Bilkent University
Department of Computer Engineering

Senior Design Project
T2431
Smart Vector Query

Final Report

Mehshid Atiq 22101335

Muhammad Rowaha 22101023

Ghulam Ahmed 22101001

Yassin Younis 22101310

Zahaab Khawaja 22101038

Supervisor: Uğur Doğrusöz

Innovation Expert: John Yuzdepski

04.03.2025
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

Final Report

SVQ: Smart Vector Query

1. Introduction

1.1. Purpose of the system

SVQ.ai is an advanced platform designed to help professionals navigate complex regulatory documents with ease. It utilizes state-of-the-art Retrieval-Augmented Generation (RAG) [1] technology, cutting-edge large language models (LLMs), and modern vector databases [2] to enhance document retrieval. Users can upload various document types, including regulatory texts, codebases, and textbooks, which are then processed semantically for more effective searches. Through sophisticated chunking and embedding techniques, SVQ.ai enables seamless querying of large files via an intuitive chatbot interface, delivering precise and context-aware responses. Prioritizing efficiency, accuracy, user experience, data privacy, and scalability, the platform is an essential tool for professionals managing regulatory information. SVQ.ai targets professionals, compliance officers, and researchers handling regulatory documents.

1.2. Design Goals

1.2.1. Accuracy

One of the primary goals of SVQ.ai is to deliver accurate and contextually relevant responses. Accuracy is ensured through precise semantic processing, robust chunking and embedding techniques, and intelligent query handling. The platform is designed to minimize misinformation, reduce hallucinations, and provide factually correct outputs that align with the original document's intent. Indeed, SVQ.ai aims to be a reliable tool for professionals dealing with complex regulatory documents, ensuring confidence in every response.

1.2.2. Reliability

SVQ.ai aims to increase reliability by introducing query responses, generated via RAG, with back references to the content. Via transparency and explainability, the system will be able to explain why a particular response was generated by linking it back to the exact sections of the retrieved text. Back-referencing also ensures that the system reliably handles ambiguity; instead of making false assumptions, the system should acknowledge when the available documents do not fully answer a query and provide a relevant response (e.g. a disclaimer). Since the

retrieved text is only loosely related to the query, SVQ.ai aims to extract response without omitting full context and minimizing misinterpretation.

1.2.3. Security

SVQ.ai is designed with security at the forefront. To minimize operational costs, SVQ.ai aims to maintain a shared database server (or a cluster) for all its users. However, it is clearly not a safe approach considering users might be uploading sensitive data. To counter this and maintain a balance between security and costs, SVQ.ai will follow a database-per-user approach (multi-tenant architecture). For the object stores maintaining users' files, access to bucket objects will need to be authorized. Objects will be stored in an encrypted format, and can only be decrypted with the user's key.

1.2.4. Supportability

To ensure supportability and improved user onboarding experience, the system includes user manuals updated with each release to guide users on app usage and features. Maintenance checks will also occur regularly, with seamless, backward-compatible updates to minimize disruption. Furthermore, users will be notified 1 day prior to scheduled updates; nevertheless, downtime will be kept minimal. The repository will maintain a comprehensive documentation for developers and administrators, and will cover architecture, APIs, and troubleshooting. A robust support system with FAQs, and a knowledge base will also ensure timely issue resolution and user satisfaction.

1.2.5. Marketability

For SVQ.ai, marketability is another key design goal, particularly given its role as a tool for professionals navigating complex regulatory documents. Following a B2C model, user adoption and trust are critical to its success; SVQ.ai aims to attract a broad audience, including legal professionals, compliance officers, researchers, and businesses seeking efficient document comprehension. By focusing on marketability, SVQ.ai prioritizes a seamless user experience, intuitive interactions, and high-value insights that encourage continued engagement. Ensuring accuracy, reliability, and ease of use will be essential in driving adoption, fostering trust, and maintaining a competitive edge in the regulatory technology market.

1.2.6. Scalability

The core workflows powering SVQ.ai can be performance intensive and introduce unnecessary latency overheads. Hence, it would have been futile to try to accommodate multiple users on the SVQ.ai server. To counter such unwelcoming circumstances, SVQ.ai embeds state-of-the-art workflow

orchestration built on top of Hatchet [3]. The performance intensive and time-consuming tasks, such as embedding generation, parsing and chunking, are all handled by Hatchet. This not only gives our users a coherent experience but also provides opportunistic scalability.

1.3. Overview

SVQ.ai is an advanced LLM-driven platform designed to streamline the process of navigating complex regulatory, legal, and technical documents. Leveraging state-of-the-art RAG technology, LLMs, and modern vector databases, SVQ.ai provides users with a seamless and intelligent way to access precise, context-aware information.

One of SVQ.ai's core strengths is its advanced semantic processing. By analyzing user-uploaded documents, the platform enables highly efficient and accurate information retrieval. Through intelligent chunking and embedding techniques, SVQ.ai ensures that users receive targeted responses to their queries, reducing the time spent searching through vast amounts of information.

SVQ.ai also features an interactive chatbot interface, allowing users to engage in dynamic, real-time conversations to extract key insights from their documents. The system provides back-referenced answers, ensuring transparency and traceability by linking responses directly to the source text. This enhances reliability and helps users validate the retrieved information.

Another feature provided by SVQ.ai is the ability for the user to group documents into a single data source–namely, Query Datasource (QD). When responding to a user query, the system makes use of all such related documents to answer with more contextual awareness, including critical information and reduced misinterpretations.

To further enhance security, SVQ.ai encrypts users' documents with a symmetric encryption algorithm–AES. This allows the user to upload and query sensitive data with strong security guarantees. Furthermore, implementing a database-per-user approach in SVQ.ai enhances security by isolating each user's data, including chat history, in separate databases. This minimizes unauthorized access risks, as users can only access their own data. Additionally, applying strict access controls simplifies compliance with data privacy regulations, further strengthening the platform's security.

2. Requirements Details

2.1. 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.

2.1.1. 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.

2.1.2. 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.

2.1.3. 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.

2.1.4. 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.

2.1.5. 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.

2.1.6. 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.

2.1.7. 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.

2.1.8. 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.

2.2. Non-Functional Requirements

The following non-functional requirements articulate the essential quality attributes and operational constraints that underpin the svq.ai platform's effectiveness. Complementing the functional requirements—which delineate the system's intended behavior and capabilities—these criteria establish rigorous benchmarks for usability, reliability, performance, security, and maintainability, thereby ensuring that the platform satisfies both user expectations and organizational standards under production conditions.

2.2.1. 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.

2.2.2. 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 [4] 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.

2.2.3. 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.

2.2.4. 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.

2.2.5. 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.

3. Final Architecture and Design Details

3.1. Overview

SVQ.ai leverages a scalable, distributed architecture to deliver its LLM-powered queries on unstructured documents. The system adheres to a classic client-server model; yet, the server orchestrates user queries by sending messages to a RAG service (acts as a forward proxy for user queries). A responsive user interface is linked with a robust, secure backend over a RESTful API and Websockets for real-time use cases like chatting.

The front-end of SVQ.ai is a web-based application and it relies on React + Typescript as its foundation; this ensures a seamless user experience across various devices, ranging from desktop to mobile platforms. To meet the required expectations of the product, it leverages industrial-standard component libraries like ShadCN to deliver a coherent yet user-friendly experience. SVQ.ai aims to keep bundle sizes minimal and leverages advanced techniques like lazy-loading to reduce TTI.

The back-end of SVQ.ai, although logically a monolithic server, is implemented on a serverless architecture. Each serverless function will be responsible for a specific task, such as managing QD, uploading documents, querying documents. For data persistence, a multi-tenant architecture has been used–powered by Turso–with the aim of achieving data isolation and improved security.

To ensure further data security and user authentication, SVQ.ai integrates Keycloak server–an OIDC-compliant authentication service. Keycloak allows SVQ.ai to maintain industrially approved standards for IAM. Each serverless function will be authenticating user requests with this Keycloak server; hence, the front-end first authenticates with the Keycloak server and uses this token to make requests to the back-end.

The RAG (Retrieval-Augmented Generation) subsystem consists of two main flows: embedding and response generation. In the embedding flow, input documents are chunked and processed using an embedding model to extract entities and relationships, which are deduplicated, enriched using an LLM, and stored in a vector database and knowledge graph. The response generation flow begins when a query is received; relevant entities, relations, and text units are retrieved using embeddings and keyword extraction. A structured context is created using system prompts and combined with the query to generate a response via an LLM. This ensures accurate, context-aware answers by leveraging stored knowledge efficiently.

This architecture offers a scalable and reliable foundation for SVQ.ai, supporting core functionalities while allowing future expansion and integration of new features, ensuring continuous growth and adaptation to user needs.

