-turbo** was selected as its steady true for all the metrics that were required for Litigat8 to functions.

This marks the end of the implementation the next chapter would discuss the testing and evaluation of the application that was carried out. And how the identified bugs and problems were rectified.

Testing:

The testing of the application was carried out in accordance to the primary objectives and use cases set out by the project. Thus, the tests would focus on the parts of the program that are critical for the success of the application.

Unit Testing

This chapter discusses the implementation and outcome of the unit tests that were designed for the application. The Unit tests TC-001, TC-002, TC-003, TC-005, TC-010 and TC-011 were designed in accordance to the development of the User interphase and testing its robustness these tests mainly focused on use case UC4 testability. Whereas, the unit tests TC-006, TC-007 and TC-003 were also part of the primary objective but focused on UC1, UC2 and UC6, and thus checked their robustness. Moving on the unit test TC-009 focused on Primary objective P2 and P4 and thus tested the UC4. The P3 objective didn't need testing as it was one of task which didn't need to be repeated again and again thus making it function one time was enough. And would not let to the failure of the system if anything goes wrong.

Designing the Unit Tests Using Pytest:

Setting up the Test Environment and setting up the database before the Tests::

```

@pytest.fixture(scope='module')
def test_client():
    flask_app = create_app()
    with flask_app.test_client() as testing_client:
        with flask_app.app_context():
            yield testing_client

@pytest.fixture(scope='module')
def init_database(test_client):

    _db.create_all()

    user1 = User(id=1, username='TestUser', password='Password')
    _db.session.add(user1)
    _db.session.commit()

    yield # No need to return _db

    _db.session.remove()
    _db.drop_all()

```

Pytest for Ending Chat Session:

```

def test_end_chat_session(test_client, init_database):
    login_user(test_client)
    with test_client.session_transaction() as session:
        session['chat_session_id'] = 1

    response = test_client.get('/chat/end_chat_session', follow_redirects=True)
    assert response.status_code == 200

```

Pytest for form submission and AI Response:

```

def test_submit(test_client, init_database):
    login_user(test_client)
    user_input = "Hello, AI!"
    response = test_client.post('/chat/submit', data=json.dumps({"user_input": user_input}), content_type='application/json')
    data = response.get_json()
    assert response.status_code == 200
    assert data['user_input'] == user_input

```

Pytest Authentication/ Login:

```

def login_user(test_client, username=None, password="testpassword"):
    if username is None:
        username = f"user_{uuid4()}" # Generate a unique username
    user = User(username=username, password=password)
    db.session.add(user)
    db.session.commit()
    with test_client.session_transaction() as session:
        session['user_id'] = user.id # Mock login
def test_submit(test_client, init_database):

```

Pytest Authentication/ Login Failure:

```
def test_login_failure(test_client, init_database):  
    response = test_client.post('/auth/login', data={  
        'username': 'wronguser',  
        'password': 'wrongpassword'  
    }, follow_redirects=True)  
  
    # Assertions about the response  
    assert response.status_code == 200  
    assert 'Invalid username or password.' in response.data.decode('utf-8')
```

Pytest Authentication/ Login Success:

```
def test_login_success(test_client, init_database):  
    # First, create a user  
    hashed_password = generate_password_hash('mysecurepassword')  
    user = User(username='existinguser', password=hashed_password)  
    _db.session.add(user)  
    _db.session.commit()  
  
    # Attempt to login  
    response = test_client.post('/auth/login', data={  
        'username': 'existinguser',  
        'password': 'mysecurepassword'  
    }, follow_redirects=True)  
    assert response.status_code == 200  
    assert 'user_id' in session
```

Pytest Authentication/ Register:

```
def test_register(test_client, init_database):  
    # Mimic form submission with POST data for registration  
    response = test_client.post('/auth/register', data={  
        'username': 'testuser2',  
        'password': 'testpassword'  
    }, follow_redirects=True)  
    assert response.status_code == 200  
    assert User.query.filter_by(username='testuser').first() is not None
```

