Functional Specification
DocGenie

Student 1: Shane Power
Student Number: 21308533

Student 2: Razvan Gorea
Student Number: 21306373

Date of Completion: 14/11/2024

Table of contents

1. Introduction

1.1 Overview

DocGenie is an application designed to provide a unified point of query for a collection of fragmented databases and datastores, that works through a multi-agented Retrieval Augmented Generation system. This system also retains the access control the user would have faced when querying the base datastores.

This unified point of query takes the form of a chatbot, to allow a variety of users to utilise the system.

This system also retains scalability, and so is suitable for large amounts of datastores. We retain scalability through the use of multiple agents for the retrieval process. Multiple agents allows you to specialise each agent towards a different distinct function, thus allowing for 'affinity' towards certain tasks. For example, you might train an agent just for retrieval, while having another agent just for reconciling newly added datastores.

This system solves the issue of organisations having lots of fragmented datastores, and so being unable to utilise lots of the data within them for future querying. By providing a unified interface, our system ensures that this data can be used far into the future.

A key part of this system is its ability to integrate with existing data stores, without too much need for standardisation of the underlying data. If the data is ingestible into the central vector database, it can be queried, regardless of whether it was in CSV format or JSON format.

1.2 Business Context

While a direct business organisation sponsorship isn't applicable to this project, this project has been prompted by a problem that many DCU staff and departments can relate to. There's an information need that can't be easily satisfied due to the fragmented nature of how every different DCU department handles their information processing and storage.

These fragmented data stores make it very time consuming and challenging to retrieve information outside of one's department. Problems such as no access to the necessary relevant information systems coupled with a lack of familiarity with using these external information systems come up time and time again. These problems are currently addressed through constant communication between these different DCU departments, which is very inefficient.

The following use case illustrates this issue:

The president of DCU is having a meeting with an IBM representative, and so he needs to know about every prior interaction between IBM and DCU. The president would have to contact multiple different departments, people and sources to gain all the necessary information. Even still, the president might not have the technical knowledge to make use of all of this data being returned to him, as it might be in a variety of formats.

This use case illustrates the necessity of a centralised point of query, presented in an understandable format.

1.3 Glossary

RAG (Retrieval Augmented Generation) - Technique for improving the accuracy of generative AI models by providing them context, along with the original prompt, to generate richer feedback.

Multi Agent - A system where many autonomous agents are working within an environment either separately or as a collective to tackle complex tasks.

LLM (Large Language Models) - Artificial intelligence model that has been trained to understand and generate text.

Agent - a process that performs a specific task as part of a system.

Context - relevant information to a user's query that a LLM receives and uses to generate a response.

