Bilkent University
Department of Computer Engineering

Senior Design Project
T2431
Smart Vector Query

Analysis and Requirement Report

Mehshid Atiq 22101335

Muhammad Rowaha 22101023

Ghulam Ahmed 22101001

Yassin Younis 22101310

Zahaab Khawaja 22101038

Supervisor: Uğur Doğrusöz

Innovation Expert: John Yuzdepski

16.12.2024

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfilment of the requirements of the Senior Design Project course CS491/2.

Table of Contents

Analysis and Requirement Report

SVQ: Smart Vector Query

1. Introduction

In today's fast-paced regulatory landscape, efficiently navigating complex documents can be a daunting challenge for professionals across various industries. To address this need, Svq.ai is being developed as a cutting-edge platform that leverages state-of-the-art Retrieval-Augmented Generation (RAG) technology, combined with advanced large language models (LLMs) and modern vector databases. The platform allows users to upload a variety of documents, from regulatory texts to codebases and textbooks, which are then processed semantically for enhanced retrieval capabilities. By employing sophisticated chunking and embedding processes, Svq.ai ensures that users can seamlessly query large files through an intuitive chatbot interface, receiving precise and context-aware responses. The system is designed not only for efficiency and accuracy but also with a strong focus on user experience, data privacy, and scalability, making it an invaluable tool for professionals seeking to streamline their interactions with regulatory documents.

2. Proposed System

2.1. Overview

Under the hood, svq.ai operates using state of the art large language model (LLM) technology alongside modern vector databases and embedding algorithms in order to store and retrieve information based on users' queries. Users start by uploading their own documents as the data source; these documents can be anything ranging from regulatory documents, codebases, or even textbooks. They are processed based on semantics and stored in a database. Users then ask questions through a chatbot interface, with their query being used to semantically search the database for the most relevant and precise answers.

One of the key aspects is the chunking and embedding process of the uploaded pdf, for instance. Since our aim is to enable the users to query large files, the system architecture of Svq.ai naturally feels distributed; a master node orchestrates chunking and embedding of large files. Based on configurations, for example, a worker node can be spinned up for every 10MBs of the input file. A task queue can be used for persistent asynchronous execution. Generated embeddings can then be returned to the master node. This way, worker nodes can be scaled based on the input file size. Moreover, since users will be uploading their documents to the cloud, data privacy is also a major concern. We aim to achieve this by storing hashed files instead of original files. Each user will be assigned a symmetric key for each datasource. If required, the file can be decrypted for the user and annotated accordingly based on query results. Cryptographic hashing of large files can be a slow process, and hence will be off loaded to another node in the system. For improving availability, file/block replication can also be implemented (cloud services e.g. Amazom S3 support data replication out of the box). To scale inferences/queries to the vector databases, serverless functions can be used; queries will be short-lived and running a dedicated service constantly maintaining a connection-pool to the database and/or third-party resources (e.g. OpenAI API) can be costly. Using serverless functions also suits this need because users can query back-to-back and function execution environments do not shut down immediately–consequent requests will have 'warm' startups for

a short period of time. Lastly, The system will have a single point of entry for all users; this node will be responsible for general user management and user authentication, and orchestration of processes using webhooks.

2.2. Functional Requirements

The regulatory document querying platform, svq.ai, aims to simplify interaction with regulatory documents through an advanced RAG-powered system. Here are the comprehensive functional requirements necessary for successful implementation and deployment.

→ User Interface and Navigation

The system requires an intuitive web interface that prioritizes user experience. The primary navigation structure must include clearly defined sections for Home, Reports, and Contact pages, with a persistent navigation bar across all pages. The interface necessitates prominent call-to-action elements, including "Try it now" and "Get started for free" buttons to maximize conversions. The chatbot interface must be seamlessly integrated into the main application, providing users with a natural conversation flow for document queries. The design must maintain consistent branding elements and styling throughout the platform to ensure a cohesive user experience.

→ Document Processing and Search Capabilities

At the core of the system lies the Retrieval Augmented Generation (RAG) technology implementation. This technology must process regulatory documents with high accuracy while reducing hallucination. The system must accept natural language queries from users and process them to provide accurate, contextual responses without hallucinations. The search functionality requires advanced filtering capabilities that allow users to refine their searches based on multiple parameters. The system must maintain an efficient indexing system for all uploaded documents, ensuring quick retrieval and processing of information.

→ User Management System

The platform requires a good user management system that supports both free-tier and premium access levels. User authentication and authorization mechanisms must be implemented with industry-standard security protocols. The system must track and maintain user sessions securely while storing user preferences and search history. Profile management capabilities should allow users to customize their experience and manage their document collections efficiently.

→ Regulatory Document Management System

The document management system must support various document formats commonly used for regulatory documentation. Version control functionality is essential to track document changes and updates over time. The system requires batch processing capabilities to handle multiple documents simultaneously while maintaining processing efficiency. Document security measures must be implemented to ensure the confidentiality and integrity of uploaded materials.

→ Search Results and Response Generation

The response generation system must provide context-aware answers derived directly from the source documents. Responses should include relevant citations and references to source materials. The system must support the export of search results and findings in multiple formats suitable for reporting and analysis. An audit trail system must track all queries and responses for compliance and quality assurance purposes.

→ Performance and Integration Requirements

System performance requirements dictate response times under three seconds for standard queries under normal load conditions. The platform must handle multiple concurrent users effectively without degradation in performance. Integration capabilities must include API access for enterprise clients, allowing seamless incorporation into existing workflows. The system must maintain compatibility with major web browsers and provide responsive design for various device types.

→ Security and Compliance

Security requirements encompass comprehensive data protection measures, including encryption for data in transit and at rest. The system must comply with relevant data protection regulations and implement appropriate data retention and deletion policies. Audit logging functionality must track system interactions for security monitoring and compliance purposes. Clear terms of service and legal disclaimers must be integrated into the platform to ensure legal compliance.

→ Reporting and Analytics

The reporting system must generate comprehensive analytics on document usage, query patterns, and system performance. Reports must be available in multiple formats and support custom report generation based on user requirements. Analytics functionality should provide insights into user behavior and system utilization patterns to support continuous improvement.

2.3. Non-functional Requirements

→ Usability

To guarantee that users can easily find and use functions, the interface must have clear and simple navigation. With just three clicks from any page, all of the main features should be available, reducing the amount of work needed to complete tasks. To enhance user experience and prevent misunderstanding, all pages must have a uniform style and design language. Interactivity and simplicity should be balanced in the interface so that users may finish challenging activities without feeling overburdened. It should also put aesthetics first, offering a visually pleasing layout that is enjoyable to use. All non-trivial UI elements should incorporate interactive tooltips or informational prompts to improve usability even more. This will help users navigate the system efficiently and without the need for extra assistance.

→ Reliability

The system must implement robust data security measures to ensure that all uploaded documents are protected from unauthorized access. Data confidentiality must be guaranteed through end-to-end encryption, with all data stored in an encrypted format using industry-standard algorithms (e.g., AES-256). During transmission, TLS (Transport Layer Security) protocols must be employed to safeguard data against interception or tampering. Access control mechanisms, including role-based access control (RBAC) and multi-factor authentication (MFA), must be enforced to limit access to authorized users only. Additionally, zero-knowledge architecture can be utilized to ensure that data is not perceivable even by the system administrators or service providers. Regular penetration testing should be conducted to identify and mitigate potential vulnerabilities, ensuring data confidentiality at all times. The system must ensure a minimum uptime of 95%. Robust error-handling mechanisms must enable recovery from failures without data loss. Automated data backups must occur every 7 days, with redundancies in place to safeguard critical information. Rigorous testing, including stress testing, is required to ensure stability and reliability under varying conditions, maintaining user trust and consistent performance in high-stakes applications.