Pytest Saving Interaction of Users and ChatSession:


```

# Make sure this test also uses the init_database fixture
def test_save_interaction_authenticated(test_client, init_database):
    # Ensuring a chat session exists in the database for the user
    chat_session = ChatSession(user_id=1, start_time=datetime.utcnow())
    _db.session.add(chat_session)
    _db.session.commit()

    with test_client.session_transaction() as session:
        session['user_id'] = 1
        session['chat_session_id'] = chat_session.id

    data = {'user_input': 'Hello', 'ai_response': 'Hi there!'}
    response = test_client.post('/chat/save_interaction', json=data)
    assert response.status_code == 200
    assert b'success' in response.data

```

Pytest to get chat sessions:

```

def test_get_chat_sessions_unauthenticated(test_client):
    response = test_client.get('/chat/get_chat_sessions')
    assert response.status_code == 200

```

```

def test_get_chat_sessions_authenticated(test_client, init_database):
    with test_client.session_transaction() as session:
        session['user_id'] = 1

    response = test_client.get('/chat/get_chat_sessions')
    assert response.status_code == 200

```

Pytest to start chat session:

```

def test_start_chat_session_unauthenticated(test_client):
    with test_client.session_transaction() as sess:
        sess.pop('user_id', None)

    response = test_client.post('/chat/start_chat_session')
    assert response.status_code == 302 # Or 401, depending on your auth handling

# Adjusted to use the init_database fixture for database setup
def test_start_chat_session_authenticated(test_client, init_database):
    with test_client.session_transaction() as sess:
        sess['user_id'] = 1

    response = test_client.post('/chat/start_chat_session')
    assert response.status_code == 201

```

Pytest get chat histories:

```

def test_fetch_chat_histories(test_client, init_database):
    login_user(test_client, "user_for_history", "password")
    user = User.query.filter_by(username="user_for_history").first()

    # Create mock chat sessions and messages
    chat_session = ChatSession(user_id=user.id)
    db.session.add(chat_session)
    db.session.commit()

    message1 = Conversation(chat_session_id=chat_session.id, message="Hi", type='user')
    message2 = Conversation(chat_session_id=chat_session.id, message="Hello", type='ai')
    db.session.add_all([message1, message2])
    db.session.commit()

    response = test_client.get('/chat/fetch_chat_histories')
    data = response.get_json()
    assert response.status_code == 200
    assert len(data) > 0 # Assuming at least one history exists
    # Further asserts can check the structure of the returned history data

```

Unit Tests Outcome:

This chapter concludes the unit tests that were conducted for the Litigat8. The test results were all satisfactory with almost 80% complete success rate. With 2 test cases that resulted that resulted in warnings but they carried out the function they were supposed to. The two tests that resulted in warnings would be discussed in the next chapter.

ID	Component	Test Case	Input	Expected Output	Success Criteria	Outcome
TC-001	Start Chat Session (Authenticated)	User is logged in and starts a chat session	User session with user_id set	HTTP status code 201, indicating creation of a new chat session	Test passes if the response status code is 201	Pass
TC-002	Start Chat Session (Unauthenticated)	User is not logged in	No user_id in session	HTTP redirect (302) or 401 Unauthorized, depending on auth handling	Test passes if the response is a redirect (302) or 401	Pass
TC-003	Get Chat Sessions (Authenticated)	User is logged in and fetches chat sessions	User session with user_id set	HTTP status code 200 and a list of chat sessions associated with the user	Test passes if the status code is 200 and the response contains the user's chat sessions	Pass
TC-004	Get Chat Sessions (Unauthenticated)	User is not logged in	No specific input needed	HTTP status code 200,	Test passes if the status code is 200	Pass
TC-005	Save Interaction (Authenticated)	User is logged in and saves an interaction	User session with user_id and chat_session_id set JSON payload with user_input and ai_response	HTTP status code 200 with a success message	Test passes if the interaction is successfully saved and the response contains 'success'	Pass
TC-006	Register User	New user registration	POST data with username and password	HTTP status code 200 and the new user is found in the database	Test passes if a new user is created and can be found in the database	Pass