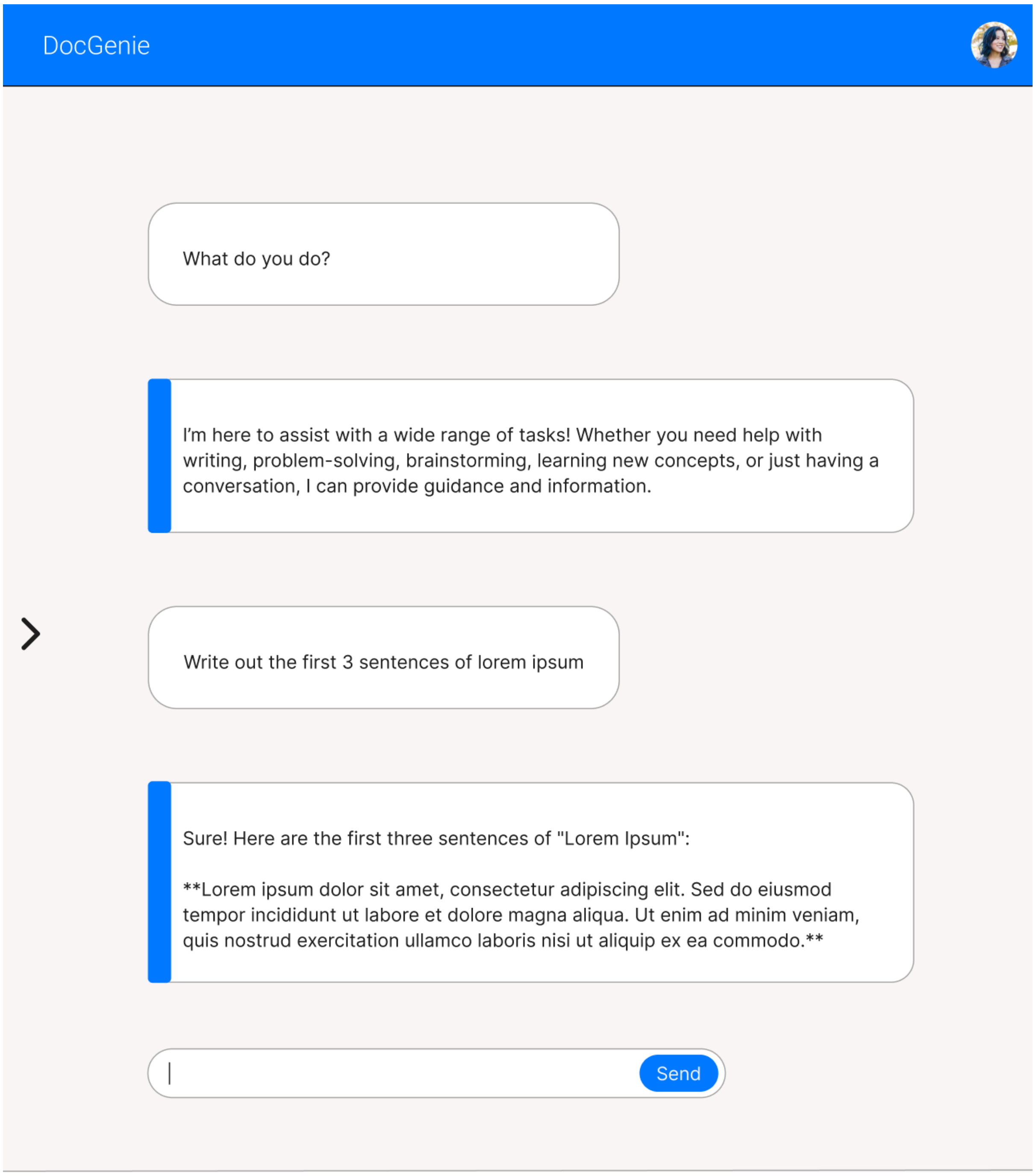
2. General Description

2.1 Product / System Functions

The functionality of the project can be divided into the following main areas:

Web User Interface

The below image is a mockup of the main chat interface:



What do you do?

I'm here to assist with a wide range of tasks! Whether you need help with writing, problem-solving, brainstorming, learning new concepts, or just having a conversation, I can provide guidance and information.

Write out the first 3 sentences of lorem ipsum

Sure! Here are the first three sentences of "Lorem Ipsum":

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo.

External LLM

The llama3 model 8 billion parameter variant. This will generate the user response when given the user prompt with the context from the integration point. The generated response will then be sent to the user interface to be displayed for the user.

Response guarding

User credentials with associated permissions will be stored in a dedicated permissions database. When a logged in user sends a prompt on the user interface, their associated permissions are also pulled from the permissions database. Their permissions are sent along with the prompt. The permissions associated with that specific user will then define what documents and context gets retrieved within the context retrieval stage, additionally altering which vector embeddings gets delivered to the external LLM.

Context retrieval

All context retrieval is done through a single specially trained agent. The agent receives a vectorised user prompt, and takes a number of steps to retrieve context relevant to that vectorised prompt. The agent has access to both the prompt history, and the response history, and so can compare against those two stores to see if it needs to make queries itself. If it does need to make a query, it will exclusively query the vector database, and vector-compare the two data sources.

Integration point

The integration point serves as the central point between the vector database, the external LLM and the Web User Interface. The user's original prompt will be vectorised at this point, and the retrieval agent will use this with the vector database. The agent responsible for response guarding will also return its response to this level.

The integration point is the only module that interacts with the external LLM, sending both the prompt and relevant context, and also interpreting the responses from the external LLM.

Vector embedded database

The vector database holds the vectors that represent the data held in our different datastores. We hold the information as vectors so that our agents can easily perform vector mathematics on the data, such as checking for similarity to our vectorised user prompt. At this point we also hold some additional data, such as the vectorised query history and answer history, to speed up retrieval.

Both our retrieval and response agents work with this database as their primary point of reference.

Datastore Ingestion

This layer is responsible for ingesting the various datastores into our larger vector database. At this stage, we have received the means to connect to these datastores, and so now we work to ingest all current data into the system. We must handle different datastores differently, for example ingesting SQL databases will take the format of CSV, while ingesting data from MongoDB will take the format of JSON.

If we have ingested a datastore, and new information is made available on that datastore, it is the responsibility of a specially trained agent to integrate these changes into our overall system.

2.2 User Characteristics and Objectives

Before discussing user objectives, it is important to outline our general user-base. Our user base is employees of large organisations that have a need to get information relevant to their work. For example, an employee of DCU, who has a need for information regarding their work, such as what organisations their user interacts with. As such, we can not expect these users to have any real technical expertise. We can expect them to be familiar with computers as a general topic though.

From the users perspective: the system should be able to achieve the following:

Take a prompt from the user via the chat interface

Return an answer to my query, with reference to both my original query, and data unique to the organisation I work in. This information must be information I am allowed access to.

From a developer perspective, this is achievable. The user does not have complex requirements, so developing a ‘black box’ is acceptable in this case, as the user only cares that our response is accurate. We also do not have to worry too much about user freedom, as they have a very specific need for information when they decide to use the system, so we can keep to a very streamlined order of operations, with little need for deviation.

2.3 Operational Scenarios

Due to the lack of user freedom in how they interact with the system, user scenarios are limited in number. In reality, there is some significant branching in terms of the retrieval agent, and so the below scenarios will be more focused on the backend, as the frontend is largely static during the query process. From a user perspective, there are only seven main scenarios.