3.2. Subsystem decomposition

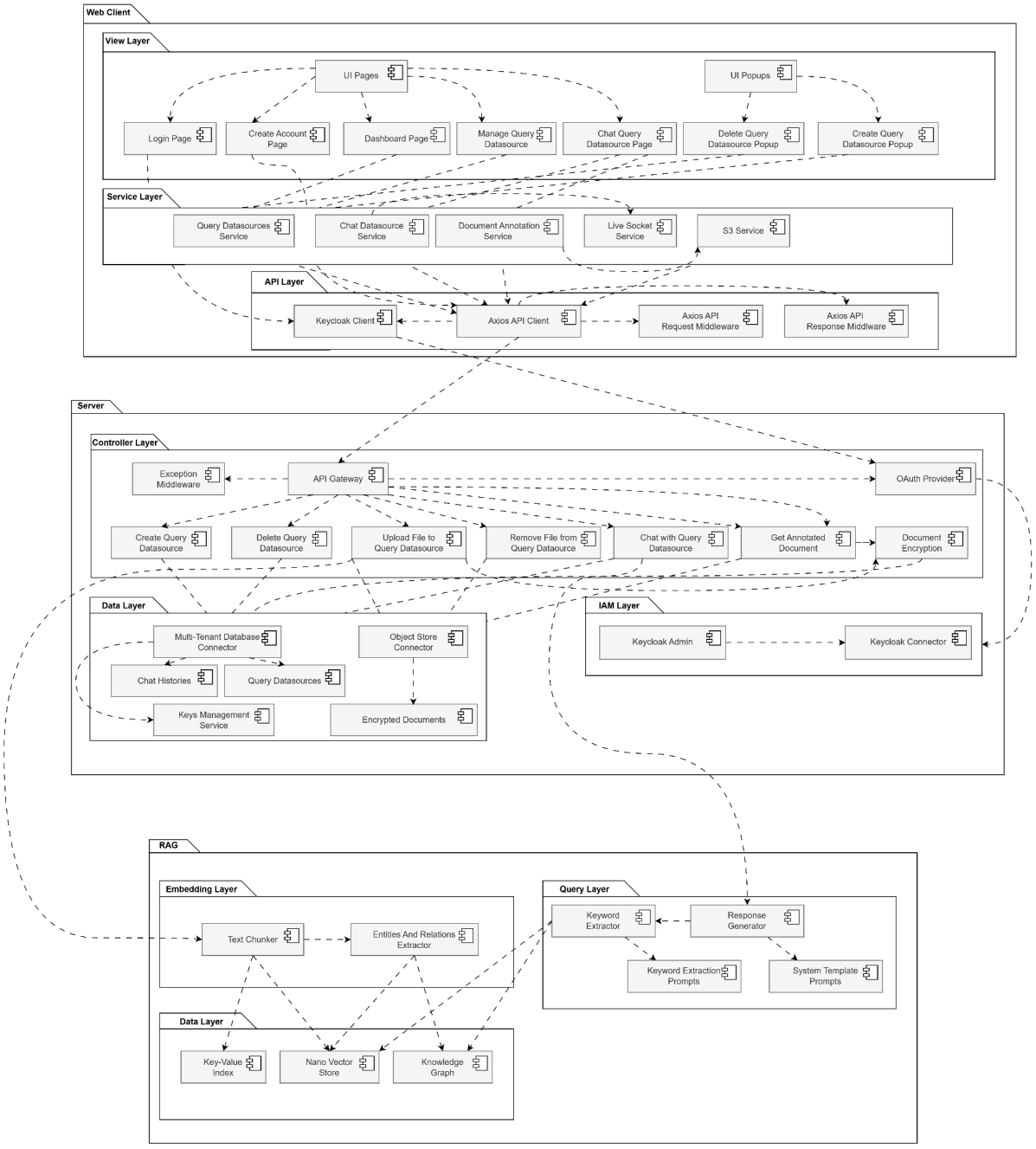

Figure 1: Subsystem Decomposition

3.3. Persistent data management

SVQ.ai leverages **Turso**, a distributed SQLite database, to efficiently implement its multi-tenant architecture, ensuring seamless and secure data isolation for different users. By utilizing Turso's edge-hosted, low-latency databases, SVQ.ai can dynamically allocate storage and computational resources per tenant, enabling scalable and efficient handling of requests. With built-in access controls and tenant-level data partitioning, SVQ.ai maintains strict data privacy and security compliance, reinforcing its role as a trusted tool for protecting users' sensitive documents.

Nano serves as a vector database in the RAG subsystem, storing and retrieving high-dimensional embeddings of document chunks, entities, and relations. During querying, it enables fast similarity searches [5], retrieving the most relevant entities, relations, and text for context-aware response generation.

3.4. Access control and security

SVQ.ai integrates **Keycloak** [6] to handle access control and authentication for Lambda functions within its system. As an IAM solution, Keycloak enables secure authentication, ensuring that only authorized users and services can interact with SVQ.ai's resources. SVQ.ai configures Keycloak to issue JWT tokens that verify identities. When a Lambda function is triggered whether for document processing, retrieval, or RAG-powered responses, it must present a valid Keycloak-issued token. The platform then validates this token against Keycloak's authorization server, ensuring that the request originates from an authenticated and authorized source. With automatic token refresh, federated identity support, and logging features, Keycloak enhances security, simplifies user management, and provides detailed audit trails for monitoring Lambda executions and user interactions.

SVQ.ai employs **AES** to securely encrypt users' uploaded documents, ensuring data confidentiality both at rest and in transit. Before storing documents in the system, SVQ.ai encrypts them using AES-256 [7], a robust symmetric encryption algorithm widely recognized for its high security and efficiency. Each tenant's data is encrypted with unique keys, preventing unauthorized access even if storage is compromised. By leveraging AES encryption, SVQ.ai safeguards sensitive regulatory documents from unauthorized exposure while maintaining fast and efficient retrieval when authorized users query the system.

3.5. Standards

- **OpenID Connect (OIDC):** this protocol will be used for user authentication and authorization. Identity and Access management providers like Keycloak support this standard protocol
- **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)
- **Advanced Encryption Standard (AES):** encryption/decryption of file contents by the user will require cryptographic hashing based off of symmetric keys
- **Unified Modeling Language (UML):** utilized for modeling the system architecture, such as subsystem decompositions and class diagrams. This standard helps in visualizing the structure of the project.

4. Development/Implementation Details

4.1. Hatchet Workflow Orchestration

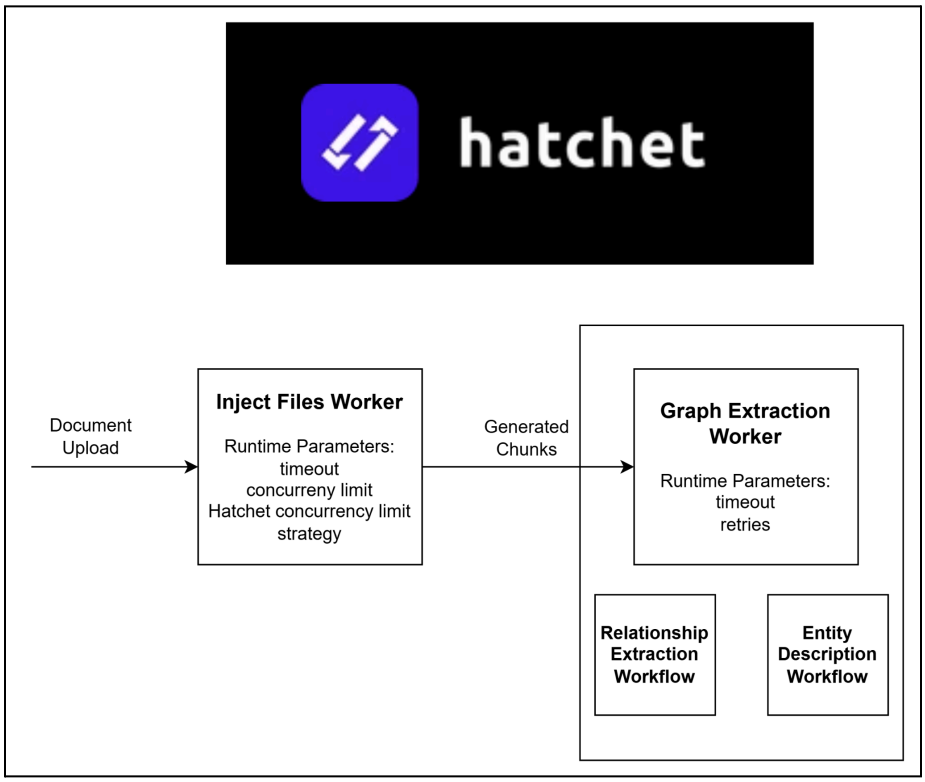

Figure 2: Hatchet Workflow Orchestration

4.2. Web Client

4.2.1. View Layer

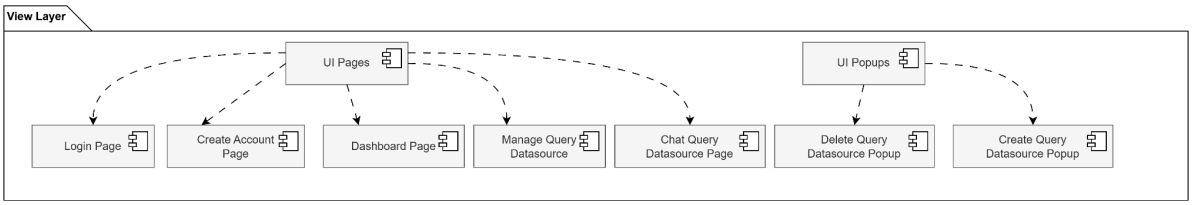

Figure 3: View Layer for Web Client

This View Layer Decomposition diagram represents the user interface (UI) structure of SVQ.ai, illustrating how different pages and popups interact.

1. UI Pages:
    - The Login Page and Create Account Page allow users to authenticate and register on SVQ.ai.
    - The Dashboard Page serves as the central hub, where users can perform actions on their query datasources.
    - The Manage Query Datasource Page enables users to organize and maintain the documents in their Query Datasources
    - The Chat Query Datasource Page provides an interface where users can interact with the SVQ.ai query pipeline to retrieve information from their selected datasource.
2. UI Popups:
    - The Delete Query Datasource Popup allows users to confirm the deletion of a datasource, ensuring they don't accidentally lose important documents.
    - The Create Query Datasource Popup enables users to add new datasources by uploading documents or configuring access to external sources.
3. Implementation:
   The frontend was bootstraped with a Next.js Pages Router application, and PNPM as the package manager. TailwindCSS + ShadCN UI were used for implementing the Figma designs.

4.2.2. Service Layer

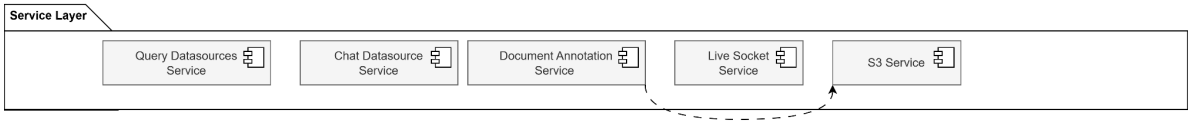

Figure 4: Service Layer for Web Client

This Service Layer Decomposition diagram represents the structure of web client global application state. The services are implemented using Mobx-Keystone models, and each service is a singleton that can be referenced from anywhere inside the application.

1. Query Datasources Service:
    - Provides functionalities to interact with the query datasources on the client level.
    - Abstracts the underlying network API endpoints and provides a seamless integration of CRUD operations on query datasources

2. Chat Datasource Service:
    - Using the Live Socket Service, enables the consumers to interact with the SVQ.ai query pipeline for real-time responses using Websockets.
    - Hides the underlying complexity of real-time communication by providing a reactive observables for consumers to react to
3. Document Annotation Service:
    - Provides the consumers with the functionality of annotating certain parts of a selected document and returns the blob objects for rendering
4. Live Socket Service:
    - maintains an open Websocket connection with the SVQ.ai servers for the entire lifetime of the application state
5. S3 Service:
    - Provides abstraction of managing documents for a given query datasource
6. Implementation:
   The service layer for the SVQ.ai app is implemented with Mobx-Keystone that provides the app with necessary reactivity e.g. Websocket events.

4.2.3. API Layer

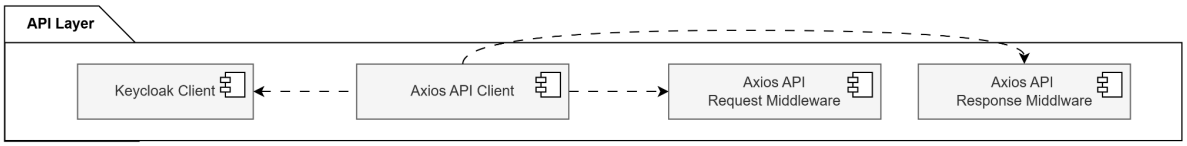

Figure 5: API Layer for Web Client

This API Layer Decomposition diagram represents how SVQ.ai handles API interactions, particularly authentication and request handling.