→ Performance

The system must ensure consistent performance, tailored to the app's specific functionalities. Response times for standard interface actions should not exceed 5 seconds, while complete responses must be processed within 10 seconds. Notifications should be dispatched within 3 seconds to ensure timely communication with users. File uploads must initiate within 20 seconds, and real-time information transfer via WebSockets must have a maximum lag of 2 seconds. Efficient resource utilization, optimized handling of vector embeddings and large language models, caching mechanisms, and load balancing must be implemented to support seamless scalability and responsiveness, even under heavy usage or complex queries. These measures will ensure a reliable and efficient experience for users engaging with large datasets and regulatory documents.

→ Supportability

The system must include user manuals updated with each release to guide users on app usage and features. Maintenance checks must occur regularly, with seamless, backward-compatible updates to minimize disruption. Users must be notified 1 day before scheduled updates, and downtime should be kept minimal. Comprehensive documentation for developers and administrators must cover architecture, APIs, and troubleshooting. A robust support system with FAQs, and a knowledge base must ensure timely issue resolution and user satisfaction. Monitoring and automated alerts should promptly detect and address potential issues.

→ Scalability

The system must support horizontal scalability by allowing the addition of servers and elastic storage to dynamically adjust based on user and data demands. Vertical scalability should enable scaling of system components, such as CPU and memory, to maintain performance standards during high usage. The architecture must be designed to efficiently handle growing user bases and datasets without compromising response times or reliability. Regular stress testing should ensure the platform’s ability to scale seamlessly as requirements evolve.

2.4. Pseudo Requirements

1. Nextjs bootstraped with TurboRepo will be used for the frontend
2. Python+FastAPI will be used for Svq Service
3. Deno+Express will be used for RAG Service
4. LangChain.js will be used for chunking in RAG Service
5. OpenAI Client will be used for generating query response and embeddings
6. MongoDB will be used as both NoSQL database and Vector Database
7. gRPC will be used at transport layer for all services+web app

2.5. System Models

2.5.1. Scenarios

Account Creation
→ **Use case name:** Create Account
→ **Actor:** User, SVQ Service
→ **Entry Condition:** User gives a valid email, password and username
→ **Exit Condition:** Account is successfully created if not already exists
→ **Flow Events:**
1. User sends email, password and username to svq service
2. Svq service validates if the account does not exist already
3. If the account already exits, svq service returns an error
4. Otherwise, svq service registers the user to the system
5. User is redirected to the login page
6. User types in email and password
7. User is logged in for the first time

Create Datasource for User
→ **Use case name:** Create new Datasource
→ **Actor:** User, SVQ Service, RAG Service
→ **Entry Condition:** User is already logged in to the system. User has a valid json web token
→ **Exit Condition:** New Datasouce with the specified name is created for the user
→ **Flow Events:**
1. User sends the name of the desired datasource to be created
2. If the datasource already exists with that name, svq service returns error
3. Otherwise, svq service registers this new datasource for the user
4. Svq service notifies rag service of this new datasource and awaits response
5. Rag service creates a new embeddings' collection for this datasource
6. Rag service notifies svq service of the creation of this collection
7. Svq service notifies user of the successful creation of new datasource

Uploading File to an existing Datasource
→ **Use case name:** Upload file to Datasource
→ **Actor:** User, SVQ Service, RAG Service, OpenAI Client
→ **Entry Condition:** User is already logged into the system. User has a valid json web token. Target Datasource already exists

→ **Exit Condition:** Embeddings are generated and both the file and its embeddings are persisted into the database

→ **Flow Events:**

1. User selects the file that it wants to upload to the datasource

2. SVQ Service receives file content and file type that was uploaded

3. SVQ Service saves the file to an S3 Object Store

4. SVQ Service notifies user of the successful upload

5. SVQ Service notifies RAG Service of new file and passes the file path

6. RAG Service downloads file from the S3 Object Store

7. RAG Service parses file to chunks and calls OpenAI Client for embeddings

8. OpenAI Client returns embeddings for the given chunk

9. RAG Service saves embeddings to a vector store

Querying Datasource

→ **Use case name:** Query from Datasource

→ **Actor:** User, SVQ Service, RAG Service, OpenAI Client

→ **Entry Condition:** User is already logged in. User has a valid json web token. Datasource with the specified name already exists

→ **Exit Condition:** SVQ Service finishes streaming OpenAI Client response to the user. Or User cancels the query

→ **Flow Events:**

1. User selects the datasource that they want to query

2. User types into a text field the query they want to perform and sends it to SVQ Service

3. SVQ Service forwards the request to RAG Service

4. RAG Service semantically embeds the query, retrieves relevant content and passes the query and content to OpenAI Client

5. OpenAI Client generetas a response to the query

6. RAG Service returns the response to SVQ Service

7. SVQ Service returns the response to User

→ **Alternative Flow Events:**

6. OpenAI Client generates a query to fetch more related content

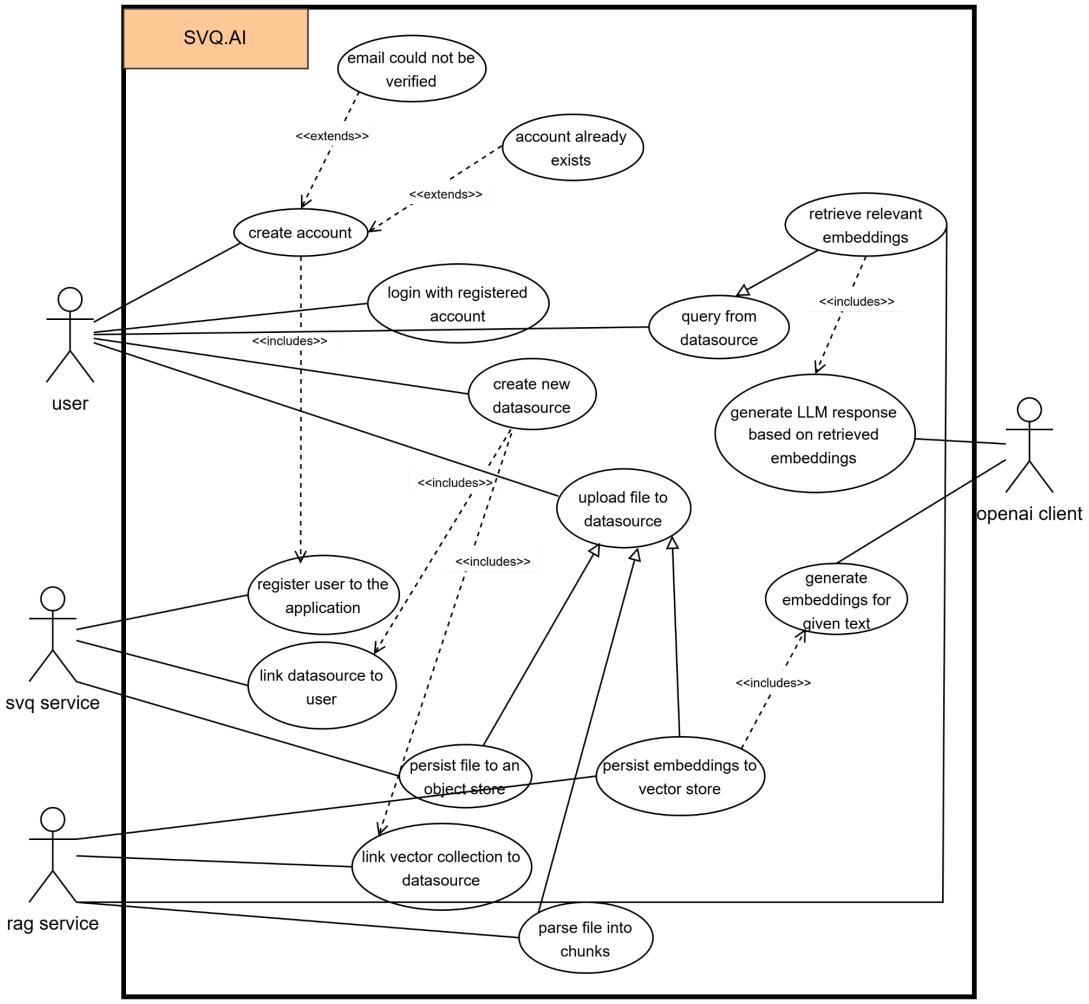
7. RAG Service semantically embeds OpenAI Client's query, retrieves new relevant content and returns to OpenAI Client