Title	User Account Creation
Actors	User, Auth
Description	As a user, I can create an account to enter and use the system.
Events	The user is greeted with the login page, they click the register button. The user is sent to the register page, they are asked to provide a username, email and password. The user fills in the required fields and presses the register button. Their account is created and they are sent back to the login page.
Alternatives	The system doesn't create an account if the proposed username or email is already used by another account. The user will then be prompted to either enter a different username, email or both.
Business	The user must have an email.

Title	Ask a question and receive a response.
Actors	User, Retrieval Agent, Response Agent.
Description	As a user, I want to ask a question via the UI and receive a response to my question.
Events	The user clicks the chatbox. The user types a query and sends it. The query is matched against some context in the datastores. The matched context is given to the external LLM. The response generated is given back to the user.
Alternatives	If no context is found to be matched with the query, We do not access the LLM. We return an error to the user. If the response generated does not match the guarding requirement, we regenerate the response.
Business	The user must be logged in.

Title	See my past queries
Actors	User, Auth, Query History
Description	As a user, I should be able to get my past queries.
Events	The user clicks the ‘history’ tab on the UI The user now sees a list of past queries, ordered by date. The user clicks the query they are interested in, to view details such as the response.
Alternatives	If the user has no query history, the history tab will not be visible.
Business	The user must be logged in.

Title	Delete my history
Actors	User, Auth, Query History
Description	As a user, I want to be able to delete my history.
Events	The user is logged in. The user clicks the history tab. In the history tab, the user clicks the ‘clear history’ button. The user confirms this is what they want to do via a second dialogue. The user's history is cleared.
Alternatives	If the user has no query history, the history tab will not be shown.
Business	The user must be logged in.

Title	Context should be permission sensitive
Actors	User, Auth, Context Guard, External LLM
Description	As a user, I want to only receive responses based on the context I have access to.
Events	The user sends a query. The user’s permissions are retrieved. The user’s permissions with the user query is used to get the relevant vector embeddings. The vector embedding is checked again to see if it aligns with the user’s permissions. The LLM retrieves the vector embeddings and user query. The LLM generates a response based context sensitive response and sends it to the user interface.
Alternatives	The vector embeddings retrieved don’t align with the user’s permissions. Retrieve the embeddings again and check if they align with the user’s permissions. Retry this process thrice.
Business	The user’s account has at least one permission tied to the account.

Title	Context should be up to date
Actors	User, Search Agent, Ingestion Agent
Description	As a user, my searches should be using the most recent context available.
Events	The user has made a request. We check that no updates to the context are ongoing. We now use the context that we confirmed is up to date. Continue the search process.
Alternatives	If we can prove that the context is out of date or undergoing updates, wait. If the waiting time expires, error.
Business	There is already context present in the system.

Title	Responses should be readable
Actors	User, External LLM, Context Guard
Description	As a user, when the system generates a response, this response should be readable.
Events	The user has made a query. The system has retrieved context related to this query. The system passes this context to the external LLM. The response of the External LLM is then passed back to guarding. The response is checked by the context guard, to ensure it resembles human speech to some degree. The response is given to the user.
Alternatives	If the LLM does not respond, we retry thrice. If the context guard does not think the response is human readable, we error.
Business	n/a

2.4 Constraints

The design team needs to assure that the user interface can be accessed on the latest browsers (i.e FireFox, Chrome). The design team needs to implement good scalability as the volume of data increases within the MongoDB database. The design team needs to ensure that the time between the user sending a prompt and receiving a response doesn't exceed 1 minute. The design team needs to ensure that users don't get responses that mention data they shouldn't have access to. The design team will need to ensure proper GPU hardware is employed to run the LLM and the RAG system either through self hosting or cloud computing (i.e. Google Cloud) for fast processing.

3. Functional Requirements

3.1 The system must be able to ingest data from multiple data formats

• **Description** - the MongoDB database must support data from many different formats, more specifically JSON, CSV and BSON. External data store connections with the system should be possible.

• **Criticality** - the system becomes useless if no data is present. Context Retrieval & LLM response generation won't occur properly or not at all. The problem the system is trying to solve can't be accomplished, making this requirement extremely necessary.

• **Technical Issues** - potentially the design team would have to manually establish this requirement on a case by case basis. This requirement can't be automated due to the nature of there being many different data formats and data stores. The overall implementation of this requirement will have multiple tailored solutions for only a select few cases as it isn't feasible to support an infinite amount of formats and data stores.

• **Dependencies with other requirements** - This requirement would have interactions with every other requirement except **(3.8)**. In relation to requirement **(3.2)**, if requirement **(3.1)** isn't implemented the system will never change its responses since no data is present and changing. In relation to requirement **(3.3)**, no context would be available for the LLM to utilise. For requirements **(3.4 & 3.5),** the retrieval agent won't have any data to concern itself with if there's no implementation of requirement **(3.1)**. For requirement **(3.6)**, no context will be returned by the retrieval agent regardless of a user's permissions if requirement **(3.1)** isn't present. For requirement **(3.7)**, the LLM will only generate responses without context which won't fully satisfy a user's information need.