ID	Component	Test Case	Input	Expected Output	Success Criteria	Outcome
TC-007	Login (Success)	Existing user logs in successfully	POST data with valid username and password	HTTP status code 200 and session contains user_id, indicating successful login	Test passes if the user is logged in and session contains user_id	Pass
TC-008	Login (Failure)	User fails to log in	POST data with invalid username and password	HTTP status code 200 with an 'Invalid username or password.' message	Test passes if the response contains the invalid login message	Pass
TC-009	Submit (Authenticated)	User submits input in a chat	JSON payload with user_input, User session contains user_id	HTTP status code 200 and the response JSON contains the same user_input	Test passes if the response status is 200 and echoes user_input	Pass
TC-010	End Chat Session (Authenticated)	User ends a chat session	User session with chat_session_id set	HTTP status code 200, chat session was successfully ended	Test passes if the chat session ends successfully	Pass
TC-011	Fetch Chat Histories (Authenticated)	User fetches their chat histories	User session contains user_id	HTTP status code 200 and a JSON payload containing the user's chat histories	Test passes if the response contains the user's chat histories and status code is 200	Pass

report.html

Report generated on 31-Mar-2024 at 16:09:09 by [pytest-html](#) v4.1.1

Environment

Python	3.11.8
Platform	Windows-10-10.0.22631-SP0
Packages	<ul style="list-style-type: none">• pytest: 8.1.1• pluggy: 1.4.0
Plugins	<ul style="list-style-type: none">• anyio: 4.2.0• html: 4.1.1• metadata: 3.1.1

Summary

11 tests took 00:00:05.

(Un)check the boxes to filter the results.

☐ 0 Failed, ☒ 11 Passed, ☐ 0 Skipped, ☐ 0 Expected failures, ☐ 0 Unexpected passes, ☐ 0 Errors, ☐ 0 Reruns

[Show all details](#) / [Hide all details](#)

Result	Test	Duration	Links
Passed	app/tests/system_test.py::test_start_chat_session_authenticated	156 ms	
Passed	app/tests/system_test.py::test_start_chat_session_unauthenticated	2 ms	
Passed	app/tests/system_test.py::test_get_chat_sessions_authenticated	3 ms	
Passed	app/tests/system_test.py::test_get_chat_sessions_unauthenticated	1 ms	
Passed	app/tests/system_test.py::test_save_interaction_authenticated	79 ms	
Passed	app/tests/system_test.py::test_register	77 ms	
Passed	app/tests/system_test.py::test_login_success	214 ms	
Passed	app/tests/system_test.py::test_login_failure	2 ms	
Passed	app/tests/system_test.py::test_submit	00:00:04	
Passed	app/tests/system_test.py::test_end_chat_session	5 ms	
Passed	app/tests/system_test.py::test_fetch_chat_histories	43 ms	

Figure 82 Showing the results of pytest for Unit Tests

Passed	app/tests/system_test.py::test_get_chat_sessions_authenticated	3 ms	
Passed	app/tests/system_test.py::test_get_chat_sessions_unauthenticated	1 ms	
Passed	app/tests/system_test.py::test_save_interaction_authenticated	79 ms	
Passed	app/tests/system_test.py::test_register	77 ms	
Passed	app/tests/system_test.py::test_login_success	214 ms	
Passed	app/tests/system_test.py::test_login_failure	2 ms	
Passed	app/tests/system_test.py::test_submit	00:00:04	

----- Captured stdout call -----

this works
Hello, AI!
Hello! How can I assist you today with UK household and tenant law? Feel free to ask your question, and I'll do my best to provide you with clear and helpful advice. Remember, your privacy is important, and I'm here to offer guidance within the scope of UK household and tenant law.

expand [-]

Figure 83 Showing the Unit Test for Interaction with Litigat8 NLP Model

```
===== warnings summary =====
app/tests/system_test.py::test_login_success
  D:\WorkOpus\Litigate\Litigate\app\__init__.py:62: LegacyAPIWarning: The Query.get() method is considered legacy as of
the 1.x series of SQLAlchemy and becomes a legacy construct in 2.0. The method is now available as Session.get() (deprec
ated since: 2.0) (Background on SQLAlchemy 2.0 at: https://sqlalche.me/e/b8d9)
    user = User.query.get(user_id)

app/tests/system_test.py::test_submit
  D:\anaconda3\envs\Litigat8\Lib\site-packages\langchain_core\api\deprecation.py:117: LangChainDeprecationWarning: The
function '__call__' was deprecated in LangChain 0.1.0 and will be removed in 0.2.0. Use invoke instead.
    warn_deprecated(

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
===== 11 passed, 2 warnings in 7.12s =====

(Litigat8) D:\WorkOpus\Litigate\Litigate>
```