1. Keycloak Client:
    - Handles authentication and authorization using Keycloak, ensuring secure user access.
    - It interacts with the Axios API Client to attach authentication tokens to outgoing requests.
2. Axios API Client:
    - Acts as the primary interface for making HTTP requests to SVQ.ai's backend services.
    - It integrates with Keycloak to include authentication headers in API requests.

3. Axios API Request Middleware:
    - Intercepts outgoing API requests to modify or validate them before they reach the server.
    - Typically used for adding headers, logging requests, or handling retries.
4. Axios API Response Middleware:
    - Processes responses from the server before they reach the frontend.
    - Used for handling errors, refreshing tokens, or transforming data formats.
5. Implementation:
   The api layer of SVQ.ai consists of integration of a RESTful API and Websocket events. For integrating the RESTful API, a global Axios instance was exposed to all services. For Websocket integration, Stomp.js client was used.

4.3. Server

4.3.1. Controller Layer

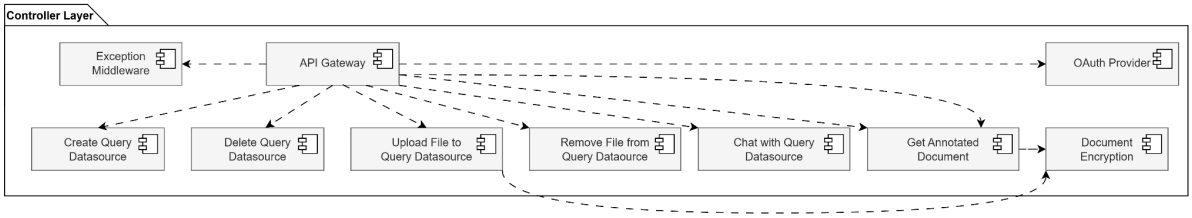

Figure 6: Controller Layer for Server

This Controller Layer Decomposition diagram represents how SVQ.ai manages core functionalities through an API gateway, handling various operations related to QD and security.

1. API Gateway:
    - Central entry point for all client requests.
    - Routes requests to the appropriate controllers for processing.
    - Ensures authentication and authorization by interacting with the OAuth Provider.
2. Exception Middleware:
    - Captures and handles errors, ensuring stable API behavior
    - Adheres to standard practices for errors such as HTTP status codes and structured error responses
3. Query Datasource Management:
    - Create Query Datasource: Initializes a new collection of files for querying.
    - Upload File to Query Datasource: Adds files to an existing datasource.
    - Remove File from Query Datasource: Deletes specific files from a datasource.
    - Delete Query Datasource: Removes an entire datasource when no longer needed, and batch deletes files
4. Chat & Document Processing:
    - Chat with Query Datasource: Allows users to interact with the RAG-powered chatbot.
    - Get Annotated Document: Retrieves processed or annotated documents based on user queries.
    - Document Encryption: Ensures secure storage and retrieval of documents.
5. Security & Authentication:
    - The OAuth Provider handles user authentication and authorization.

- Document Encryption protects sensitive data before storage or transmission.

6. Implementation:
   The controller layer of SVQ.ai is implemented with FastAPI running inside Uvicorn web server. FastAPI exposes RESTful and Websocket endpoints to the application

4.3.2. Data Layer

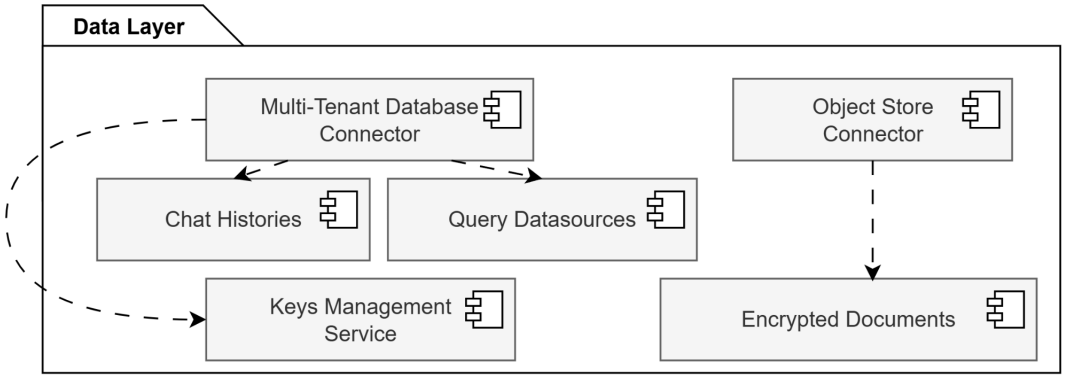

Figure 7: Data Layer for Server

This Data Layer Decomposition is a critical component of the SVQ.ai platform, responsible for handling data storage, retrieval, encryption, and secure access across multiple tenants.

1. Multi-Tenant Database Connector
   - Acts as a central interface for managing database connections across multiple users or organizations.
   - Ensures data isolation and access control for different tenants using secure authentication and authorization mechanisms.
2. Chat Histories
   - Stores past user interactions, allowing for personalized experiences and context-aware query handling.
   - Enables users to retrieve previous conversations, improving response accuracy and reducing redundant queries.
3. Query Datasources
   - Stores metadata about query datasources for a given user
4. Keys Management Service
   - Oversees encryption key storage, rotation, and security policies.

- Ensures sensitive data, such as chat histories and encrypted documents, remains protected.
- Works in tandem with other components to enforce secure access to confidential information.

5. Object Store Connector
    - Acts as an interface between the platform and external object storage services (e.g., AWS S3, Google Cloud Storage).
    - Enables the retrieval and storage of large documents while maintaining scalability.
    - Supports seamless integration with Encrypted Documents for secure data handling.
6. Encrypted Documents
    - Stores uploaded documents in a secure, encrypted format.
    - Ensures compliance with data privacy regulations by preventing unauthorized access.

4.3.3. IAM Layer

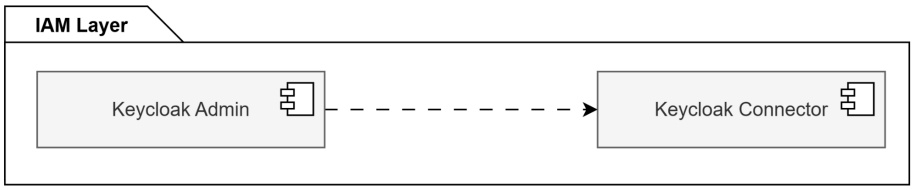

Figure 8: IAM Layer for Server

The IAM (**Identity and Access Management)** Layer is responsible for handling authentication, authorization, and user management within the SVQ.ai platform. It ensures secure access to resources by managing user identities, roles, and permissions. This layer integrates with **Keycloak**, an open-source identity and access management solution, to provide centralized authentication.

1. Keycloak Admin
    - Serves as the administrative interface for managing users, roles, permissions, and authentication policies.
    - Provides tools for configuring identity providers, user federation, and session management.
    - Works with the **Keycloak Connector** to enforce authentication and authorization rules across the platform.

2. Keycloak Connector
    - Acts as an intermediary between the Keycloak Admin and the application's authentication system.
    - Facilitates secure communication for user login and token validation.
    - Ensures seamless integration with other services by providing identity federation and access token management.

4.4. RAG

4.4.1. Embedding Layer

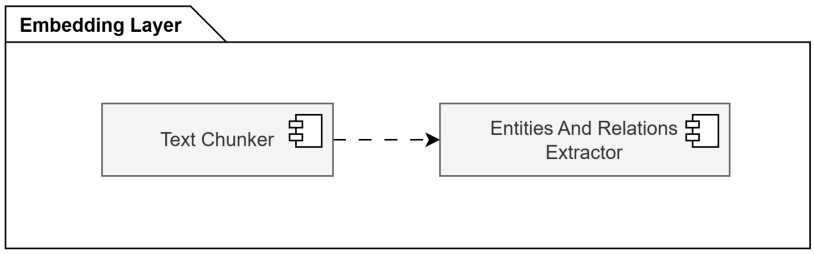

Figure 9: Embedding Layer for RAG

This subsystem module is responsible for processing input documents, extracting entities and relationships, and storing them in structured formats for efficient retrieval. Document chunks are indexed using a Hierarchical Navigable Small World (HNSW) graph for ultra-fast nearest-neighbor lookup [9].

1. Text Chunker
    - Text Chunker splits documents into smaller chunks.
    - Embedding Generator uses an embedding model (EB Model) to create vector representations of text.
2. Entity & Relation Extraction Module
    - Entities & Relations Extractor extracts named entities, relationships, and relevant metadata.
    - Deduplication ensures uniqueness of entities and relations using a "Set" deduplication approach.

4.4.2. Query Layer

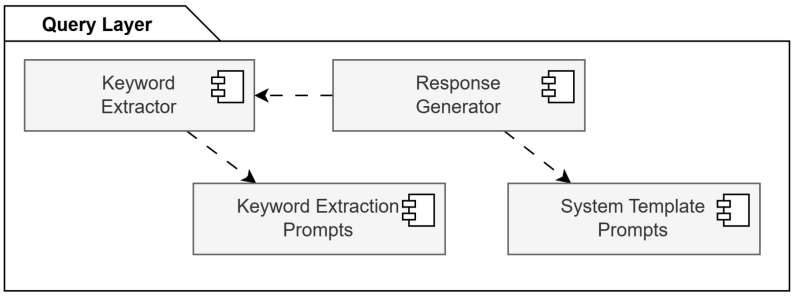

Figure 10: Query Layer for RAG

This subsystem decomposition handles user queries by retrieving relevant context from the stored embeddings and knowledge graph, then generating responses using a language model.

1. Keyword Extractor
    - Uses Keyword Extraction Prompt to extract relevant keywords from the user query.
    - The extracted keywords are then passed to the Response Generator for further processing.
2. Response Generator
    - Takes extracted keywords and uses System Template Prompts to construct a meaningful response.
    - Generates the final response based on the structured information received.

4.4.3. Data (Knowledge) Layer

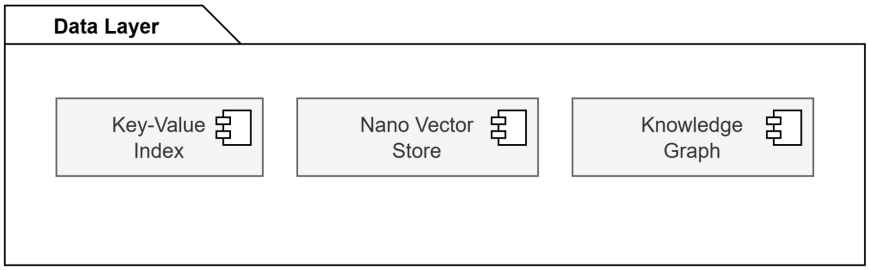

Figure 11: Data Layer for RAG

The subsystem decomposition serves core components connecting both embedding and response generation flows for efficient retrieval and reasoning.