8. OpenAI Client generates an aggregated response

9. RAG Service returns response to AVQ Service

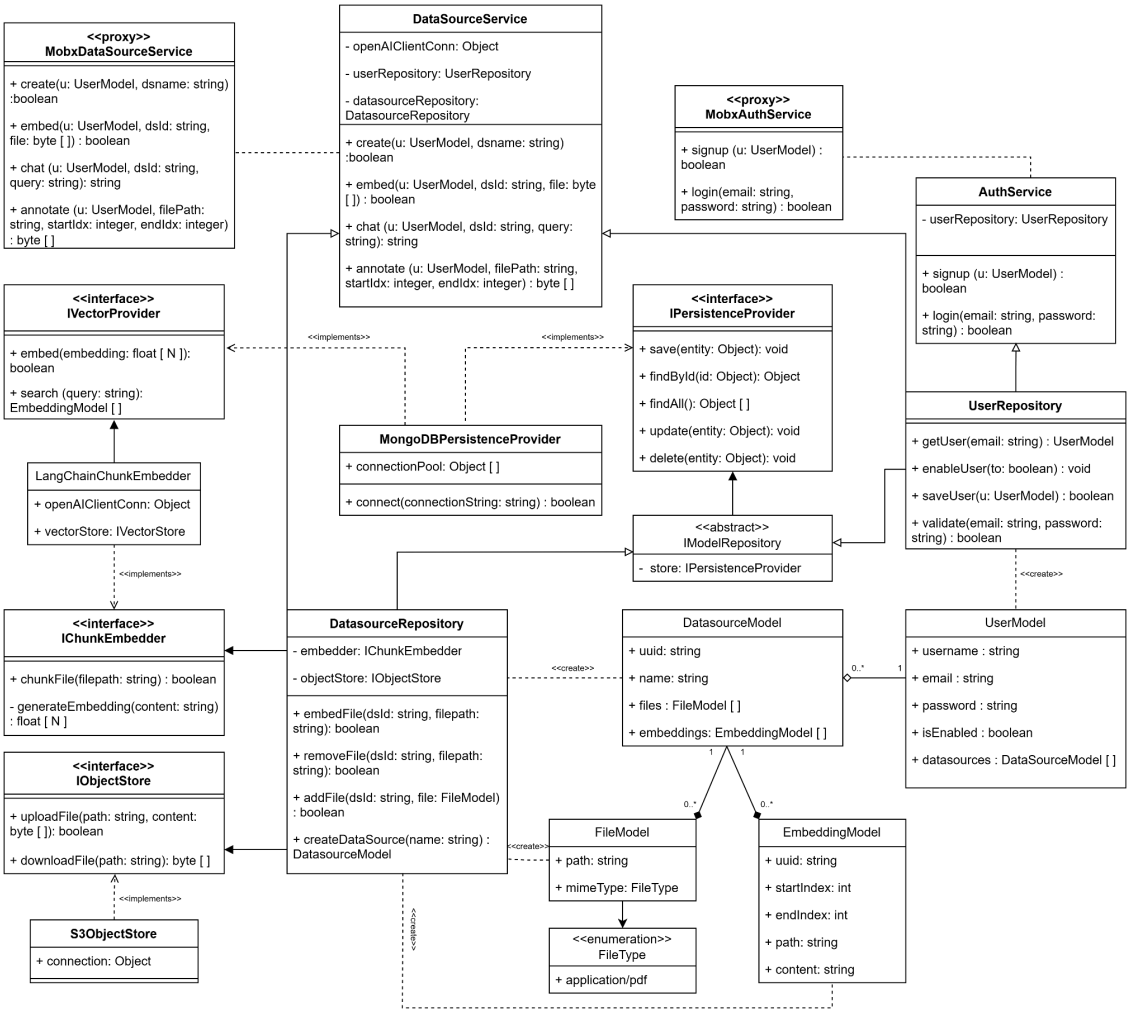
10. SVQ Service returns response to User

2.5.2. Use-Case Model



We have compiled downloaded the actions that can be performed by the user within our current system. However, beware that this is subject to change since our proposed system aims to include features such as public data sources.

2.5.3. Object and Class Model



The software architecture of svq.ai can be clearly seen via this class diagram. The archiecture obeys the MVC architecture. There are 4 basic models that directly co-relate with the entities that will be persisted in the system; these entities are, namely, users, datasources(datasets), files, and embeddings. Models will be accessible to the application via Repositories–namely, User Repository and Datasource Repository.

User Repository is pretty straight-forward; it will be responsible for creating and validating existence of users. User repository will be used directly by the AuthService to expose access management endpoints to the web.

Datasource Repository is a little more complex; it will not only be responsible for creating datasources/datasets but will also manage uploads of files and their embedding generation. This repository will require access to a chunk generation and object store interfaces to persist files and their embeddings.

The transport layer is not shown in the diagram but the concept of service proxies makes use of the fact that client-side code will have mirror services that abstract away the underlying

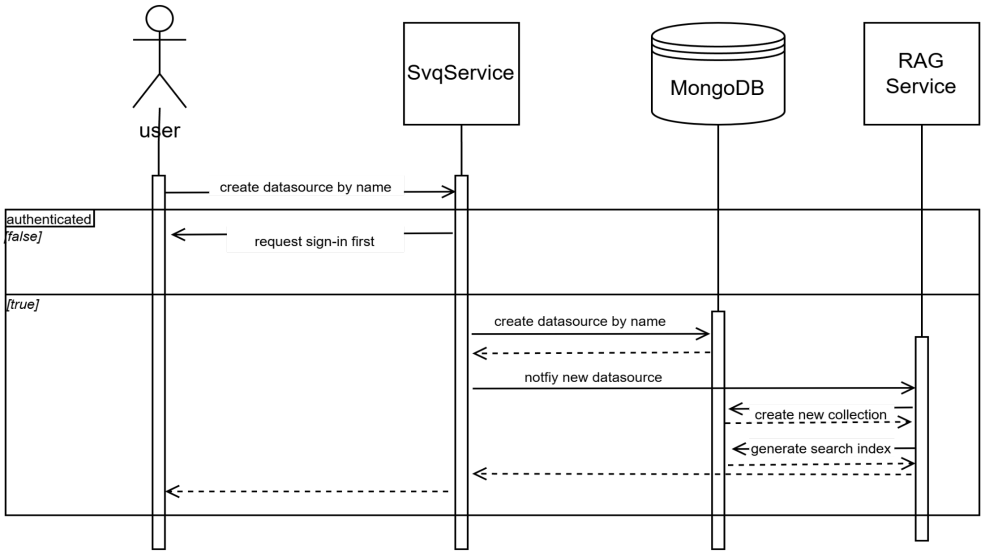
transport layer. In the future, svq.ai will be have the ability to switch protocols, e.g. to gRPC, for improved performance and real-time updates/server side streams.