Figure 84 Showing Warnings Generated by the Tests

Unit Tests Outcome Discussion:

Component Login(Success) :

func test_login_success: The warning was originated from SQLAlchemy. Which warned me about the Query.get() method has depreciated in the version 2.0 of the library. And instead advised me to use Session.get() instead to resolve the warning and any other compatibility issues in the future.

As a result, the login functions was updated to use the newer version of the function thus eliminating the warning for the test_login_success function

```
@auth.route('/login', methods=['GET', 'POST'])
def login():
    user_id = session.get('user_id')
    if user_id:
        return redirect(url_for('main.chat'))
```

Figure 85 Showing the Fix of Login (Success) component warning

Component Submit (Authenticated):

func test_submit:

The warning was originated from langchain_core_api warning about the deprecation of `__call__` function within the version LangChain 0.1.0 and warning that it would be removed in 0.2.0. I wasn't able to exactly pinpoint which method would be required to change as the error is not being originated from. But for the sake of this project and the version of LangChain that is being used for the project. It wouldn't effect the functioning of the project.

Integration Testing Plan:

Integration testing was carried out with the development of each sub system of the project to make sure that addition of a new functionality doesn't compromise the stability of the system and doesn't break it. The results of the integration testing was mostly positive as a result of through unit testing which reduced the number of problems to be face in the integration testing.

Integration Testing Outcomes:

Integration Components	Test Case	Input	Expected Output	Success Criteria	Outcome
Frontend <-> Backend API Calls	Test login API call	User login credentials	User authenticated / Session token	User can log in and start a session	Pass
	Test query submission API call	Valid query	Query accepted / Advice returned	Query processed and advice returned	Pass
Backend <-> Database CRUD Operations	Test user creation in database through backend	New user details	User created in Users table	New user can log in	Pass
	Test query processing and storage	Sample query	Query stored in UserQueries & NLPAnalysis	Query results and analysis are stored	Pass
NLP Engine <-> Database	Test storing NLP analysis results	Sample query	Results stored in NLPAnalysis	Analysis results retrievable for the query	Pass
Authentication <-> Database	Test authentication with database-stored credentials	User credentials	Authentication success/failure	Correct authentication against database data	Pass
Chat Interface <-> Chat Handling <-> Database	Test chat message flow	Chat message	Message stored in ChatHistories	Chat history updated with new message	Pass

Evaluation:

There were two sets of objectives—primary and secondary—that defined the scope of the project. Primary objectives were of utmost importance for the success of the project, followed by secondary objectives. Following an agile solution approach with SCRUM, outputs were produced during each sprint, serving as building blocks for the application, thus resulting in a fully functioning application. The resulting application demonstrates a wide variety of functional features, which, combined with the functional and non-functional system requirements analyzed by the MoSCoW Method, address the set four primary objectives. However, in the case of secondary objectives, two failed to be accomplished.

The P1 objective, which stated the development of a User Interface that is intuitive and accessible, was accomplished, with the exception that a proper feedback system, UC5, was not implemented in this phase of development. The objective was accomplished by following a proper agile solution approach with an emphasis on time management, where tools like the Trello board and Gantt Chart were used. The objective was approached systematically, where the initial design was constructed using wireframes, which were then implemented in code to avoid any changes in design at a later stage. Following P1, the P2 and P3 objectives were put under development. The focus of P1 was on developing a natural language processing system unit that could understand user queries and process them accurately in real-time. The accomplishment of this objective was challenging, as I had to work with many new technologies. However, thanks to proper time management and a sound solution approach, I was able to tackle it appropriately, thus producing a proof-of-concept NLP model capable of performing the set tasks using libraries such as LangChain, CTransformers, OpenAI(), etc. The P3 objective was carried out in parallel, where many articles concerning household and tenant law were collected and integrated into the NLP model by creating vector databases. Most of the articles collected were in PDF format, thus the processing of this information was facilitated by libraries such as PyPDF Loader. Both of these objectives were completed systematically following the proper plan and designs set in place.