1. Key-Value Index Storage: Stores extracted structured data in a KV storage.

2. Vector Database Storage: Stores vector embeddings in a vector DB for fast retrieval.
3. Knowledge Graph Storage: Stores deduplicated entities and relationships in a structured knowledge graph.

5. Testing Details

5.1. Test Case Code Conventions

The following Context-Free Grammar notation illustrates the test code convention:

$$< test\_code > ::= < test\_type > < test\_number > < iteration >$$
$$< test\_type > ::= 1 \mid 2$$
$$< test\_number > ::= < digit > < digit >$$
$$< iteration > ::= < digit > \mid < iteration > < digit >$$
$$< digit > ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

For example:
10101: refers to the first iteration of the first test case of the test category of type one.

5.1.1. Test Types

Test Type	Test Category Name
1	Functional Testing
2	Integration Testing
3	User Acceptance Testing
4	Performance Testing
5	Security Testing
6	Usability Testing
7	Compatibility Testing

5.2. Test Cases

Test ID: 10101

- **Type:** Functional Testing
- **Objective:** Create Account and Login
- **Steps:**
    - Click on ‘Create Account’ button on Login Page

    - Redirect to the Create Account page
    - Enter unique username and password
    - Notify successful account registration
    - Redirect to Login page
    - Enter registered username and password
    - Redirect to Dashboard page
- **Test Results:** User authentication flow worked successfully, and could access protected routes successfully
- **Priority:** Critical
- **Status:** Passed

Test ID: 10201

- **Type:** Functional Testing
- **Objective:** Verify Back-Referencing of Documents with Annotations
- **Steps:**
    - Upload a document to a Query Datasource in SVQ.ai
    - Query the document using the chat interface
    - Receive a response with referenced document sections
    - Verify back-references by clicking on citations or highlighted text
    - Ensure correct navigation to the referenced document section
- **Test Results:**
    - Responses included accurate back-references to the document
- **Priority:** Critical
- **Status:** Passed

Test ID: 10301

- **Type:** Functional Testing
- **Objective:** Verify the Ability to Maintain and View Chat History for a Given Query Datasource
- **Steps:**
    - Select a Query Datasource from the dashboard
    - Start a chat session and enter multiple queries
    - Receive responses and ensure they are correctly displayed in the chat interface
    - Close the chat session and navigate away from the page
    - Reopen the same Query Datasource's chat session and verify that previous conversations are retained
    - Scroll through chat history to check if messages are correctly loaded
    - Test session persistence by logging out and logging back in, then accessing the same Query Datasource
    - Ensure timestamps and user messages are correctly associated with the chat history

- **Test Results:**
    - Chat history is maintained and correctly linked to the Query Datasource
    - Users can view past conversations even after logging out and back in
- **Priority:** Major
- **Status:** Passed

Test ID: 10401

- **Type:** Functional Testing
- **Objective:** Verify Handling of Unsupported File Formats
- **Steps:**
    - Log in to SVQ.ai with valid credentials.
    - Navigate to the "Manage Query Datasource" page for an existing QD.
    - Attempt to upload an unsupported file type (e.g., .exe file).
    - Observe any system notifications or error messages.
- **Test Results:**
    - System rejects unsupported file types, displaying a clear error message.
    - No chunking or embedding process is triggered.
    - The QD remains unchanged.
- **Priority:** Minor
- **Status:** Passed

Test ID: 10501

- **Type:** Functional Testing
- **Objective:** Validate Document Deletion Flow
- **Steps:**
    - Log in and open an existing Query Datasource containing multiple documents.
    - Select one document and click "Remove File."
    - Confirm deletion when prompted.
    - Reload the page or navigate away and return.

- **Test Results:**
    - The selected document is permanently removed from the QD.
    - Any associated embeddings and encrypted object storage entries are deleted.
    - The user is notified that the file was successfully removed.
- **Priority:** Major
- **Status:** Passed

Test ID: 10601

- **Type:** Functional Testing
- **Objective:** Check Multi-Document Query Handling
- **Steps:**
    - Log in and create or open a QD with **two or more** documents uploaded.
    - Enter a chat query referencing content from multiple documents (e.g., "Compare Section 2 from Doc A with Chapter 4 from Doc B").
    - Observe the system's combined response.
    - Verify the references/citations returned are from the correct sections in both documents.
- **Test Results:**
    - The system identifies and retrieves content across multiple documents.
    - The aggregated response is coherent, with correct back-referencing to each source.
- **Priority:** Major
- **Status:** Passed

Test ID: 10701

- **Type:** Functional Testing
- **Objective:** Validate Document Re-Upload (Versioning)
- **Steps:**
    - Log in and open a QD.
    - Upload "Document A" (initial version).
    - Make changes to "Document A" locally (e.g., add paragraphs).
    - Re-upload the updated "Document A" under the same file name.
    - Observe how SVQ.ai handles this new version (overwrite or version control).
- **Test Results:**
    - The updated document is recognized and re-processed (chunking, embedding).
    - The system either replaces the old file or creates a new version, depending on design specs.
    - No leftover references point to outdated data.
- **Priority:** Normal
- **Status:** Passed

Test ID: 10801

- **Type:** Functional Testing
- **Objective:** Verify Large File Size Limits and Error Handling
- **Steps:**

    - Log in to SVQ.ai with valid credentials.
    - Open an existing QD.
    - Attempt to upload a file that exceeds the platform's max size limit (e.g., >500MB).
    - Observe any error messages or system behavior.
    - **Test Results:**
    - The system immediately rejects files larger than the configured limit.
    - A clear, user-friendly error message is displayed.
    - No partial data is stored.
- **Priority:** Normal
- **Status:** Passed

Test ID: 10901

- **Type:** Functional Testing
- **Objective:** Validate Partial Document Search
- **Steps:**
    - Create or open a QD with two large documents.
    - Use the chat to search for content you know exists *only in the middle section* of one document.
    - Compare the system's result with the actual text location.
- **Test Results:**
    - The system locates and references the correct segments, not only the beginning or end.
    - Back-references accurately cite the relevant mid-document chunk(s).
- **Priority:** Major
- **Status:** Passed

Test ID: 11001

- **Type:** Functional Testing
- **Objective:** Confirm Handling of Multi-Language Documents
- **Steps:**
    - Upload one English-language PDF and one non-English (e.g., Spanish or French) PDF to the same QD.
    - Query content in both languages.
    - Verify the system's ability to recognize and retrieve the correct text segments.
- **Test Results:**
    - The system can handle multi-language embedding and chunking without errors.

    - Back-references in responses accurately point to the language-specific document.
- **Priority:** Normal
- **Status:** Passed

Test ID: 11101

- **Type:** Functional Testing
- **Objective:** Verify User Settings Persistence (e.g., Notification Preferences)
- **Steps:**
    - Login and navigate to the user settings page.
    - Change notification preferences (e.g., email alerts on/off).
    - Refresh the page or log out and back in.
    - Confirm that settings remain as previously selected.
- **Test Results:**
    - Settings changes persist across sessions.
    - No reversion to defaults without user action.
- **Priority:** Low
- **Status:** Passed

Test ID: 11201

- **Type:** Functional Testing
- **Objective:** Validate Renaming Query Datasource
- **Steps:**
    - Login, go to the Dashboard, and select a QD.
    - Click “Edit QD Name” to rename it.
    - Confirm the new name appears throughout the UI (Dashboard, Chat page, etc.).
    - Attempt a chat query on the renamed QD.
- **Test Results:**
    - The system correctly updates all references to the QD’s name.
    - No broken links or references to the old name.
- **Priority:** Normal
- **Status**: Passed

Test ID: 11301

- **Type:** Functional Testing
- **Objective:** Test Quick Search for Existing Documents in a QD
- **Steps:**
    - Open a QD containing multiple documents (different titles).

    - Use a “Quick Search” or “Filter” box (if provided) to locate a specific document by title or keyword.
    - Click the result to open or preview that document.
- **Test Results:**
    - The quick search function displays only matching documents.
    - Opening a document from search loads the correct file details.
- **Priority:** Low
- **Status:** Passed

Test ID: 11401

- **Type:** Functional Testing
- **Objective:** Confirm Behavior on Partial Document Upload Cancellation
- **Steps:**
    - Begin uploading a large document (e.g., 200MB).
    - Cancel the upload partway through (e.g., close the dialog or click a “Cancel” button).
    - Observe system logs or state to ensure no partial chunks are saved.
- **Test Results:**
    - The upload is fully aborted.
    - No partial data or embeddings remain in the system.
- **Priority:** Minor
- **Status:** Passed

Test ID: 11501

- **Type:** Functional Testing
- **Objective:** Verify Custom Disclaimers for Specific Document Types
- **Steps:**
    - Configure the system so that legal or highly sensitive documents automatically trigger a custom disclaimer.
    - Upload a legal document (e.g., a legislative text).
    - Open the Chat and ask for advice on that document’s content.
    - Check that the response includes or prefaces with a custom disclaimer (e.g., “This is not legal advice”).
- **Test Results:**
    - The platform consistently appends relevant disclaimers based on the document category.
    - Users are clearly informed about disclaimers when referencing specific doc types.
- **Priority:** Major

- **Status:** Passed

Test ID: 20101

- **Type:** Integration Testing
- **Objective:** Create First Query Datasource (QD)
- **Steps:**
    - Click on ‘Create Query Datasource’ button on Dashboard Page
    - Redirect to Manage Query Datasource Page w/ param type=’create’
    - Create new database for user
    - Associate symmetric key with database, and store in key management system
    - Create first Query Datasource in the users’ database
    - Notify user of successful Query Datasource creation
    - Redirect to Manage Query Datasource Page w/ param type=’edit’
- **Test Results:** The system is able to create a new database per user, obeying multi-tenant architecture, and associates symmetric key to each user for encrypting documents
- **Priority:** Critical
- **Status:** Passed

Test ID: 20201

- **Type:** Integration Testing
- **Objective:** Upload File to Query Datasource
- **Steps:**
    - Click on ‘Upload Document’ button on Manage Query Datasource page
    - Select a document from the system
    - Click ‘upload’ once the document has been loaded
    - Forward document to chunking and embedding lambda
    - Store embeddings, content and citations in QD vector database
    - Forward document to file management lambda
    - Encrypt document with user’s symmetric key
    - Upload to S3 store
    - Notify user of successful file upload
- **Test Results:** The system is able to not only chunk and embed documents, but also successfully encrypts them and stores them in the object store
- **Priority:** Critical
- **Status:** Passed

Test ID: 20301

- **Type:** Integration Testing
- **Objective:** Chat with a Query Datasource

- **Steps:**
    - Select a Query Datasource from the Dashboad Page
    - Redirect to Chat with Query Datasource page
    - Enter a query in the chat prompt
    - Forward query to the response-generation lambda
    - Retrieve embeddings from the QD vector database
    - Generate response from the LLM
    - Stream response to the user using Websocket connection
- **Test Results:** User is able to query a selected query datasource and receive replies from the system in real time using a live socket connection
- **Priority:** Critical
- **Status:** Passed

Test ID: 20401

- **Type:** Integration Testing
- **Objective:** Validate Automatic Key Rotation
- **Steps:**
    - Log in with valid credentials.
    - Upload a document to a QD.
    - Trigger an event that forces key rotation (e.g., an admin function or scheduled rotation).
    - Attempt to re-download or query the uploaded document.
- **Test Results:**
    - The system generates a new encryption key and updates the Key Management Service.
    - Previously uploaded documents remain accessible using the new key.
    - No data corruption or access errors occur.
- **Priority:** Major
- **Status:** Passed

Test ID: 20501

- **Type:** Integration Testing
- **Objective:** Check Cross-Service Error Handling (Chunking Lambda Failure)
- **Steps:**
    - Temporarily disable or simulate an error in the chunking-and-embedding Lambda.
    - Attempt to upload a new document to a QD.
    - Observe error handling in the serverless workflow.