3.2 The system must be able to change its responses as the underlying data changes

• **Description** - When the underlying data, contained with the original client provided datastores changes, then we must be able to recognise these changes and account for them when we retrieve context. For example, if a database we have ingested has 200 new rows written into one of its tables, we must be able to recognise that this change happened, and take into account this new data, before the next query is made. In practice, this means that a specialty trained agent, responsible for checking for changes, can update the vector database with new data when needed.

• **Criticality** - This requirement is of high importance. If we are unable to update our vector database to utilise new data, then our retrieval agent will quickly become inadequate, as it will lose knowledge of the available context.

• **Technical Issues** - Implementation wise, we could have an agent to check each datastore at certain times each day, and see if new data has become available since it last checked that datastore. If it finds new data, it can trigger the ingestion process to update the vector store.

• **Dependencies with other requirements** - We are dependent on requirement 3.1, as we require the ability to ingest data before we can check for changes to the data.

3.3 All available data must be able to be used as context

• **Description** - When the retrieval agent is looking for context related to the users query, we must search through all available data stores that we have permission to access. If we search through only a fraction of the available datastores, we risk not discovering relevant context.

• **Criticality** - This is of a lower criticality than some other requirements. For example, if we only search through 80% of the available datastores, then we still have a high chance of retrieving all relevant context. If this coverage decreases, then this issue becomes much more important.

• **Technical Issues** - Determining when to stop while searching is problematic. We really do not want to exhaustively search for context, so an implementation that reduces our search time while retaining a high search coverage is required. Keeping knowledge of past queries, and the context they returned, would be very useful.

3.4 The retrieval agent must not alter the underlying context data

• **Description** - The retrieval agent is specially designed to only ever retrieve context. As such, the agent must not have any ability to change underlying data. This would violate the concept of specially trained agents, and would give far too much for this agent to control. This must be enforced both for the vector database, and the underlying datastores.

• **Criticality** - This requirement is of the highest criticality. The consequences of letting the retrieval agent modify data would be detrimental for both the user experience and the overall system.

• **Technical Issues** - Preventing the agent from modifying data is actually trivial. We are training these agents, so we simply train the agent to have no knowledge on how to modify data, and we should accomplish this.

• **Dependencies with other requirements** - This requirement is related to requirement 3.2, as there we define what exactly is able to change the underlying data.

3.5 The retrieval agent must avoid using unauthorised context

• **Description** - When the retrieval agent tries to obtain context related to the users prompt, it must not be able to view any context which does not match the users security access. The agent must instead only retrieve context it has explicit permission to retrieve.

• **Criticality** - This is of some importance, as the implications associated with allowing a user to read data that they do not have access to is problematic. However, the agent responsible for response guarding can also catch this issue later on in the serving process, and so the needed guards at the retrieval agent are not as important.

• **Dependencies with other requirements** - We are dependent on the retrieval agent being passed what datastores it is allowed access to.

3.6 The response must not contain context that the user is not authorised to see

• **Description** - After the retrieval agent has found context that is relevant to the user, it returns this context to be assessed by the response guarding agent. This guarding agent evaluates the context in respect of the user's permissions, and if the context contains forbidden content, The context will not be passed to the external LLM, therefore generating no response which could be served to the user.

• **Criticality** - This is higher criticality than requirement 3.5, as this is our final chance to review the context before generating a response. If we miss any security breaches here, the user will be served data they should not have access to.

• **Technical Issues** - Implementation wise, there are issues with determining whether context is not suitable due to security. Approaches like comparing the returned context with vectors we know are in forbidden datastore is possible, but expensive and time consuming, depending on the scale of the datastores.

3.7 The LLM must generate a response and send it to the user interface

• **Description** - The LLM must take (1. user's prompt) along with the (2. vectorized embeddings (context)) and generate a text response based on these two parameters. Once the response has been created by the LLM, the LLM would then transmit the response to the user interface.

• **Criticality** - This requirement is very essential as without it being satisfied the user will not be able to receive any responses at all for any prompts or queries they submit on the user interface.

• **Technical Issues** - One technical issue could be the connection between the LLM and user interface. The LLM could either not receive the user's prompt or it could receive a user's prompt, generate a response but can't return it back to the user interface because of a connection issue. Another issue could be hallucinations, a hallucination is where a LLM produces an inaccurate response. To tackle this issue, it's important to ensure the LLM is rich in context, adequately tested and observed.

• **Dependencies with other requirements** - this requirement is dependent on the context retrieval requirement(s) and the retrieval agents requirement(s). The LLM depends on the context retrieval process and retrieval agents doing their tasks properly as the richer and larger the context provided to the LLM, the less likely it's prone to hallucinations. Lastly, there is also a dependency relationship between the LLM and user interface. The user interface needs to receive the response from the LLM in order to render the response onto the webpage, as well as vice versa the LLM needs to receive a prompt from the user interface to start generating response in return.