Moving on to the final primary objective, P4, real-time response generation technologies like JS Fetch Requests and Flask Server Handling were used to initiate requests, gather responses, and display them on the front-end. This enabled users to receive advice from the AI in real-time. Comparative studies were also conducted to find the best model and parameters for quick advice, which resulted in the best configuration of the model, thus enabling the accomplishment of this objective.

When it comes to secondary objectives, S1 was achieved partly due to most resources and time being allocated to achieving the primary objectives. Steps were taken to make the application dynamic so that it could be supported on multiple devices, increasing user access. The objective that wasn't accomplished was S4, to establish a feedback system for the model in such a way that it would remember the conversation with the user and use that as context for the conversation,

which will still be part of the future development of the project. Objectives S2 and S3 were accomplished by setting up an authentication model with ample security measures, such as locking the URLs and using Bcrypt to encrypt all passwords for users, signifying the successful completion of these objectives.

Using the Scrum project solution approach, I was able to tackle all the problems I faced in a timely manner. Putting in place a proper time management system ensured that I was able to accomplish my objectives on time, whether they were primary or secondary. Through requirement analysis and planning, I was able to prioritize all objectives that were of prime importance for the success of the project. Using these techniques, I was able to complete the project and produce a proof-of-concept application that met the objectives set out for it, which I am really proud of. These techniques ensured that if any problems arose, there would be a failsafe in place

Future Work:

There are number of areas that could be improved in the scope of this project. Which I believe can be implemented to make the system more robust and have a lot more features than it currently has.

The first improvement that would be implemented in the future for this application would be to establish a **feedback system** along with chat remembering feature for the AI so that the user could have a conversation with the model and gradually improve the output the model results in. And the feedback would make the model learn and gradually improve its performance.

The second improvement that would be implemented will be an **online vector database** to increase its accessibility across devices. One of the implementations of it was explored but was failed to be implemented for the current application. So, I would like to explore alternatives such as Weaviate to implement it. Thus adding more versatility to the project.

The third improvement would be to explore models like **Grok** to implement the functionality as this model can support upwards of 314 billion parameters with context token size of 8K tokens. Which is great improvement from gpt-3.5-turbo model but still relatively less effective than GPT-4 and I want to test if this can serve as a better option relative to gpt-3.5-turbo model in response time.

The fourth improvement I would like to make it is to increase the information available in the dataset and focus on **other domains of law** as well which I feel like would provide more utility instead of just focusing on a single domain but given the restraint of time and resources for this proof-of-concept application it couldn't be implemented in this development phase but if this application is decided to work on this improvement would be implemented.

The fifth improvement would be to make the advice that generated out of the model should be more structured with headings and sub-headings which I believe would be done with the help of **markdown language**. This would make the advice look more readable and more understandable for the user.

The sixth area of development would be supporting **file uploads to proof-read-contracts** or even draw up contracts for you like tenancy agreement and such. Which I believe would provide a lot of utility to the users and hence would greatly increase user satisfaction. And I feel like this area would make Litigat8 stand out and provide real utility to the users.

These are some of the key areas of improvements that could be worked on to improve the litigat8's application and functionality. If I decide to work on the application beyond proof-of-concept then all of these improvement areas would be analysed and worked on and would be implemented in the real-world application.

Conclusion:

References

1. Khurana D, Koli A, Khatter K, Singh S. Natural language processing: state of the art, current trends and challenges. *Multimed Tools Appl.* 2023;82(3):3713-3744. doi: 10.1007/s11042-022-13428-4. Epub 2022 Jul 14. PMID: 35855771; PMCID: PMC9281254.(for Image and the introduction)
2. Sutskever, I., Vinyals, O., and Le, Q.V. (2014) 'Sequence to Sequence Learning with Neural Networks', in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2 (NIPS'14)*, MIT Press, Cambridge, MA, USA, pp. 3104–3112.
3. and Short Papers), pp. 4171–4186.