    - Verify that no partial data is stored and that the user receives an appropriate error message.
- **Test Results:**
    - The system gracefully handles Lambda failures without storing partial/invalid data.
    - The user receives a descriptive error message prompting them to retry or contact support.
- **Priority:** Normal
- **Status:** Passed

Test ID: 20601

- **Type:** Integration Testing
- **Objective:** Validate Logging and Audit Trails for Chat Queries
- **Steps:**
    - Log in and initiate several chat sessions with different QDs.
    - Log out and log in as an admin.
    - Access the Keycloak or system admin panel to view audit logs.
    - Verify that each chat query is recorded with a timestamp, user ID, and QD reference.
- **Test Results:**
    - Every chat query and user interaction is properly logged.
    - Audit logs clearly indicate the user, timestamp, and QD used.
    - No sensitive content is stored in the logs (only minimal metadata).
- **Priority:** Major
- **Status:** Passed

Test ID: 20701

- **Type:** Integration Testing
- **Objective:** Test Collaboration with External Security Tools (e.g., Virus/Malware Scanner)

- **Steps:**
    - Configure SVQ.ai to run a third-party malware scan on each uploaded file.
    - Upload a known benign file and confirm it passes.
    - Attempt to upload a file with a known virus/malware signature (in a controlled environment).
    - Observe system response and logs.
- **Test Results:**
    - The benign file uploads normally after passing the malware scan.

    - The infected file triggers an immediate block, with a warning message shown to the user.
    - Logs clearly record any blocked uploads.
- **Priority:** Major
- **Status:** Passed

Test ID: 20801

- **Type:** Integration Testing
- **Objective:** Validate Integration with Third-Party LLM API for Query Processing
- **Steps:**
    - Send a test query from SVQ.ai's chat interface to the backend
    - Forward the query to the third-party LLM API via the API gateway
    - Verify authentication using API keys
    - Receive the response from the LLM API and check for a valid JSON output
    - Parse and process the response within SVQ.ai's application logic
    - Display the response to the user in the chat UI
    - Test error handling by simulating API failures (e.g., network errors, rate limits, invalid requests)
    - Review logs of API interactions for debugging, analytics and cost predictions
- **Test Results:**
    - The third-party LLM API processes queries and returns structured, relevant responses.
    - API authentication and error handling function correctly.
    - Query responses are displayed in real-time with acceptable latency.
- **Priority:** Critical
- **Status:** Passed

Test ID: 20901

- **Type:** Integration Testing
- **Objective:** Validate Asynchronous Queuing for Document Processing
- **Steps:**
    - Upload multiple documents in rapid succession (e.g., 5–10 large files).
    - Confirm that chunking/embedding is handled asynchronously via a message queue or Lambda queue.
    - Monitor processing logs to ensure each document is queued, processed in order, and completed.
    - Check for any collisions or resource starvation.
- **Test Results:**

    - All documents are eventually processed without collisions or lost tasks.
    - The queue mechanism scales automatically if many documents arrive together.
- **Priority:** Major
- **Status:** Passed

Test ID: 21001

- **Type:** Integration Testing
- **Objective:** Confirm Partial Embedding Storage Failure Recovery
- **Steps:**
    - Temporarily induce a partial failure in the vector DB (e.g., misconfigured DB or limited capacity).
    - Upload a document and observe whether chunking steps continue while embeddings fail to store.
    - Check system logs and final state of the QD.
- **Test Results:**
    - The system detects embedding storage failure and either retries or notifies the user.
    - No incomplete or inconsistent embedding data remains.
- **Priority:** Normal
- **Status:** Passed

Test ID: 21101

- **Type:** Integration Testing
- **Objective:** Check Integration with System Notifications (Email/Slack, etc.)
- **Steps:**
    - Configure a test email or Slack webhook for user notifications.
    - Perform a significant action (e.g., create QD, upload documents, trigger key rotation).
    - Verify whether an email or Slack message is sent out with the relevant info.
    - Attempt actions with minimal significance and confirm no spam notifications occur.
- **Test Results:**
    - Users/admins receive notifications only for configured high-priority events.
    - Messages contain correct event details and timestamps.
- **Priority:** Low
- **Status:** Passed

Test ID: 21201

- **Type:** Integration Testing
- **Objective:** Validate Multi-Document Summaries from the LLM
- **Steps:**
    - Upload multiple related documents (e.g., chapters of a regulation).
    - Ask the chat for a "summarized highlight across all docs."
    - Confirm the system calls the LLM with references to each relevant chunk.
    - Check that the aggregated summary is accurate and references each doc in the final response.
- **Test Results:**
    - The LLM correctly merges data from multiple documents into one coherent summary.
    - Each doc's key points are included, with references or citations.
- **Priority:** Major
- **Status:** Passed

Test ID: 21301

- **Type:** Integration Testing
- **Objective:** Verify Real-Time Collaboration Among Multiple Users
- **Steps:**
    - Have two users (User A & User B) log in simultaneously to the same QD.
    - User A uploads a new document, while User B observes.
    - Confirm that User B sees the new document appear in near real-time (e.g., via websockets or poll).
    - Both users initiate queries concurrently.
- **Test Results:**
    - New documents appear quickly in the QD for all authorized users.
    - Concurrent queries do not conflict or block each other.
- **Priority:** Major

Test ID: 21401

- **Type:** Integration Testing
- **Objective:** Check Data Export Functionality for Compliance or Backup
- **Steps:**
    - Log in as an admin user.
    - Attempt to export a user's entire QD (documents, embeddings, chat history) as a structured file or archive.
    - Verify that the system packages and encrypts data for secure download.

    - Attempt partial export of specific documents only.
- **Test Results:**
    - The system successfully bundles and exports all user data in a consistent format.
    - Partial exports work as intended, only including chosen items.
- **Priority:** Normal
- **Status:** Passed

Test ID: 21501

- **Type:** Integration Testing
- **Objective:** Validate Scheduled Maintenance Mode Integration
- **Steps:**
    - Schedule a short maintenance window in the admin console.
    - Observe how the system notifies active users (e.g., a banner or popup).
    - Ensure that attempts to upload new documents or create new QDs are blocked during the downtime.
    - Check recovery steps when maintenance ends.
- **Test Results:**
    - Users are pre-warned about upcoming downtime.
    - All in-progress actions are either paused or gracefully aborted.
    - Full functionality resumes automatically post-maintenance.
- **Priority:** Normal
- **Staus:** Passed

Test ID: 21601

- **Type:** Integration Testing
- **Objective:** Validate QD Re-Naming Across System Components
- **Steps:**
    - Create a QD named “RegDocs2025.”
    - Use the system’s rename function to change it to “RegDocs2026.”
    - Verify that all references in the DB, vector index, object store, and logs update accordingly.
    - Confirm that the chat interface also references the updated name.
- **Test Results:**
    - The new QD name is consistently reflected across all integrated services.
    - No stale references or naming mismatches remain.
- **Priority:** Normal
- **Status:** Passed

Test ID: 30101

- **Type:** User Acceptance Testing
- **Objective:** Analyze contextual/domain relevance of responses
- **Steps:**
    - Select a group of regulatory professionals
    - Login to SVQ.ai with provided credentials
    - Create a Query Datasource
    - Upload sample regulatory documents
    - Switch to chat with Query Datasource page
    - Enter queries, only previously known to the candidate, related to the uploaded documents
    - Receive and review response generated by the system
    - Evaluate response accuracy and provide feedback on relevance and clarity
    - Submit feedback on an associated survey form
- **Test Results:**
    - Query responses are contextually accurate and relevant to the uploaded document.
    - Users feel confident in using SVQ.ai for regulatory document analysis.
- **Priority:** Major
- **Status:** Passed

Test ID: 30201

- **Type:** User Acceptance Testing
- **Objective:** Confirm Ease of Onboarding for New Users
- **Steps:**
    - Provide an onboarding user with credentials and minimal instructions.
    - Have them create an account, log in, and upload a small document to a newly created QD.
    - Observe user interactions and note any confusion points.
    - Collect feedback on the user's initial impression.

- **Test Results:**
    - The user can intuitively register, log in, and upload documents.
    - Any confusion points are minor and easily resolved by in-app tooltips or messages.
- **Priority:** Normal
- **Status:** Passed

Test ID: 30301

- **Type:** User Acceptance Testing

- **Objective:** Validate Clear Error Messaging for Non-Technical Users
- **Steps:**
    - Introduce a non-technical user to SVQ.ai.
    - Simulate typical errors (e.g., invalid file upload, incorrect credentials).
    - Observe whether they understand the error messages without assistance.
    - Collect feedback on the clarity and usefulness of each message.
- **Test Results:**
    - Error messages are user-friendly, avoiding overly technical jargon.
    - Users can self-correct their actions based on the guidance provided.
- **Priority:** Normal
- **Status:** Passed

Test ID: 40101

- **Type:** Performance Testing
- **Objective:** Analyze systems performance under multiple concurrent requests for query responses
- **Steps:**
    - Prepare test environment with multiple simulated users using Selenium
    - Select a Query Datasource containing multiple files
    - Simultaneously send multiple query requests from different users (e.g., 50–100 concurrent requests)
    - Measure response times for each request and log delays, if any
    - Increase concurrency levels gradually (e.g., 200, 500, 1000 requests) to identify system limits
    - Introduce variations in query complexity (short vs. long queries) to test system adaptability
    - Observe WebSocket performance to ensure real-time streaming remains stable under load
    - Analyze results and determine if system meets required performance benchmarks

- **Test Results:**
    - The system handles a high number of concurrent queries with minimal latency
    - No significant system crashes, timeouts, or degraded functionality
    - WebSocket connections remain stable for real-time response streaming
- **Priority:** Major
- **Status:** Passed

Test ID: 40201

- **Type:** Performance Testing
- **Objective:** Measure Latency in Retrieval-Augmented Generation with Large QD
- **Steps:**
    - Upload multiple large documents (e.g., 1,000+ pages total) into a single QD.
    - Query the QD with complex questions requiring multiple document references.
    - Measure end-to-end latency from query submission to final response.
    - Repeat with concurrent queries from multiple users.
- **Test Results:**
    - The system maintains acceptable response times even with large documents.
    - No excessive memory usage or timeouts.
- **Priority:** Major
- **Status:** Passed

Test ID: 40301

- **Type:** Performance Testing
- **Objective:** Evaluate RAG Pipeline Under Sustained Usage
- **Steps:**
    - Simulate a continuous load of queries over a 12-hour period using a tool like JMeter.
    - Generate queries of varying complexity, referencing different QDs.
    - Track response time, error rates, and serverless function invocations.
    - Identify any trends in performance degradation.
- **Test Results:**
    - System remains stable and responsive under prolonged load.
    - No memory leaks or large spikes in function errors over time.
- **Priority:** Major
- **Status:** Passed

Test ID: 40401

- **Type:** Performance Testing
- **Objective:** Benchmark S3 Document Upload and Download Speeds
- **Steps:**
    - Use a set of files (small, medium, large) to upload in batch.
    - Measure the average upload time per file.
    - Immediately download the same files (authorized user) and measure average download time.