3.8 The user interface must send the user’s response to the LLM

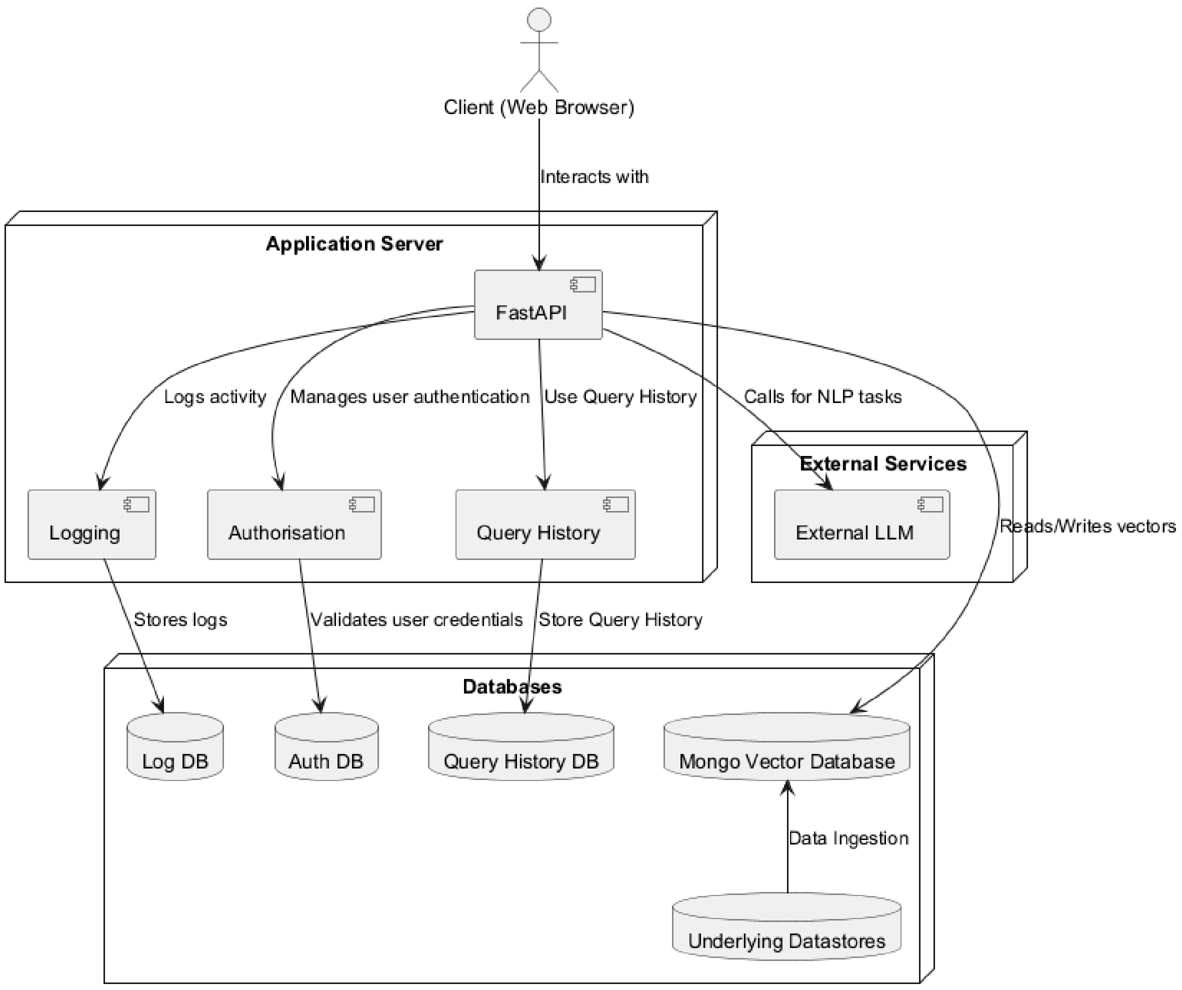
• **Description** - when a user types a prompt in the prompt bar and clicks the submit button or hits their enter key, this sends the user’s response to the LLM.

• **Criticality** - not to the overall system but still necessary for the end user. This requirement could be absent and the LLM could still be queried with LangChain/LangGraph.

• **Technical Issues** - the only issues that could arise is the events related to user interface aspects (e.g buttons, prompt bar) not working as expected. An example being the prompt submit button doesn't actually submit the user’s prompt when clicked.

• **Dependencies with other requirements** - no dependencies with other requirements.