4. Hutchins, W.J. (1986) 'Machine Translation: Past, Present, Future', Ellis Horwood, Chichester, England.
5. Hutchins, W.J. (1995) 'Machine Translation: A Brief History', in Concise History of the Language Sciences: From the Sumerians to the Cognitivists, pp. 431–445.
6. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019) 'Language Models are Unsupervised Multitask Learners', OpenAI Blog.
7. Winograd, T. (1972) 'Understanding Natural Language', Academic Press, New York, NY.
8. Hassler, M., & Fliedl, G. (2006). Text Preparation Through Extended Tokenization.(for tokensation images)
9. <https://www.freecodecamp.org/news/an-introduction-to-part-of-speech-tagging-and-the-hidden-markov-model-953d45338f24/> (link for the pos image)
10. Goodfellow, I., Bengio, Y., and Courville, A. (2016) 'Deep Learning', MIT Press, Cambridge, MA, USA.
11. Vaswani, A. et al. (2017) 'Attention is All You Need', in Proceedings of the 31st International Conference on Neural Information Processing Systems, pp. 5998–6008.
12. <https://www.codemotion.com/magazine/video/nlp-techniques-and-deep-learning/> (image for machine learning in NLP)
13. [https://sonix.ai/articles/difference-between-artificial-intelligence-machine-learning-and-natural-language-processing\(image](https://sonix.ai/articles/difference-between-artificial-intelligence-machine-learning-and-natural-language-processing(image) for machine learning in NLP
14. Barocas, S., Hardt, M., and Narayanan, A. (2019) 'Fairness and Abstraction in Sociotechnical Systems', in Proceedings of the Conference on Fairness, Accountability, and Transparency, pp. 59–68.
15. Romanosky, S. (2016) 'Examining the Costs and Causes of Cyber Incidents', Journal of Cybersecurity, 2(2), pp. 121–135.
16. Voigt, P., and Von dem Bussche, A. (2017) 'The EU General Data Protection Regulation (GDPR)', Springer International Publishing, Cham.
17. Ashley, K.D. (2017) 'Artificial Intelligence and Legal Analytics: New Tools for Law Practice in the Digital Age', Cambridge University Press, Cambridge.
18. Branting, L.K. (2021) 'Explainable Artificial Intelligence and the Interpretability Problem in Legal Analytics', Jurimetrics, 61, pp. 45–68.
19. Kreutzer, R.T., and Sirrenberg, M. (2019) 'Understanding Artificial Intelligence: A Guide for Business Leaders', Springer Nature, Cham.
20. Tsarapatsanis, D., and Aletras, N. (2021) 'Machine Learning in Law: Present and Future', in Artificial Intelligence and Law, 29, pp. 355–385.
21. Zhong, H., Guo, Z., Tu, C., Xiao, C., Liu, Z., and Sun, M. (2020) 'How Does NLP Benefit Legal System: A Summary of Legal Artificial Intelligence', in Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pp. 5218–5230.