The software architecture relies heavily on the interfaces, thereby the underlying system architecture can scale as required. For example, access management and RAG services can be decoupled and can interop via an event driven approach. This can improve scalability and maintenance of the entire application.

As can be seen, we have initially chose for persistence of models and embeddings because this allows us to use the same database for both requirements; MongoDB supports vectors. We have also chosen LangChain to implement chunk generation and OpenAI embedding generating algorithm for storing embeddings. For UI-API integration, we will be using Mobx-keystone because of the object-oriented approach of the framework.

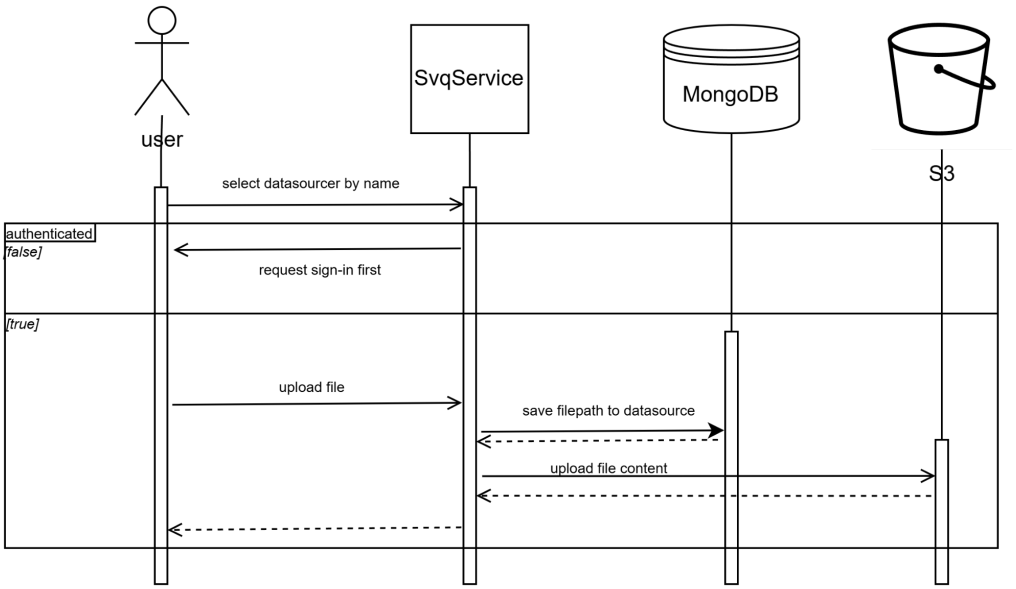
2.5.4. Dynamic Models

→ New DataSource creation



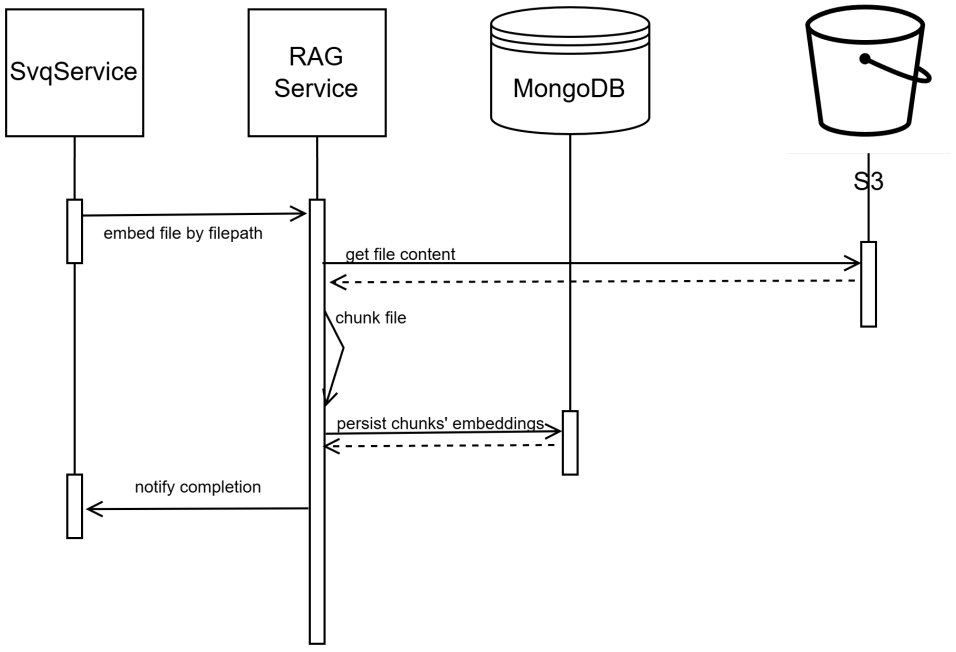
This sequence diagram illustrates how the user will be able to create new datasources. As evident, to create a datasource, the user must first be logged in and registered in the system. User only needs to specify the name of the datasource that they want to create. SVQ Service first creates a new datasource for the given user and then notifies the RAG Service to create a new collection for the datasource

→ File Upload in DataSource



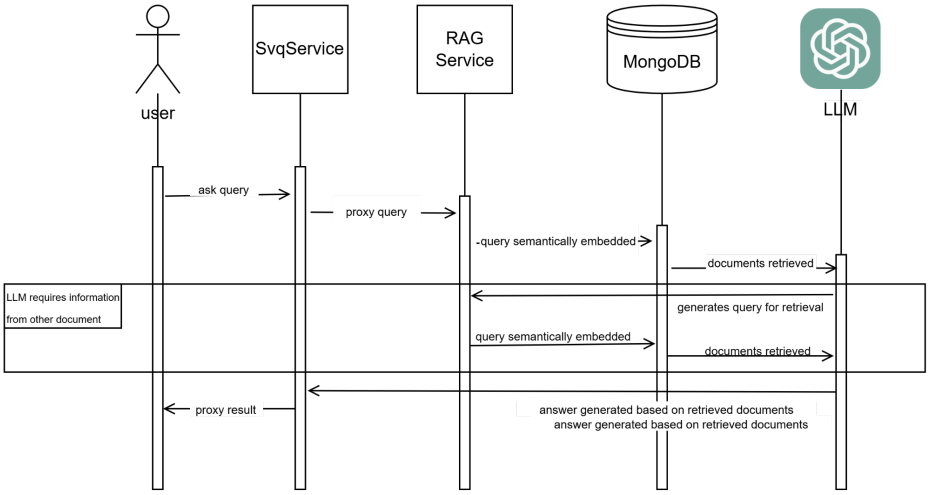
This figure illustrates how uploading of file works in the system. First a user selects the datasource that they want to upload and then uploads the file. This process also requires authenticated users. It can be seen clearly that the user does not have to wait for the embedding algorithm to complete for large files. Once SVQ Service saves the file to an object store, it notifies success to the user

→ File Embedding in the background



This sequence diagram shows how file embedding will run in the background. SVQ Servcie asks the RAG Service to download the file from a shared object store. Once the embedding completes, SVQ Service is notified via a webhook.