4. System Architecture

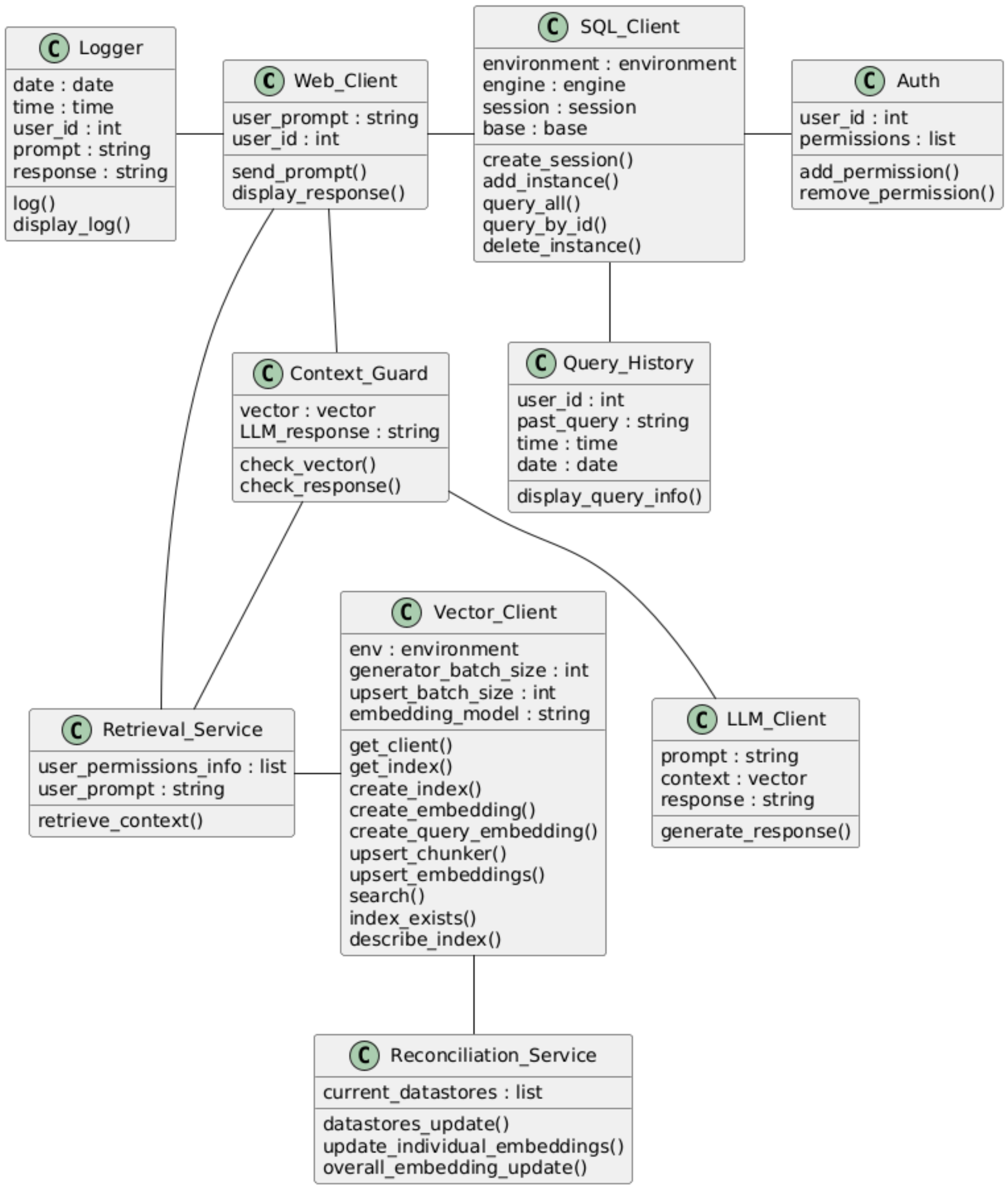


The above diagram shows our basic architecture, independent of specific modules. As such, the intricacies of the interaction is lost here. We can see that the FastAPI, within the Application Server, is of high importance within the architecture of the system. We can consider this layer as an orchestration layer, through which we can centralise much of our flow. This orchestration layer simplifies our interactions, and importantly lets us centralise our access control issues, which will simplify our implementation.

One other point to consider is the relation between the databases. While on the architecture side, they appear to have multiple points of query, in reality, we are not accounting for the existence of the SQL client, which in reality provides us a unified point to query all of these databases. From an architecture perspective, these databases are independent.