22. Alarie, B., Niblett, A., and Yoon, A.H. (2018) 'How Artificial Intelligence Will Affect the Practice of Law', *University of Toronto Law Journal*, 68(S1), pp. 106–124.
23. Katz, D.M., Bommarito II, M.J., and Blackman, J. (2017) 'A General Approach for Predicting the Behavior of the Supreme Court of the United States', *PLoS One*, 12(4), e0174698.
24. Rule, C. (2017) 'Online Dispute Resolution for All: The History and Future of Online Dispute Resolution', in *Online Dispute Resolution: Theory and Practice*, Eleven International Publishing, The Hague, pp. 13–32.
25. Sourdin, T. (2018) 'Judge v. Robot? Artificial Intelligence and Judicial Decision-Making', *University of New South Wales Law Journal*, 41(4), pp. 1114–1133.
26. <https://www.itv.com/news/2024-02-08/number-of-no-fault-evictions-hit-eight-year-high-new-data-shows>
27. <https://www.insidehousing.co.uk/news/large-councils-disrepair-cases-up-1584-in-five-years-83281>
28. <https://www.nimblefins.co.uk/number-homeowners-and-renters-uk#:~:text=Number%20of%20Renter%20UK,private%20renters%20than%20social%20renters.>
29. 5, No. 1, 2015, Available at SSRN: <https://ssrn.com/abstract=2565733>
2. Andy Dickerson, Emily McDool & Damon Morris. (2023) [Post-compulsory education pathways and labour market outcomes](#). *Education Economics* 31:3, pages 326-352.
3. Peta Wolifson, Sophia Maalsen & Dallas Rogers. (2023) [Intersectionalizing Housing Discrimination Under Rentier Capitalism in an Asset-Based Society](#). *Housing, Theory and Society* 40:3, pages 335-355.
4. Moore, T. and Dunning, R., 2017. Regulation of the private rented sector in England using lessons from Ireland. JRF Report, Joseph Rowntree Foundation, York
5. Flint, J., 2004. The responsible tenant: Housing governance and the politics of behaviour. *Housing studies*, 19(6), pp.893-909.
6. Ball, M., 2010. The UK private rented sector as a source of affordable accommodation. York: Joseph Rowntree Foundation.
7. Power, E. R. and Gillon, C. (2022) 'Performing the 'good tenant'', *Housing Studies*, 37(3), pp. 459–482. doi: 10.1080/02673037.2020.1813260.
8. McKee, K., Muir, J. and Moore, T. (2017) 'Housing policy in the UK: the importance of spatial nuance', *Housing Studies*, 32(1), pp. 60–72. doi: 10.1080/02673037.2016.1181722.
9. Hilber, Christian A. L. (2015) UK housing and planning policies: the evidence from economic research.

Dataset Document

Reference:

1. Carr, Helen and Cowan, Dave, The Social Tenant, the Law and the UK's Politics of Austerity (February 16, 2015). *Oñati Socio-Legal Series*, Vol.

- Election Analysis (33). Centre for Economic Performance, The London School of Economics and Political Science, London, UK.
10. Rolfe, S. et al. (2023) 'The role of private landlords in making a rented house a home', *International Journal of Housing Policy*, 23(1), pp. 113–137. doi: 10.1080/19491247.2021.2019882.
 11. S. Moffatt, S. Lawson, R. Patterson, E. Holding, A. Dennison, S. Sowden, J. Brown, A qualitative study of the impact of the UK 'bedroom tax', *Journal of Public Health*, Volume 38, Issue 2, June 2016, Pages 197–205, <https://doi.org/10.1093/pubmed/fdv031>
 12. Hulse, K. and Haffner, M. (2014) 'Security and Rental Housing: New Perspectives', *Housing Studies*, 29(5), pp. 573–578. doi: 10.1080/02673037.2014.921418.
 13. Murie, A. (1997) 'The social rented sector, housing and the welfare state in the UK', *Housing Studies*, 12(4), pp. 437–461. doi: 10.1080/02673039708720909.
 14. Stone, M. E. (2006) 'A Housing Affordability Standard for the UK', *Housing Studies*, 21(4), pp. 453–476. doi: 10.1080/02673030600708886.
 15. Harris, J. and McKee, K., 2021. Health and Wellbeing in the UK Private Rented Sector: enhancing capabilities. Glasgow, UK: UK Collaborative Centre for Housing Evidence (CaCHE).
 16. Harris, J. and McKee, K., 2021. Health and Wellbeing in the UK Private Rented Sector: enhancing capabilities. Glasgow, UK: UK Collaborative Centre for Housing Evidence (CaCHE).
 17. Kemp, P. A. (2011) 'Low-income Tenants in the Private Rental Housing Market', *Housing Studies*, 26(7–8), pp. 1019–1034. doi: 10.1080/02673037.2011.615155.
 18. Jordan M. Contesting the property paradigm amid 'radical' constitutional change: Living Rent and the Private Residential Tenancies (Scotland) Act 2016. *Legal Studies*. Published online 2024:1-18. doi:10.1017/lst.2024.4
 19. Lister, D. (2006). Unlawful or just awful? Young people's experiences of living in the private rented sector in England. *YOUNG*, 14(2), 141-155. <https://doi.org/10.1177/1103308806062738>
 20. Powell, R. (2015) 'Housing Benefit Reform and the Private Rented Sector in the UK: On the Deleterious Effects of Short-term, Ideological "Knowledge"', *Housing, Theory and Society*, 32(3), pp. 320–345. doi: 10.1080/14036096.2015.1027830.
 21. Green, G., Barratt, C. and Wiltshire, M. (2016) 'Control and care: landlords and the governance of vulnerable tenants in houses in multiple occupation', *Housing Studies*, 31(3), pp. 269–286. doi: 10.1080/02673037.2015.1080818.
 22. Badarinza, Cristian and Ramadorai, Tarun, Long-Run Discounting: Evidence from the UK Leasehold