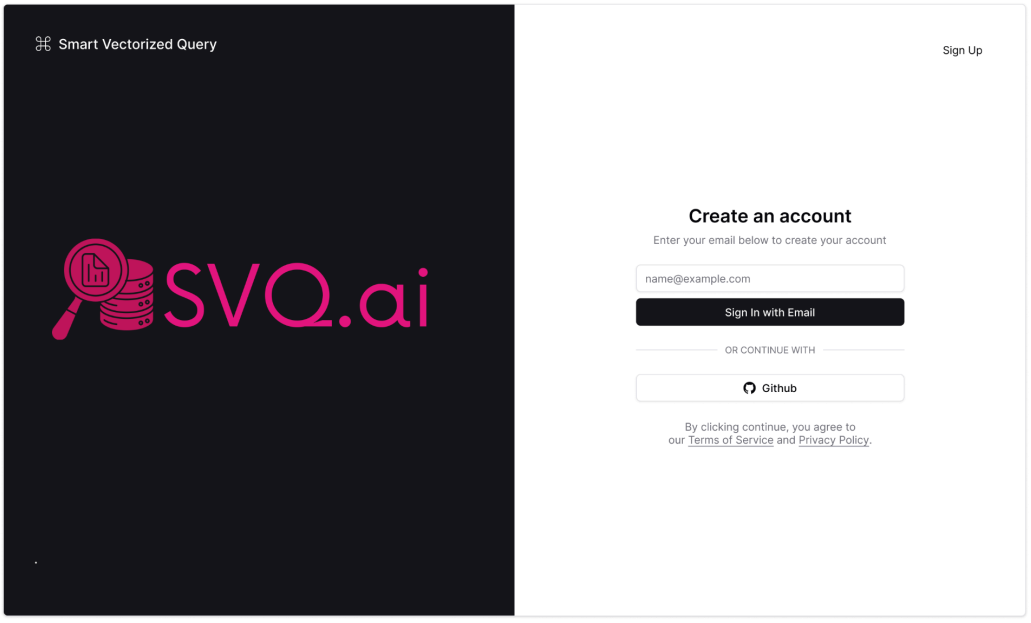
→ Querying Datasource



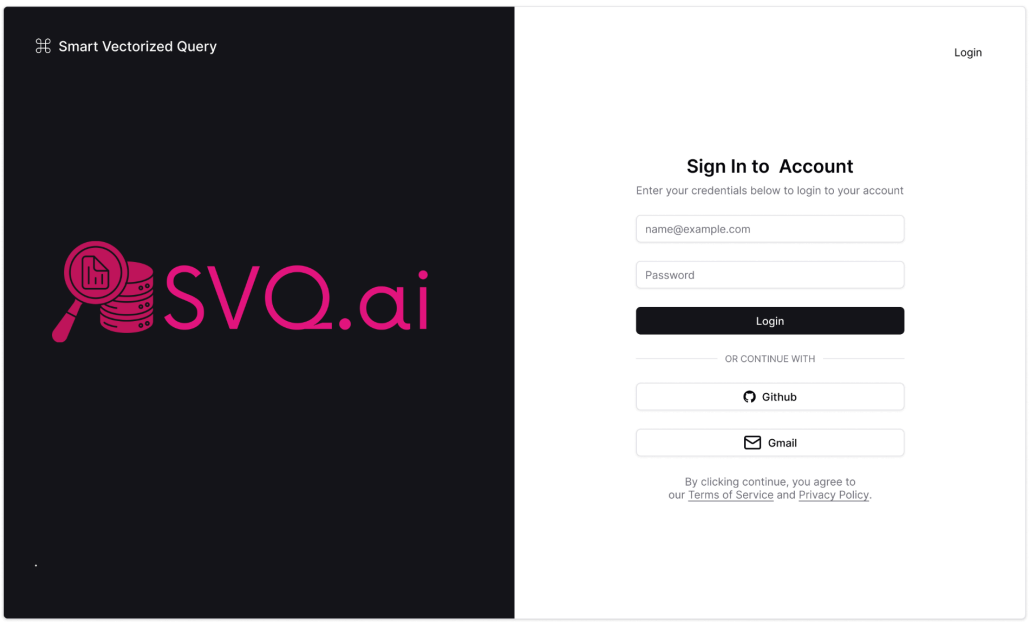
This sequence diagram shows how the datasource querying will be performed by the system. This activity is just an encapsulation of a standard RAG pipeline but our SVQ Service will act as a reverse proxy to the user, since all queries to a datasourcr need to be authenticated

2.5.5. User Interface

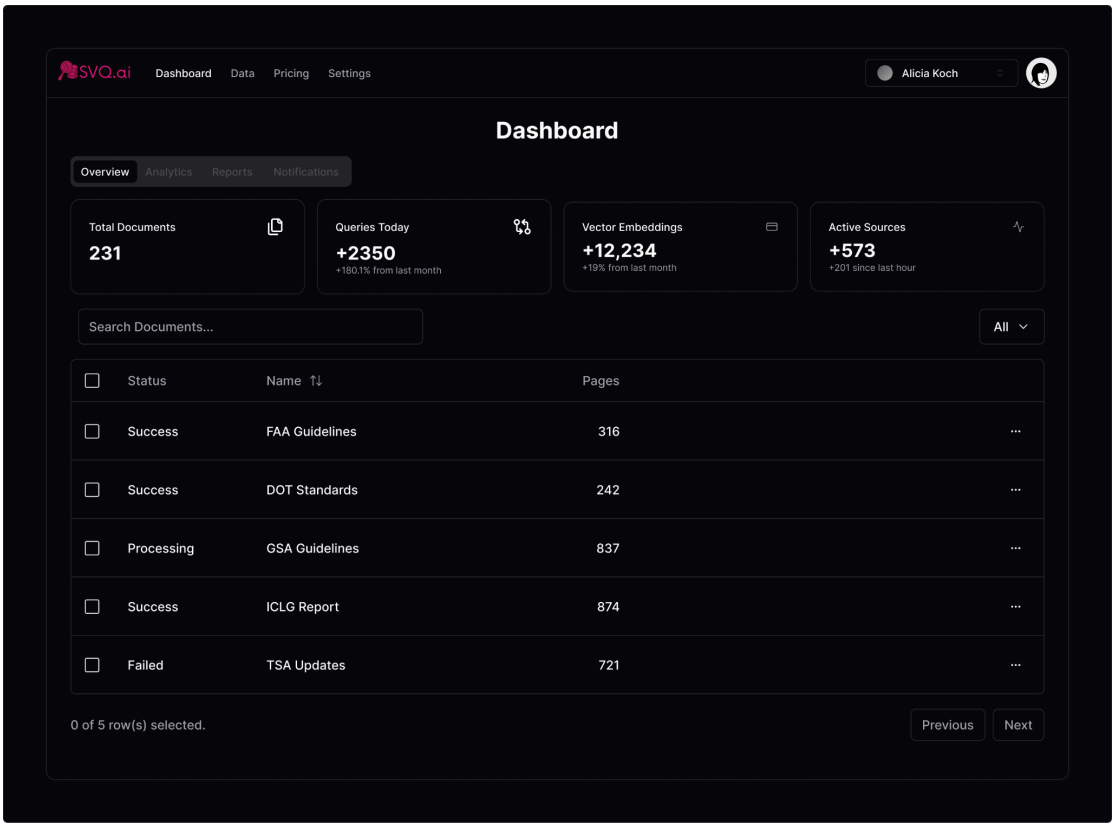
→ Create Account Page:



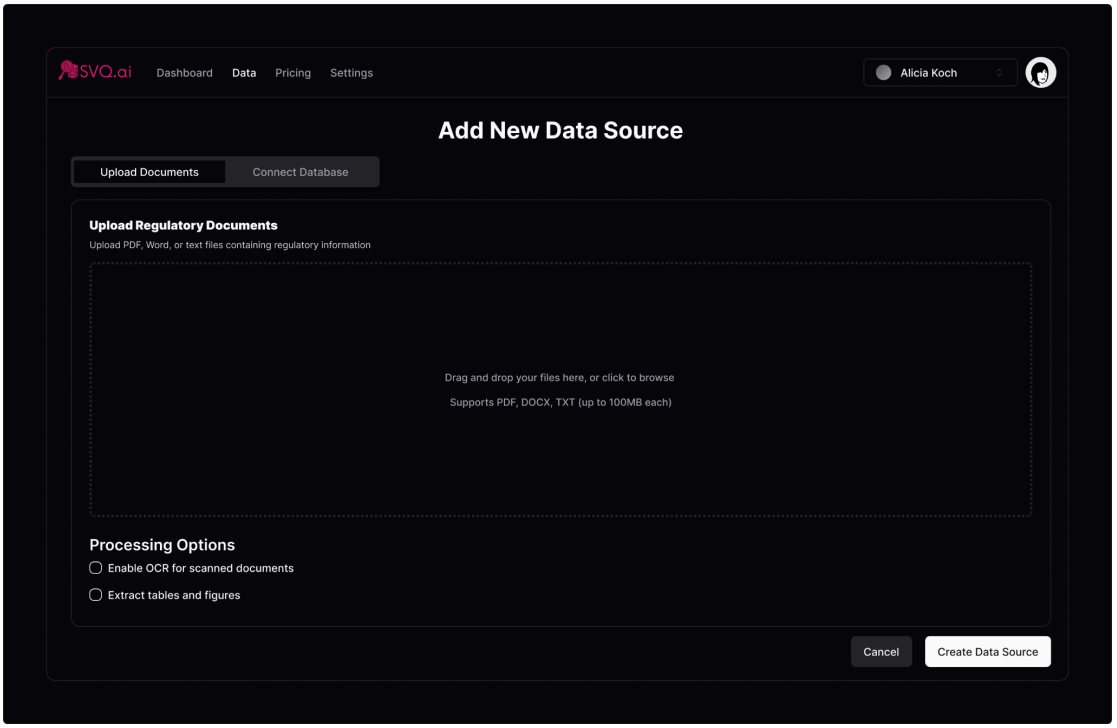
→ Login Page:



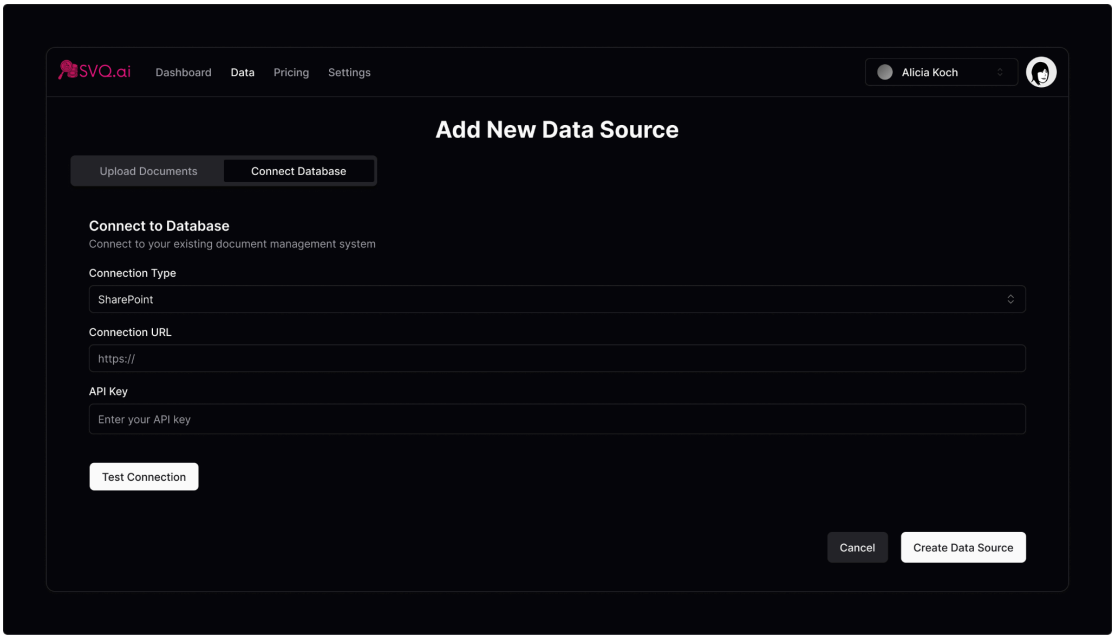
→ User Dashboard:



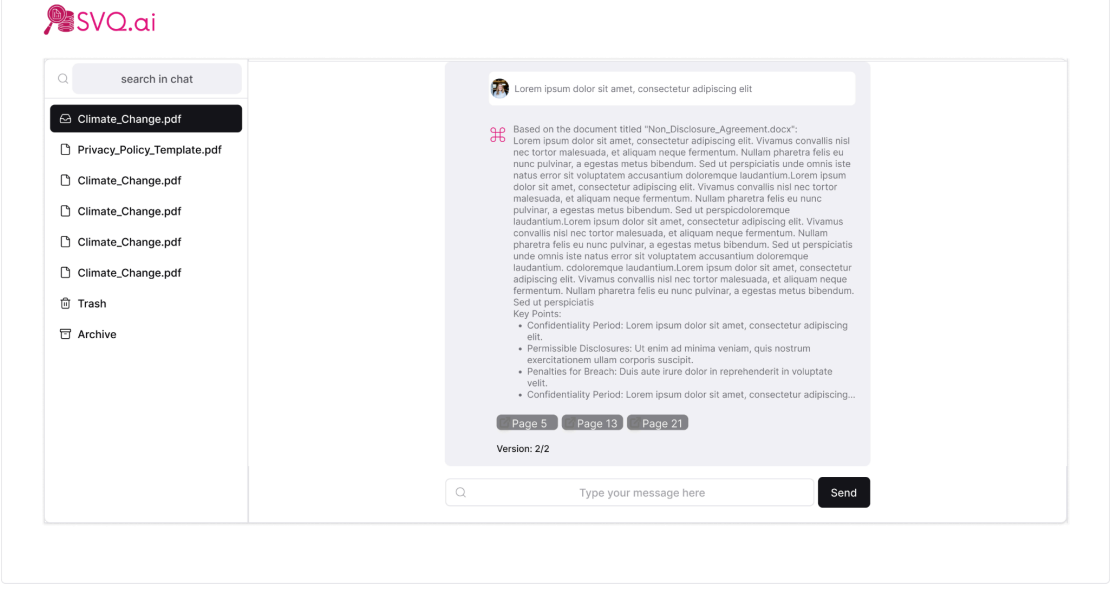
→ Create New Datasource:



→ Connect to Datasource:



→ Query Datasource Page:



3. Other Analysis Elements

3.1. Consideration of Various Factors in Engineering Design

In this section, many aspects that may affect Svq.ai will be discussed.