5. High-Level Design

5.1 System Components

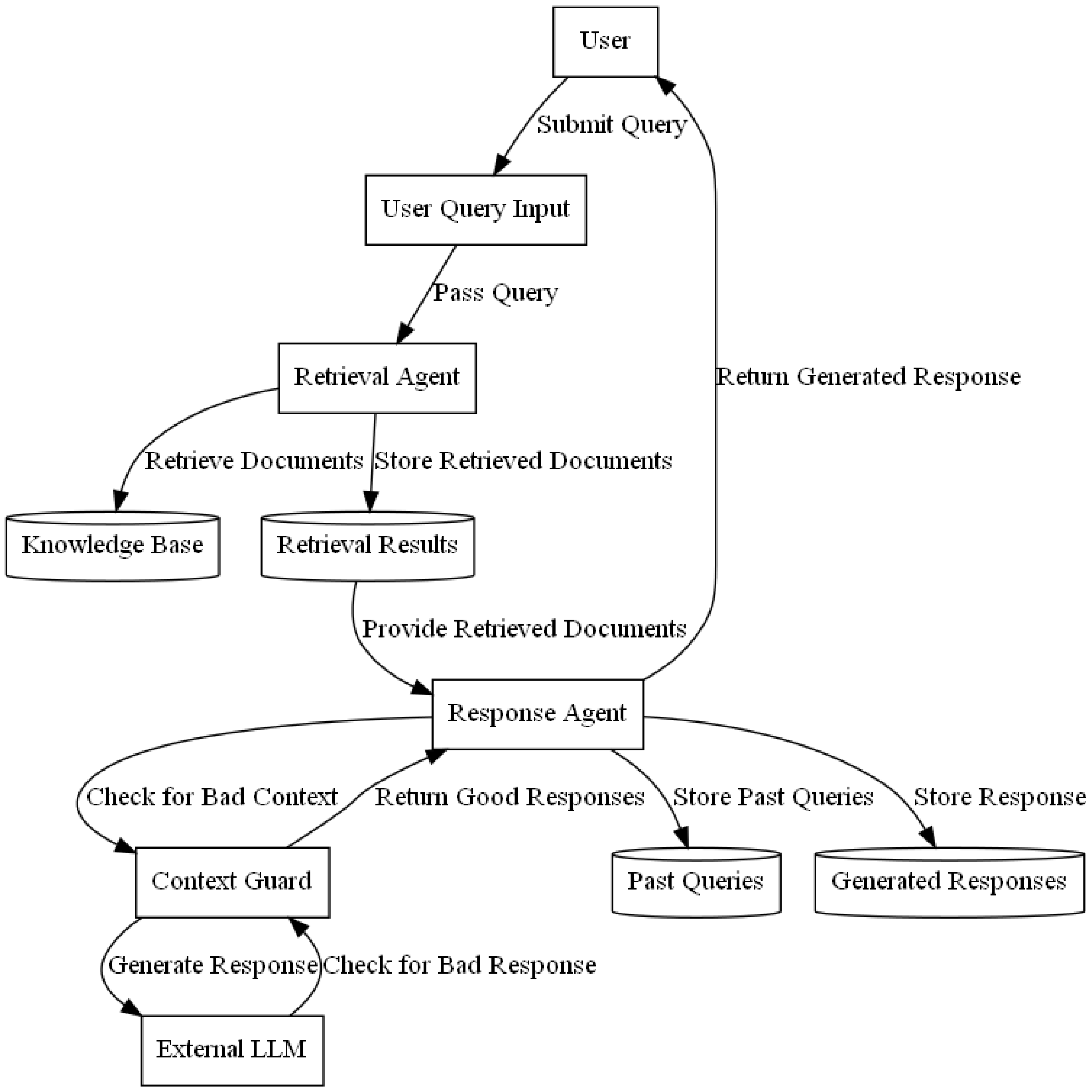


This is a class diagram of the system. This diagram illustrates every interaction of each class amongst other classes and in the grand scheme of the entire system.

An emphasis should be made on the Context_Guard and Retrieval_Service class. The Web_Client class interacts first with the Retrieval_Service class and the Context_Guard class last. The Context_Guard class only interacts with the Web_Client class when a valid secure response has been generated for the user.

5.2 Data Flow

The below Data flow diagram shows how a user query data would be handled through the system.



One point to note regarding this Data Flow Diagram, is that the ingestion process is not being represented here. It is not directly related to the user, and so to avoid bloating the diagram, we exclude it.

5.3 Security

The important fact to note before discussing security, is that we are talking about the security aspect of retrieval and ingestion. We are not really concerned with aspects of security such as site security.

We will start with the data ingestion process. There are two main scenarios in ingestion where we need to concern ourselves with security.

Firstly, when we are ingesting an entirely new datastore, we need all context within this datastore to have some common security ‘tags’, so the retrieval agent knows what it should and shouldn't retrieve. We accomplish this by requiring all data ingested to have security metadata in its vector embedding. We will not ingest any data where we can’t attach security metadata to it, as this context would cause huge complications, and would cause the retrieval agent to become confused.

Secondly, if we are finding content that is violating the security constraints, i.e. failing on the context guards, then we must terminate the whole process. We have some tolerance for failing a context guard: Three tries is an appropriate tolerance. While this might seem like an excessive delay in getting the user a response, improving the retrieval agent will indirectly reduce the chance of this occurring.

5.4 Performance

Performance is obviously a large part of DocGenie. As the system is intended for use within large businesses, with large amounts of data, a slow system will not be acceptable. Slow fetch speed within any information retrieval system is generally not acceptable to the user. As such, we can identify a few bottlenecks in advance, and describe the methods to optimise each of them. We have ordered the bottlenecks from least important to most below.

1. Data ingestion, where our system tried to both initially ingest datastores into the system. This process is a bottleneck as we require ingested data to be able to use it as context. If this process is slow, we delay the availability of that context. This is of some concern, but is not the worst bottleneck, as users can use all previously ingested context while they wait for new context to be ingested. TO optimise this process, we essentially need to optimise how many steps we take to create a write query to our MongoDB database. That means writing performant SQL queries, and not using auto generated ones, as they tend to be slow.