- Valuation Tribunal (October 26, 2015). Available at SSRN: <https://ssrn.com/abstract=2412296> or <http://dx.doi.org/10.2139/ssrn.2412296>
23. Feijten, P., & van Ham, M. (2010). The Impact of Splitting Up and Divorce on Housing Careers in the UK. *Housing Studies*, 25(4), 483–507. <https://doi.org/10.1080/02673031003711477>.
 24. Morgan, J. (2001) 'Nuisance and the Unruly Tenant', *The Cambridge Law Journal*, 60(2), pp. 382–404. doi:10.1017/S0008197301000162.
 25. Coulter, Rory, and Michael Thomas. "A New Look at the Housing Antecedents of Separation." *Demographic Research*, vol. 40, 2019, pp. 725–60. JSTOR, <https://www.jstor.org/stable/26727015>. Accessed 4 Apr. 2024.
 26. Easthope, H. (2014) 'Making a Rental Property Home', *Housing Studies*, 29(5), pp. 579–596. doi: 10.1080/02673037.2013.873115.
 27. Soaita, A. M. et al. (2017) 'Becoming a landlord: strategies of property-based welfare in the private rental sector in Great Britain', *Housing Studies*, 32(5), pp. 613–637. doi: 10.1080/02673037.2016.1228855.
 28. Martin, C., 2004. Law and Order in Public Housing: the Residential Tenancies Amendment (Public Housing) Act 2004 (NSW). *Current Issues in Criminal Justice*, 16(2), pp.226-232.
 29. Tunstall, R. (2003) "Mixed tenure" policy in the UK: privatisation, pluralism or euphemism?', *Housing, Theory and Society*, 20(3), pp. 153–159. doi: 10.1080/14036090310019445.
 30. Gibb, K. (2015) 'The multiple policy failures of the UK bedroom tax', *International Journal of Housing Policy*, 15(2), pp. 148–166. doi: 10.1080/14616718.2014.992681.
 31. Shankley, W. and Finney, N., 2020. Ethnic minorities and housing in Britain. In *Ethnicity, Race and Inequality in the UK* (pp. 149-166). Bristol: Policy Press.
 32. Adriana Mihaela Soaita, Kim McKee, Assembling a 'kind of' home in the UK private renting sector, *Geoforum*, Volume 103, 2019, Pages 148-157, ISSN 0016-7185, <https://doi.org/10.1016/j.geoforum.2019.04.018>.
 33. Carr, H., 2002. Renting Homes 2: Co-Occupation, Transfer and Succession. A Consultation Paper. The Stationary Office.
 34. <https://www.landmarkchambers.co.uk/wp-content/uploads/2018/07/Landlord-and-Tenant-Recent-Cases-John-Male-QC-Katharine-Holland-QC.pdf>
 35. Law Commission, 1996. Landlord and tenant: Responsibility for State and Condition of Property. The Law Commission, London.
 36. <https://www.camden.gov.uk/documents/20142/2134745/Rent+Repayment+Order-Tenants+Guide.pdf/03d9c275-6cca-c7ab-0363-34e72fc0409f>
 37. <https://researchbriefings.files.parliament.uk/documents/SN01998/SN01998.pdf>

30,230 No fault eviction were served last
year Eviction Cases

Sheffield City Council's housing disrepair cases, the number of new cases the council received as of April 2018 was 117, compared with 1,970 as of April 2023.

The cost of the cases, including legal fees and compensation paid to residents, has increased from £292,655 in 2018-19 to £2,986,269 in 2022-23, up 920%.

8.5 million households renting in the
UK