    - Repeat at different times of day to account for network variability.
- **Test Results:**
    - Upload/download times remain within acceptable thresholds for each file size.
    - No significant or unexplained throughput drops.
- **Priority:** Minor
- **Status:** Passed

Test ID: 40501

- **Type:** Performance Testing
- **Objective:** Evaluate the Efficiency of Chunking and Embedding Large Documents
- **Steps:**
    - Upload a large document (e.g., a 500+ page pdf) to a QD
    - Measure preprocessing time for document chunking
    - Verify the number of chunks generated and their consistency
    - Process embeddings for all chunks and record embedding generation time
    - Store embeddings in the vector database and measure database write latency
    - Query the document and check retrieval speed for relevant embeddings
    - Repeat with increasing document sizes to identify potential performance bottlenecks.
- **Test Results:**
    - Chunking and embedding complete within an acceptable time frame.
    - Querying remains fast, even for large documents.
    - No significant slowdowns, crashes, or resource exhaustion.
- **Priority:** Major
- **Status:** Passed

Test ID: 50101

- **Type:** Security Testing
- **Objective:** User's Documents are auth-protected
- **Steps:**
    - Upload a document to a selected Query Datasource
    - Verify successful uploading of document to correct s3 bucket
    - Switch to a different user
    - Access S3 bucket (URL) from a different authenticated user using Selenium web scraping to execute javascript to fetch from bucket url
    - Notify user of Internal Server Error 500–do not notify user of unauthorized access
    - Switch to a correct owner

    - Redo bucket fetch with Selenium bot
    - Notify user of Status OK 200
- **Test Results:**
    - System maintains user-authenticated access to buckets
    - If a bucket access is unauthenticated, response is masked with an Internal Server Error to hide existence of bucket
- **Priority:** Major
- **Status:** Passed

Test ID: 50201

- **Type:** Security Testing
- **Objective:** Users' Documents are encrypted successfully
- **Steps:**
    - Upload document for an authenticated user, and a selected QD
    - Use authenticated user to redownload uploaded document
    - View document content and expect to be encrypted
    - Fetch user symmetric key from the key management system
    - Decrypt document with symmetric key
    - View document and review content
- **Test Results:**
    - System successfully encrypts documents with users's symmetric keys before uploading to an object store
    - Users' symmetric keys successfully decrypt object content
- **Priority:** Major
- **Status:** Passed

Test ID: 50301

- **Type:** Security Testing
- **Objective:** Verify SQL Injection Protections in QD Queries
- **Steps:**
    - Log in with a valid user and open a QD chat.
    - Enter malicious payloads resembling SQL injection (e.g., '; DROP TABLE users; --).
    - Observe how the system handles the input.
    - Check logs for any suspicious query attempts.
- **Test Results:**
    - The system properly sanitizes or escapes user input, preventing SQL injection.
    - Any suspicious activity is logged without harming database integrity.
- **Priority:** Major

- **Status:** Passed

Test ID: 50401

- **Type:** Security Testing
- **Objective:** Test Role-Based Access Control (RBAC)
- **Steps:**
    - Assign “Regular User” role to one account and “Admin” role to another.
    - Attempt to perform admin-level functions (e.g., forced key rotation) from the “Regular User” account.
    - Confirm the action is denied.
    - Sign in with the “Admin” account and retry the same action.
- **Test Results:**
    - Regular users cannot perform restricted admin functions.
    - Admin account successfully executes the action.
- **Priority:** Major
- **Status:** Passed

Test ID: 50501

- **Type:** Security Testing
- **Objective:** Validate Keycloak Token Expiration and Refresh Flow
- **Steps:**
    - Log in to SVQ.ai with a short token expiration set in Keycloak (e.g., 5 minutes).
    - Keep the browser session active, performing normal queries.
    - After the token expires, attempt another chat query.
    - Observe whether the system automatically refreshes the token or prompts re-authentication.
- **Test Results:**
    - Expired tokens are invalidated.
    - The system either seamlessly refreshes the token or redirects the user to log in again.
- **Priority:** Normal
- **Status:** Passed

Test ID: 50601

- **Type:** Security Testing
- **Objective:** Validate Data Isolation in Multi-Tenant Architecture
- **Steps:**
    - Create two different user accounts (User A and User B).

    - Each user creates a QD and uploads unique documents.
    - Login as User A and try to access or query documents from User B's QD.
    - Observe the system's response and any logs.
- **Test Results:**
    - User A cannot see or query documents from User B's QD.
    - No data leakage or unauthorized cross-tenant access.
- **Priority:** Critical
- **Status:** Passed

Test ID: 60101

- **Type:** Usability Testing
- **Objective:** Ensure Accessibility and Responsiveness of SVQ.ai's User Interface
- **Steps:**
    - Access SVQ.ai on multiple devices (desktop, tablet, mobile) to assess responsiveness
    - Verify that the interface adapts properly to different screen sizes without layout issues
    - Verify that text is responsive to screen sizes
    - Verify static images scale resolution to adapt to screen size
    - Test the platform using screen readers and assistive technologies for accessibility compliance
    - Confirm that alternative text is present for non-text content (e.g., icons, images)
    - Validate that keyboard navigation functions correctly across all interactive components
- **Test Results:** SVQ.ai interface remains fully accessible, responsive, and adheres to accessibility best practices.
- **Priority:** Normal
- **Status:** Passed

Test ID: 60201

- **Type:** Usability Testing
- **Objective:** Evaluate the Intuitiveness of the Query Datasource Flow
- **Steps:**
    - Shortlist candidates to interact with SVQ.ai without any prior guidance/exposure to the system
    - Login to SVQ.ai for each candidate and navigate to the dashboard.
    - Locate the option to create a new Query Datasource.
    - Upload a set of documents and verify that they are processed without errors.
    - Edit an existing Query Datasource by adding or removing files.

    - Delete a Query Datasource and confirm that it no longer appears in the dashboard.
    - Assess if system messages, tooltips, and UI prompts guide users effectively.
    - Assess if candidates can successfully complete the flow without oversight
- **Test Results:** Users can seamlessly create, manage, and delete Query Datasources without confusion or assistance
- **Priority:** Normal
- **Status:** Passed

Test ID: 60301

- **Type:** Usability Testing
- **Objective:** Verify Clarity of System Tooltips and Tutorials
- **Steps:**
    - Invite a small group of first-time users to explore the platform.
    - Observe if they notice and use tooltips or tutorials.
    - Gather feedback on whether the explanations were sufficient or confusing.
    - Measure time taken before they independently upload documents and query them.
- **Test Results:**
    - Users rely on tooltips/tutorials to learn the system quickly.
    - Feedback indicates that tutorials are neither too verbose nor too sparse.
- **Priority:** Normal
- **Status:** Passed

Test ID: 60401

- **Type:** Usability Testing
- **Objective:** Assess Workflow for Chat Interface Customization
- **Steps:**
    - Present users with a settings page that allows customizing chat background color, font size, or notification preferences.
    - Observe if users can easily locate and modify these preferences.
    - Verify that changes reflect immediately in the chat interface.
- **Test Results:**
    - Users can intuitively personalize the chat settings.
    - No confusion or searching for hidden settings.
- **Priority:** Low
- **Status:** Passed

Test ID: 60501

- **Type:** Usability Testing
- **Objective:** Evaluate Grouping Documents into a Single QD
- **Steps:**
    - Have users create multiple QDs with different sets of documents (e.g., regulatory docs, code repos, textbooks).
    - Guide them to merge or group selected documents into one QD.
    - Record any difficulty or confusion about how grouping affects future queries.
- **Test Results:**
    - Users easily understand the grouping function and can combine relevant documents logically.
    - Queries reference all merged documents in the new QD.
- **Priority:** Normal
- **Status:** Passed

Test ID: 70101

- **Type:** Compatibility Testing
- **Objective:** Ensure SVQ.ai Web Client functions properly across different web browsers
- **Steps:**
    - Open SVQ.ai on multiple web browsers (e.g., Chrome, Firefox, Safari, Edge)
    - Verify that the platform loads correctly and maintains full functionality in each browser
    - Test key features such as user authentication, document upload, and querying a datasource
    - Identify and document any inconsistencies or browser-specific issues.
- **Test Results:** SVQ.ai operates smoothly across all major web browsers, maintaining consistent performance and usability
- **Priority:** Minor
- **Status:** Passed

Test ID: 70201

- **Type:** Compatibility Testing
- **Objective:** Validate Responsiveness on Older Mobile Devices
- **Steps:**
    - Use device emulators for older Android/iOS versions.
    - Access SVQ.ai, log in, and attempt core operations (document upload, chat).
    - Note any layout issues or performance lags.