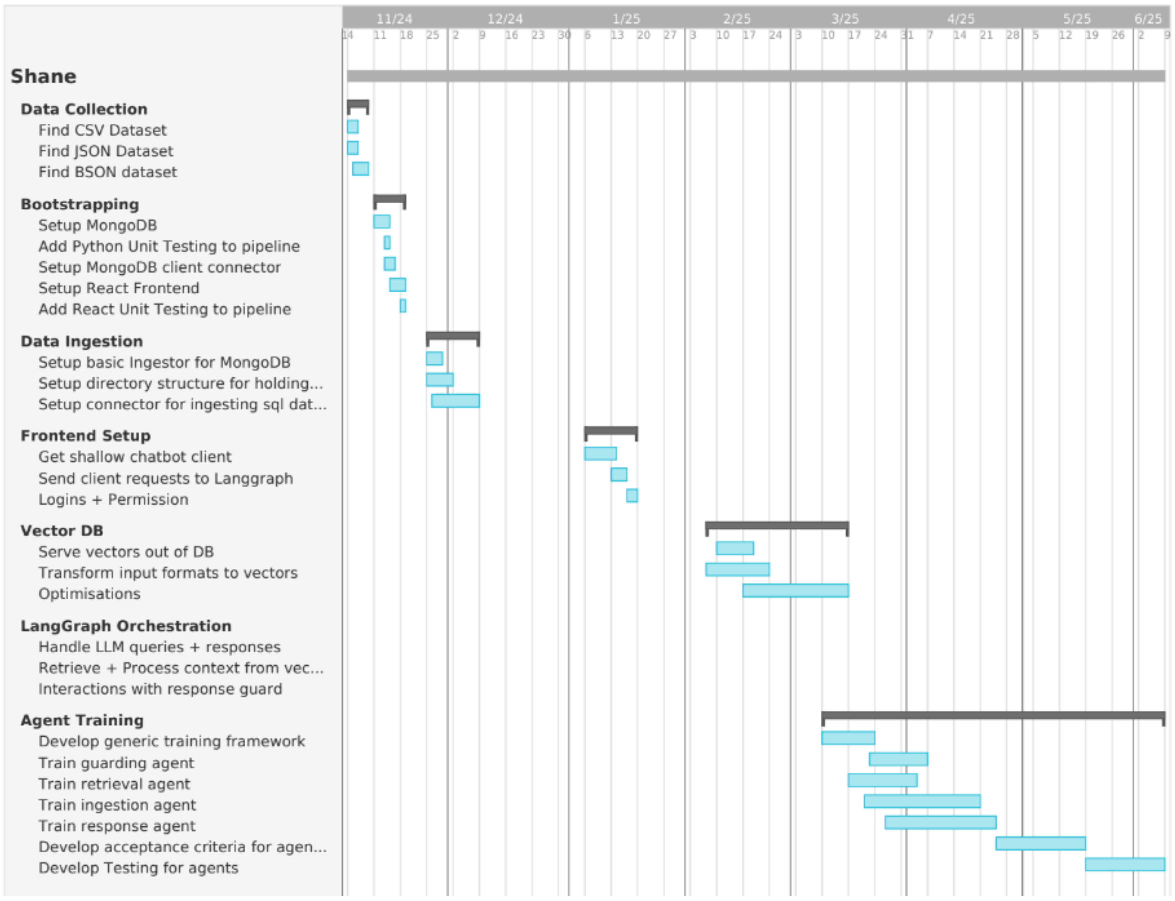
2. The external LLM, which we serve context and our prompt to, can be subject to long response times if we pass it too much context, or ask for too high responses. We can circumvent this by limiting response length, and improving the retrieval agent.

3. The response guard does need to be optimised, as trying to determine if retrieved context has originated from a database that the user does not have access to can be time consuming. We can optimise this process by letting the response guard directly view the db context too, so it can do simple matching against the underlying context. This is made easier by only comparing the vector data, which is quicker to query. We can also tune down the 'scrutiny' of the response guarding agent, and instead settle for a 'good enough' approach, which would save on time.

4. The largest bottleneck is the search agent. As our datastores scale and our vector database scales according to the underlying datastores, our search will get slower. Thankfully, vector search scales very gracefully, and is the only way we are able to search our data at any acceptable speed. By using Available Nearest Neighbours in our search, we retain sub-linear time on our searches relative to the datasets. By restricting the datasets available to search through our security enforcement, we can speed this up, albeit by only a small amount.

6. Preliminary Schedule

Before discussing our scheduling it is important to preface this section by stating that this schedule is optimistic, rather than pessimistic. If we are too restrictive in our preliminary schedule, we might end up with gaps in our scheduling. As such, we have tried to fill this schedule. Importantly, the prioritisation of these tasks will become vital in actual implementation, as we will likely encounter time pressures.



Based on the GANTT chart above, we can see that we anticipate some sections taking more effort than others. In particular, we think that training the agents will be particularly problematic. Other topics, such as bootstrapping the project, are more trivial. The ordering of our deliverables are quite important. Our later topics such as training the agents, or building the LangGraph orchestration layer, are reliant on data ingestion being present, and so we have laid out our tasks accordingly.

We have also ordered out tasks within the GANTT so we have some form of application running quite quickly. This ensures we have some form of deliverable, if there are issues.