3.1.2. Constraints

→ Public Health:

There is no direct effect of public health that influences Svq.ai design

→ Public Safety:

Svq.ai internally uses a LLM. Hence, data integrity is a critical concern that affects Svq.ai's design. It must be ensured that the retrieval sources are from trusted and validated datasets to avoid propagating misinformation, especially if the system will address critical public safety topics such as emergency protocols or medical advice.

→ Public Welfare:

Accessibility of Svq.ai is one of our critical considerations. For accessibility in Svq.ai, we want to prioritize ensuring that the system is usable by diverse populations, including those with disabilities and limited technological resources. We will provide multi-modal interaction options, such as text, voice, and screen-reader compatibility, to accommodate users with visual, auditory, or physical impairments.

→ Global factors:

There is no direct impact of global factors because datasources will be private to the users.

→ Cultural factors:

For cultural factors inSvq.ai, we want to focus on **language diversity** and **regional adaptability**. Therefore, we will ensure that our system supports multiple languages and accounts for cultural nuances to provide accurate, context-sensitive responses.

→ Social factors:

Trustworthiness of the responses generated by Svq.ai are important. Henc,e for trustworthiness, we will ensure that our system provides transparent, explainable outputs by clearly indicating the sources of retrieved information and how they inform generated content.

→ Environmental factors:

Since Svq.ai is purely a software, there are no environmental considerations to be made

→ Economic factors:

One of the primary economic constraints is the storage of users' data. For each dataset that a user creates, a new database will have to be created. Given the size of each dataset and the number of datasets per user, the upkeep of the system can become very expensive. For this reason, it is crucial to operate efficiently in order to minimize the economic costs associated

with data storage. Additionally, the business model must be well thought, limiting users' storage capacity based on their subscription roles. Moreover, our reliance on LLMs as the central part of the project makes it very resource demanding. With a need for large amounts of memory and a powerful GPU and CPU, locally running our own LLM model will be very costly [1]. An alternative is to leverage existing LLM APIs such as OpenAI. Additionally, another fundamental aspect to our project is semantic embedding, which also requires powerful hardware, also making it costly [2]. Once again, we will need to rely on existing models that are accessible through API endpoints. These endpoints will be chosen through the efficacy and cost of the respective models.

Table 1: Factors that can affect analysis and design

	Effect level	Effect
Public health	0	No Effect
Public safety	8	Data integrity is critical in case Svq.ai is used for queries responses to urgent situations
Public welfare	6	Svq.ai should be accessible by people with disabilities and auditory impairment
Global factors	0	No Effect
Cultural factors	5	Svq.ai responses should support multiple languages
Social factors	8	Responses from Svq.ai should be trustworthy and referenced
Environmental factors	0	No Effect
Economic factors	10	Runnings inferences on LLMs is expensive. Svq.ai will explore ways to scale LLM inference while keeping costs low

3.1.3. Standards

During the development phase of our AI models, it is important to know whether optimizations are being made in the right direction. In order to clarify our direction of improvement, we have created an evaluation criteria and testbench that will allow us to standardize and quantify the evaluation of our models. The testbench consists of queries whose responses will be evaluated using the criteria. This will allow us to make direct comparisons between the different iterations of our system and see exactly where the differences and improvements lie. Moreover, having a quantifiable score for different aspects of the models allows us to better understand the weaknesses and areas of improvement. The evaluation criteria can be found in appendix A and the testbench can be found in appendix B.

Furthermore, the system architecture will adhere to the following standards, just to name a few:

1. OpenID Connect (OIDC): this protocol will be used for user authentication and authorization. Identity and Access management providers like Keycloak support this standard protocol

2. Advanced Message Queuing Protocol (AMQP): Message-oriented middlewares such as RabbitMQ are based on AMQP application layer protocol and will be used for distributed computation of chunks

3. Representational State Transfer (REST): client/server communication will take place adhering to the RESTful standard. Data exchange format will be JSON (except for files, which will uploaded as a multipart request)

4. Advanced Encryption Standard (AES): encryption/decryption of file contents by the user will require cryptographic hashing based off of symmetric keys

5. Best Matching 25 (BM25): for querying reports/analytics of a datasource, full-text search engines like Lucene will use this standard to return results of reports' queries

3.2. Risks and Alternatives

Table 2: Risks

	Likelihood	Effect on the project	B Plan Summary
Risk 1	Medium	Data Breaches or Unauthorized Access	Implement end-to-end encryption, robust authentication mechanisms, and regular security audits to safeguard data.
Risk 2	Low	Ingestion of Malicious or Corrupted Files	Validate and sanitize all uploaded files and implement file scanning tools to detect and block threats.
Risk 3	High	Hallucination or Misinformation in Queries	Enhance validation pipelines for responses and provide source attribution for transparency.
Risk 4	Medium	Scalability Challenges	Use scalable cloud infrastructure and implement rate-limiting to manage resource demands.

3.3. Project Plan

Table 3: List of work packages

WP#	Work package title	Leader	Members involved
WP1	Implement Figma Design Mockups	Muhammad	Ghulam Ahmed, Mehshid Atiq
WP2	Initialize Backend SVQ Service	Muhammad	Muhammad Rowaha
WP3	Initialize RAG Service	Zahaab	Zahaab Khawaja, Yassin Younus
WP4	Support Datasource Creation	Zahaab	Zahaab Khawaja, Yassin Younus, Muhammad Rowaha
WP5	Integrate UI and SVQ Service	Muhammad	All

WP 1: *Implement Figma Design Mockups*
Start date: *Nov' 24* **End date:** *Dec' 24*

Leader:	*Muhammad*	**Members involved:**	*Ghulam Ahmad, Mehshid Atiq*
Objectives: *Figma mockups will be converted into jsx components without integration of backend*			
Tasks: **Task 1.1 Initialization of Next.js App Router Repository** This repository will be solely responsible for working on the User interface of the app. Issues assigned to the frontend will be solved in this repository **Task 1.2 Deploy Project** Next.js app will be deployed on Vercel cloud and the app url will be made public **Task 1.3 Convert Figma Mockups** Figma mockups will be converted into jsx components and client-side routing of pages will be implemented			
Deliverables: **D1.1** Github Nextjs Repository **D1.2** Deployed Frontend **D1.3** Figma Mockups to UI			
WP 2: *Initialize Backend SVQ Service*			
Start date: *Nov' 24* **End date:** *Dec' 24*			
Leader:	*Muhammad*	**Members involved:**	*Muhammad Rowaha*
Objectives: SVQ Backend Service repository will be initialized with all dependencies			
Tasks: **Task 1.1 Initialization of FastAPI-based SVQ Backend Service Repository** This repository will be solely responsible for working on the SVQ Service of the app. Issues assigned to the backend will be solved in this repository **Task 1.2 Deploy Project** This service will be containerized and deployed to a cloud-provider, e.g. Railway **Task 1.3 Implement Auth Service** An access management service, e.g. Keycloak, will be integrated into the project and Restful endpoints will be exposed for creation and management of users			
Deliverables: **D1.1** Github FastAPI Repository **D1.2** Deployed SVQ Service **D1.3** Enabled Access Management for SVQ Service			
WP 3: *Initialize RAG Service*			
Start date: *Nov' 24* **End date:** *Jan' 25*			
Leader:	*Zahaab*	**Members involved:**	*Zahaab Khawaja, Yassin Younus*
Objectives: *RAG Service repository will be initialized with all dependencies*			
Tasks: **Task 1.1** Initialization of LangChain-based RAG Service Repository This repository will be solely responsible for working on the RAG Service of the app. Issues assigned to RAG will be solved in this repository **Task 1.2** Deploy Project This service will be containerized and deployed to a cloud-provider, e.g. Railway **Task 1.3** Integrate OpenAI Client OpenAI Client, via API key, will be integrated into the RAG service and will be used for generating embeddings and query results **Task 1.4** Integrate Vector Database			