    - Record any device-specific or OS-specific errors.
- **Test Results:**
    - SVQ.ai remains usable (though potentially slower) on older devices.
    - No major UI breakage or unstoppable crashes.
- **Priority:** Minor
- **Status:** Passed

Test ID: 70301

- **Type:** Compatibility Testing
- **Objective:** Validate Integration with Different S3-Compatible Storage Providers
- **Steps:**
    - Switch the object store connector from AWS S3 to another S3-compatible service (e.g., MinIO).
    - Upload a document and ensure it is encrypted and stored.
    - Download the same document to confirm data integrity.
    - Check logs for any compatibility issues or warnings.
- **Test Results:**
    - The system seamlessly supports multiple S3-compatible providers.
    - No data corruption or unexpected errors occur.
- **Priority:** Normal
- **Status:** Passed

Test ID: 70401

- **Type:** Compatibility Testing
  Objective: Confirm Behavior with Different Keycloak Versions
  Steps:
- Deploy different Keycloak versions (e.g., older stable vs. newest release).
- Perform standard authentication flows (login, logout, token refresh).
- Document any version-specific issues (e.g., changed endpoints, deprecations).
  Test Results:
- SVQ.ai's authentication layer remains stable across Keycloak versions.
- Minimal code changes are needed to maintain compatibility.
  Priority: Minor
- **Status:** Passed

Test ID: 70501

- **Type:** Compatibility Testing

- **Objective:** Confirm Cross-Platform Behavior of the Chat UI (Desktop vs. Mobile Browsers)
- **Steps:**
    - Access the Chat UI from a desktop browser (Chrome/Firefox/Edge) and test queries.
    - Switch to a mobile browser and repeat typical tasks (upload doc, query doc).
    - Compare performance, layout, and feature availability.
    - Document any discovered inconsistencies.
- **Test Results:**
    - Core functionality (upload, chat, referencing) works similarly across devices.
    - Minor UI differences are acceptable, but no blocking issues occur.
- **Priority:** Norma
- **Status:** Passed

5.3. RAG Benchmarks

5.3.1. Testing the RAG model

An important part of testing our system is being able to benchmark [9] and evaluate the RAG model that we are using. In order to do so, we have devised a testbench which includes one hundred questions, alongside four metrics that allow us to evaluate the responses to those questions. This section will discuss the types of questions included in our benchmark, how they were generated, and then finally will establish the four metrics being used.

5.3.2. Types of Questions in the Testbench

The RAG model has to respond to any question that is asked to it; however, the RAG model may perform better at answering some questions than others. Therefore, we decided to proceed by splitting our questions into four different question-types. The four types can be seen in the table below. In the table, the term ‘chunk’ refers to a part of the text retrieved by the RAG model.

Question Type	Description	Information Distribution	Examples	Failure Mode	No. of Info Chunks	Info Consistency	Ratio
Single-Source Questions	Questions with answers that are found in one specific chunk.	Information exists in one place.	Who discovered penicillin? What is the capital of France?	Failure to retrieve the correct chunk or confusion with irrelevant chunks.	1	TRUE	40
Multi-Source Questions	Questions that require information to be aggregated or linked across multiple chunks.	Information is scattered across multiple chunks.	Which company acquired Instagram and when? Who invented the telephone and how did it evolve?	Failure to connect or aggregate information from multiple chunks. Failure to retrieve multiple relevant chunks.	≥2	TRUE	40
Conflicting-Source Questions	Questions with contradictory information across chunks.	Conflicting or contradictory data in different chunks.	What year was Pluto classified as a dwarf planet? Is the Earth flat or round?	Model picks one side of the contradiction arbitrarily, without acknowledging the conflict.	≥2	FALSE	10
No-Source Questions	Questions with no available information or subjective opinions.	No relevant data in any chunk or contains misinformation	Who was the 75th President of the US? What is the best smartphone? Did Einstein win a Nobel Prize for Relativity?	Model hallucinates an answer, fails to reject or states that there is no information.	0	-	10

Each question type has a distinct failure mode and the diversity within these four question types will be able to sufficiently test the ability of our RAG model. Additionally, the distribution of these questions should not be equal because the distribution should replicate real world conditions. For that reason, we have picked the ratio above. Using these question types, we have established a skeleton that can be used to generate a structured benchmark allowing us to test different failure modes within our RAG system.

5.3.3. Generating a Testbench

In order to generate questions, we decided to use a one thousand page Canadian aviation regulation document inline with our goal of establishing SVQ.ai as an effective tool to interact with regulatory documents. The nature of regulatory documents means that there would be no conflicting-source questions, meaning we would have to redistribute the question type as seen in the table below. Additionally, in order to be able to evaluate the testbench effectively, we want to establish a ground-truth. Given the nature of the document, its length and the different question types, it is difficult to have an absolute ground-truth; therefore, when generating questions, we try to get as close to a ground-truth as possible. The difficulty in finding the ground truth can be highlighted with the following question: 'What regulations must be kept in mind when starting an air charter company in Canada?'. This multi-source question can be answered from the aspect of business regulations, aviation regulations, and safety regulations. Choosing what content to include as a response to this question is somewhat subjective, and so defining a ground-truth is difficult. Additionally, each aspect would have to be searched for in a one thousand page document manually in order to find the absolute ground-truth, which would take far too long for a hundred questions. Therefore, we have aided the process using LLMs in a way that will be explained further for each question type.

Question Type	Ratio
Single-Source Questions	40
Multi-Source Questions	50
No-Source Questions	10

Generating Single-Source Questions:
In order to generate single-source questions, we split the original document into smaller chunks, feeding a random chunk to the LLM and asking it to come up with very specific, factual questions whose answer is found only within a single chunk. Alongside the question, we also asked the LLM to reference the exact part

of the document that includes the answer to the question. This part will be used in our metrics as the ground-truth. Additionally, we made the LLM generate a sample answer to get an idea of what we were looking for. It is important to note that the answer does not have to match exactly what the LLM comes up with in that stage; however, it is very important that the RAG model is able to retrieve the part of the document used as a reference in this part as it acts as a ground-truth for our model. This was done to automatically generate forty questions, after which we manually checked to make sure that they were single-source questions, the questions were answered by the references, and that the referenced text actually exists in the original document.

Generating No-Source Questions:
A similar approach was used to generate no-source questions. The LLM was also fed with a random chunk; however, this time it was asked to generate either opinion based questions, trick questions or questions that are based on fake information that looks similar to what is in the chunk. For this part, the references were empty since the answer does not exist, and the expected answers highlight that the question cannot be answered based on the information in the document. Once again, the generated questions were manually checked.

Generating Multi-Source Questions:
In order to generate multi-source questions, we attempted to generate question topics, semantically search the document, and then ask the LLM to come up with questions based on the search results; however, we saw little success with this approach. The LLM was generating improper questions, with it frequently generating single-source questions rather than multi-source ones. Moreover, we had difficulty checking whether the answer to the questions also exists somewhere apart from the chunks that were provided to the LLM. For this reason, we decided to take a different approach.

We decided to aggregate single-source questions in a random order to create a two-part multi-source question. An example of such a question is: ‘According to the Canadian Aviation Regulations, how many passengers can a holder of a pilot permit, an ultra-light aeroplane endorsed with a passenger-carrying rating carry on board an ultra-light aeroplane? Additionally, To what operations does Subpart 1, Foreign Air Operations apply in Canada according to section 701.01?’. We did this for fifty questions, storing the answers and references similar to the other question-types. Additionally, we aggregated between 2-6 single-source questions in order to generate multi-source questions. The entire testbench that we created can be found under our github repo.

By using this approach, we are able to see whether the RAG model is able to retrieve relevant chunks from different parts of the document, and whether it is able to put these together to form a complete and coherent answer. However, this approach fails to test how the RAG model performs when the parts of the document needed to be searched are not directly apparent as with the example query: 'What regulations must be kept in mind when starting an air charter company in Canada?'. Including such questions in our testbench would weaken our use of ground-truth references which will limit us in evaluating the responses effectively. Given that a compromise is made here, this must be kept in mind during evaluation and this sort of edge-case should be checked manually during testing.

5.3.4. Evaluation Metrics

In order to create an evaluation criteria to use alongside our testbench, we have followed [1] closely; however, we saw the lack of a ground-truth in their approach to be a major limitation and therefore, have decided to extend upon their research. Although we use the same metrics, our calculations are slightly different due to the fact that we have decided to use a ground-truth as a part of our evaluation. These differences will be explained in more detail alongside the method of calculation for each of our four criteria.

Context Relevance:
Context relevance is defined as the length of overlap between the ground-truth references and the total length of all the chunks retrieved by the RAG model. This relationship has been expressed in the equation below, where $|D|$ is the length of the set of chunks retrieved by the RAG model, i is the index of the chunk within the set D, and G is the ground-truth.

$$\frac{\sum_{i=1}^{|D|} Len(G \cap d_i)}{\sum_{i=1}^{|D|} Len(d_i)}$$

This metric is a number between zero and one. A zero would mean that nothing that the RAG model has retrieved overlaps with the ground-truth, highlighting its inability to retrieve relevant information. Whilst, a context relevance score of one would mean that everything that is retrieved matches with the ground-truth. Although a score of one is ideal, modern RAG solutions are highly unlikely to

achieve this. Retrieved documents are usually quite long relative to the ground-truth, and additionally, RAG models tend to retrieve multiple documents. Therefore, it can be expected that the models will achieve a score closer to zero than one; however, a higher score indicates better retrieval capabilities.

In [1], the context relevance is calculated by using an LLM to identify relevant pieces of information within the set of retrieved chunks. However, this approach is limited since it is possible there is additional relevant information outside of the chunks that are retrieved. In this scenario, the context relevance score calculated in [1] may be high despite the fact that the retriever did not retrieve all relevant chunks. Using our ground-truth approach, we have eliminated this problem, and are able to calculate context relevance with all relevant tokens in mind.

Context Utilization:

To measure how much of the retrieved chunks the generator is using to come up with the answer, we use the context utilization metric. In order to determine what the generator uses to come up with an answer, we prompt the LLM to highlight the part of the retrieved chunks it uses. As explained in [1], this approach aligns well with human judgement when using a chain of thought model. Using this, context utilization is defined as the length of utilized text divided by the length of the retrieved documents. This is explained in the equation below, where U_i is the text that the generator utilizes from chunk i.

$$\frac{\sum_{i=1}^{|D|} Len(U_i)}{\sum_{i=1}^{|D|} Len(d_i)}$$

Once again, this metric gives us a score between zero and one where zero corresponds to low utilization of the retrieved information and one corresponds to a complete utilization. Similar to context relevance, it is highly unlikely that modern RAG solutions will achieve a utilization of one; however, the higher they are able to score, the more efficient their retrieval. For this metric, our calculation is identical to that seen in [1].

Completeness:

Completeness is defined as the length of the overlap between the ground-truth and the utilized text divided by the length of the ground truth. This is also explained in the equation below:

$$\frac{\sum_{i=1}^{|D|} Len(G \cap U_i)}{Len(G)}$$

This metric also lies between zero and one, with a score of zero meaning that the answer generated does not use any information from the ground-truth, and a score of one meaning that the answer generated uses everything from the ground-truth. It is important that this metric is high because it ensures that the answer does not miss out on any relevant and important details.

Similar to context relevance, unlike [1], we use a ground-truth in the calculation in place of having an LLM identify relevant pieces of information within the chunk. As mentioned before, this helps with accuracy because it also includes cases where there is relevant information outside of what is retrieved and labelled as relevant by the LLM.