A vector database, e.g. MongoDB, will be integrated to the RAG service for storage and retrieval of embeddings			
Deliverables: **D1.1** Github RAG Service Repository **D1.2** Deployed RAG Service **D1.3** Generation of embeddings and query results			
WP 4: *Support Datasource Creation*			
Start date: *Mid-Feb' 25* **End date:** *Apr' 25*			
Leader:	*Zahaab*	**Members involved:**	*Zahaab Khawaja, Yassin Younus, Muhammad Rowaha*
Objectives: *SVQ Service and RAG Service will interop for data-source creation*			
Tasks: **Task 1.1 Integrate S3 Object Store** S3 based Object Store will be integrate into the application. SVQ Service will expose endpoints to create datasources and upload files. RAG Service will download files from the S3 Store and generate embeddings **Task 1.2 Event-based InterOP** SVQ Service will expose webhooks to the RAG Service. SVQ Service will request RAG Service to begin embedding generation. RAG Service will notify success via exposed webhook			
Deliverables: **D1.1** S3 Object Store will be integrated **D1.2** SVG Service and RAG Service will interop			
WP 5: *Integrate UI and SVQ Services*			
Start date: *Feb' 25* **End date:** *Mid-May' 25*			
Leader:	*Muhammad*	**Members involved:**	*All*
Objectives: *SVQ Service and SVQ Web App will be integrated*			
Tasks: **Task 1.1 Mobx-Keystone Proxy Client-Side services** Frontend will develop mobx-based proxy (mirror) services to the SVQ Service. This will allow seamless integration of the backend into the web app			
Deliverables: **D1.1** SVQ Service and SVQ App will InterOP			

WP#	Work Package	Nov 24		Dec 24		Jan 25		Feb 25		Mar 25		Apr 25		May 25	
1	Implement Figma Mockups														
2	Initialize SVQ Service														
3	Initialize RAG Service														
4	Support DataSource creation														
5	Integrate UI and SVQ Service														

Table 4: Gantt charts for work packages

3.4. Ensuring Proper Teamwork

- All group members are required to participate in weekly meetings.
- Every member must actively contribute to the project development lifecycle and group decision-making process.
- Members are expected to research topics relevant to their assigned tasks and collaborate by providing support to each other, seeking guidance from the supervisor when necessary.
- Task assignments should align with individual interests and be distributed as fairly as possible.

3.5. Ethics and Professional Responsibilities

The source code will be privately accessible to group members and graders on GitHub. It should not be shared with any third parties until the project is completed.

- All software frameworks and libraries used must be acknowledged in accordance with their licensing requirements.
- Weekly group meetings with the team and the supervisor will be scheduled for Fridays, preferably in person.

3.6. Planning for New Knowledge and Learning Strategies

RAG models have recently become a prominent area of research, particularly with the rise of large language models (LLMs). In the past month alone, numerous research papers have explored the feasibility and potential optimizations of RAG models. Additionally, several RAG models, including Hugging Face's open-source model (https://huggingface.co/docs/transformers/en/model_doc/rag), are available for developers and academics.

During its development phase, Svq.ai will focus on incorporating the latest industry insights into its core model. Given the extensive fine-tuning required for the hundreds of parameters Svq.ai will utilize, the team may also develop innovative strategies for optimizing RAG models, thereby contributing to the expanding body of research in this area.

By prioritizing ongoing learning and knowledge acquisition, as well as a research-driven approach, Svq.ai aims not only to create a robust and competitive product but also to position itself as a leader in the field of AI-driven retrieval systems.

4. Glossary

Criterion	Sub-Criterion	Brief Description	Rating Scale (1-5)
Accuracy	Document Retrieval Accuracy	How well the model retrieves the most relevant regulatory documents or sections.	1: Irrelevant documents retrieved most of the time. 2: Limited relevance. 3: Moderately relevant. 4: Mostly relevant documents retrieved. 5: Highly relevant and precise retrieval.
	Fact-Checking	Ensures generated responses align with retrieved documents.	1: Responses are mostly incorrect. 2: Frequent factual errors. 3: Some factual errors, mostly correct. 4: Rare factual errors. 5: Fully accurate responses.
	Context Preservation	Maintains the original meaning and context of regulatory clauses.	1: Context is often lost or misinterpreted. 2: Frequent context errors. 3: Maintains context with some lapses. 4: Mostly accurate context. 5: Perfectly maintains context.

Comprehensiveness	Regulation Coverage	The model's ability to handle the entire scope of applicable regulations.	1: Major gaps in coverage. 2: Limited coverage. 3: Moderate coverage but with gaps. 4: Broad coverage with minor gaps. 5: Fully comprehensive.
	Handling Complex Queries	Handles nuanced, multi-part, or detailed regulatory questions effectively.	1: Fails to address complexity. 2: Struggles with complexity. 3: Handles some complexity but lacks precision. 4: Handles most complexities well. 5: Excels at addressing complex queries.
	Generic Query Handling	Ability to provide meaningful and accurate responses to broad or generic regulatory questions.	1: Provides vague or irrelevant responses. 2: Struggles to address generic questions. 3: Provides moderately useful responses but lacks depth. 4: Handles broad queries well with minor gaps. 5: Fully understands and addresses generic queries with insightful responses.

Relevance	Query Understanding	Ability to understand and interpret user queries accurately.	1: Often misinterprets queries. 2: Frequent misinterpretations. 3: Adequate understanding but lacks precision. 4: Strong understanding with rare issues. 5: Consistently accurate query interpretation.
	Pertinence of Outputs	Relevance of the retrieved documents and generated responses.	1: Responses are mostly irrelevant. 2: Frequently irrelevant responses. 3: Somewhat relevant. 4: Mostly relevant. 5: Highly relevant.
Legal Consistency	Compliance with Regulations	Ensures generated outputs align with the latest regulations.	1: Outputs are often non-compliant. 2: Frequent compliance issues. 3: Adequate compliance with occasional lapses. 4: Mostly compliant. 5: Fully compliant with regulations.
	Consistency Across Queries	Produces consistent results for similar or identical regulatory queries.	1: Inconsistent across queries. 2: Frequently inconsistent. 3: Some inconsistencies but generally acceptable. 4: Mostly consistent. 5: Fully consistent.

Memory	Context Retention	The model's ability to retain information provided earlier in a conversation.	1: Forgets context frequently. 2: Retains context inconsistently. 3: Retains context moderately well but may miss key details. 4: Retains context with minor lapses. 5: Excellent retention of context across multi-turn conversations.
	Memory Accuracy	How accurately the model remembers and applies past context to new queries.	1: Frequently recalls inaccurate or irrelevant information. 2: Often applies context incorrectly. 3: Moderately accurate recall. 4: Accurate recall with rare issues. 5: Consistently accurate and contextually relevant recall.

5. References

[1] Manuel, “Memory Requirement for LLM Training and Inference,” 2024. [Online]. Available on: https://medium.com/@manuelescobar-dev/memory-requirements-for-llm-training-and-inference-97e4ab08091b

[2] Szymon Palucha, “Running a SOTA 7B Parameter Embedding Model on a Single GPU,” 2024. [Online]. Available on: https://towardsdatascience.com/running-a-sota-7b-parameter-embedding-model-on-a-single-gpu-bb9b071e2238