Adherence:
The adherence metric measures whether or not the RAG model answers with faithfulness, groundedness and attribution. In order to calculate this metric, the RAG model's response was fed to an LLM alongside the relevant context, and the LLM was prompted to judge the RAG model's response. The adherence score is binary with a value of zero meaning the RAG model is not adherent and a value of one meaning that the RAG model is adherent. An adherent RAG model means the model produces responses that align with the truth and correctly reference the utilized information without fabricating or hallucinating details. As explained in [1], chain of thought LLMs are able to make similar judgements to humans; therefore, the scores given by the LLMs can for the most part be trusted.

Overall, these four metrics allow us to evaluate different stages of the RAG pipeline, and weaknesses in certain metrics point to different stages in the pipeline. Overall, it is important to score well in all four metrics; however, due to the fact that SVQ.ai is being designed for mission critical industries like aviation regulations, adherence and completeness are very important. The table below highlights the different parts of the pipeline that these metrics try to quantify, allowing us to understand the direction improvements need to take place in.

Metric	Purpose of Metric	Pipeline Stage
Context relevance	Captures the accuracy of the retrieval process	Indexing and retrieval
Context utilization	Checks for the efficiency of retrieval and utilization by the generator	Retrieval and generation
Completeness	Ensures the response is thorough and accurate based on the ground-truth	Retrieval and generation
Adherence	Highlights hallucinations or inaccuracies in the response	Generation

Although we have developed a rigorous testbench and quantifiable metrics for evaluation, it is very important to keep the limitations of this methodology, which were highlighted above, in mind when referring to these metrics. Additionally, the metrics should be looked at separately for each question-type, because good performance across one question type does not ensure good performance across the others.

6. Maintenance Plan and Details

Maintaining the SVQ.ai platform involves a multi-faceted approach focusing on proactive monitoring, regular updates, data integrity, security, and user support. This plan ensures the system remains reliable, secure, and performs optimally as it evolves.

6.1. Routine Monitoring and Health Checks

Continuous monitoring of serverless function performance—tracking execution time, error rates, and resource utilization via AWS CloudWatch (or a similar service)—will be implemented to proactively identify and address potential bottlenecks or failures. Automated alerts will notify the development team of critical issues for prompt response, while regular checks on Turso database availability, performance, and storage capacity will ensure that our data layer remains robust. Additionally, we will monitor the health and response times of the Keycloak authentication service to maintain secure, uninterrupted user access.

6.2. Software and Infrastructure Updates

All dependencies including frontend libraries such as React and ShadCN, backend libraries, and serverless runtimes; will be regularly reviewed and updated to patch vulnerabilities and incorporate improvements. Updates to core infrastructure components (e.g., Lambda runtimes, Turso DB versions, and the Keycloak server) will be scheduled during low-traffic periods to minimize user disruption. We will strive for backward compatibility in every update and perform comprehensive regression testing in a staging environment before deploying to production. The performance and relevance of our RAG models—both embedding and generation LLMs will be periodically benchmarked, with updates or fine-tuning applied as needed to reflect advances in the field.

6.3. Data Management and Integrity

To safeguard user data, automated backups of Turso tenants and configuration data will be taken every seven days and stored securely; these backups will be tested periodically for restorability. We will perform regular integrity checks on vector embeddings and the underlying knowledge graph, and manage AES encryption keys for document storage through secure procedures, including scheduled key rotations as validated in our testing protocols.

6.4. Security Maintenance

Security audits and vulnerability scans of both codebase and infrastructure will be conducted on a regular basis, supplemented by periodic penetration testing to uncover any weaknesses. Security patches for operating systems, libraries, and services will be applied promptly in accordance with their severity. Access logs and audit trails captured via Keycloak and our application logging framework will be reviewed continuously to detect and investigate any suspicious activity.

6.5. Issue Resolution and Support

We will maintain a clear process for reporting, tracking (via YouTrack), prioritizing, and resolving bugs to ensure that issues are handled efficiently. Our user support system; including an up-to-date FAQ and knowledge base will incorporate information on new features, common pitfalls, and troubleshooting steps.

6.6. User Communication

For any planned maintenance requiring significant downtime, users will receive at least one day’s advance notice, and efforts will be made to keep downtime to an absolute

minimum. A maintenance mode validated through testing will be employed to inform users and prevent disruptions during updates. Every release will be accompanied by detailed notes describing new features, improvements, and bug fixes.

7. Other Project Elements

7.1. Consideration of Various Factors in Engineering Design

In the design and development of SVQ.ai, several critical factors must be considered to ensure the system is not only functional but also responsible, sustainable, and aligned with societal and environmental needs. These factors include economic, environmental, social, political, ethical, safety, and sustainability considerations. Each of these plays a vital role in shaping the system's design, implementation, and long-term impact.

7.1.1. Economic Factors

Economic considerations were central to the design of SVQ.ai, as the platform needed to balance cost-effectiveness with high performance. To minimize operational costs, the system leverages a serverless architecture powered by AWS [10] Lambda, which dynamically scales resources based on demand, eliminating the need for continuously running servers. This approach significantly reduces infrastructure expenses while maintaining scalability. For data storage, SVQ.ai uses Turso, a distributed SQLite database, which provides cost-efficient, low-latency storage with built-in multi-tenant support. Additionally, the platform relies on open-source models and libraries, such as Hugging Face Transformers for LLM integration [11] and FAISS (Facebook AI Similarity Search) [12] for vector embeddings, to minimize development and licensing costs. By optimizing resource utilization and leveraging cost-effective technologies, SVQ.ai ensures financial sustainability without compromising performance.

7.1.2. Environmental Factors

Environmental sustainability was a key consideration in the design of SVQ.ai. The platform’s serverless architecture reduces energy consumption by automatically scaling resources up or down based on user demand, avoiding the energy waste associated with idle servers. Furthermore, the use of Turso’s edge-hosted databases ensures data is stored and processed closer to users, reducing the carbon footprint associated with data transmission. The platform also employs efficient chunking and embedding techniques to minimize computational overhead, further contributing to lower energy usage. By prioritizing

environmentally conscious design choices, SVQ.ai aligns with global efforts to reduce the environmental impact of technology.

7.1.3. Social Factors

SVQ.ai was designed with a strong focus on social impact, particularly in enhancing accessibility and usability for professionals dealing with complex regulatory documents. The platform's intuitive chatbot interface, built using React + Typescript front-end and integration with ShadCN component libraries make advanced document analysis accessible to a broader user base. This helps *non-technical* individuals and smaller organizations—often without dedicated legal/IT departments, gain insights from regulatory documents.Additionally, the platform's back-referencing feature provides explicit citations linking answers back to the original text segments stored in Turso. This transparency empowers users, regardless of their background to trust and verify the system's answers, fostering higher confidence in the information provided.

The platform also provides Document Grouping into QD (Query Datasource). By allowing users to upload, group, and query multiple documents as a single data source, SVQ.ai encourages collaboration across teams and communities. This functionality can be particularly transformative for grassroots organizations or nonprofits that need coherent, quick access to policy or legal references.

7.1.4. Political Factors

Political considerations, particularly compliance with data privacy and regulatory standards, were critical in the design of SVQ.ai. The platform adheres to stringent data protection regulations, such as GDPR, by implementing robust encryption using AES-256 for securing user documents both at rest and in transit. The database-per-user approach ensures data isolation, minimizing the risk of unauthorized access. Additionally, SVQ.ai integrates Keycloak, an OIDC-compliant authentication service, to enforce strict access controls and ensure compliance with industry standards. These measures not only protect sensitive user data but also align with global regulatory requirements, fostering trust among users and stakeholders.

7.1.5. Ethical Factors

Ethical considerations were paramount in the development of SVQ.ai. The platform prioritizes accuracy and reliability by leveraging Retrieval-Augmented Generation (RAG) technology, which grounds responses in the source documents, reducing the risk of misinformation or biased outputs. For Bias Monitoring the RAG service includes processes that compare AI-generated answers with the original embeddings to detect significant divergences or questionable content.

This helps reduce unethical biases or misinformation by flagging or refusing to provide answers when the system lacks sufficient context.To ensure transparency, SVQ.ai includes disclaimers when documents do not fully answer a query, promoting ethical responsibility in AI interactions. The platform also employs AES encryption at Rest and in Transit and database isolation to protect user data, reflecting ethical principles of user autonomy and confidentiality. AES-256 encryption is applied to all user-uploaded documents in object stores, and TLS/HTTPS is enforced for data in transit.By adhering to these ethical guidelines, SVQ.ai ensures responsible and trustworthy AI usage.

7.1.6. Safety Factors

SVQ.ai incorporates Disclaimers for Legal/Regulatory Advice. Since SVQ.ai deals heavily with legal and regulatory documents, each response in the chatbot interface may include a "non-expert system" disclaimer. This ensures users understand that final decisions should involve professional judgment or legal counsel. The platform also ensures Context Verification in Subsystem Services. Subsystems responsible for chunking and embedding (e.g., a Document Preprocessing Lambda) verify that uploaded files match the expected formats and do not contain malicious content (such as script injections). This prevents the distribution of harmful data through the system. Additionally, SVQ.ai integrates Audit Trails in Keycloak. All user activities—logins, queries, file uploads—are logged. In the event of misuse or potential incorrect interpretation of crucial safety information, these audit trails help trace potential issues back to their source, enabling a quick response.

7.1.7. Sustainability Factors

Sustainability was a key consideration in the long-term viability of SVQ.ai. The platform ensures Long-Term Code Maintenance by building the front-end in React + Typescript and orchestrating serverless functions with well-documented APIs, SVQ.ai ensures that future developers can readily onboard, enhance features, and fix bugs. This technical stability over time underpins the project's broader sustainability goals. Moreover the system also boasts Modular Architecture for Incremental Upgrades. Each subsystem—like the RAG service, the Keycloak authentication layer, and the embedding storage solution—can evolve or be replaced independently without disrupting the rest of the platform. This modularity allows for continuous adoption of more efficient or greener technologies as they become available. Lastly, the system also integrates Monitoring of Serverless Metrics. Tools like AWS CloudWatch or equivalent services in other cloud platforms help track usage metrics, enabling dynamic

scaling rules. By right-sizing resources, SVQ.ai avoids both performance bottlenecks and unnecessary wastage, ensuring sustainable resource allocation.

Factor	Effect Level	Effect
Economic	6	**Moderate to High Impact**: By using serverless architectures and open-source tools, the platform significantly reduces operational costs. This earns a *6* because cost savings are crucial at scale but balanced against other priorities.
Environmental	5	**Notable Impact**: Serverless scaling and edge-hosted databases lower energy consumption and carbon footprint. Rated *7* due to the tangible yet not all-encompassing benefits to sustainability.
Social	7	**Significant Impact**: An intuitive chatbot and transparent references broaden access and trust for users (including non-technical or smaller organizations). A high rating of *8* reflects its crucial role in user adoption.
Political	4	**Low to Moderate Impact**: Strong data-protection compliance (GDPR, secure authentication) is essential, but political influences are primarily addressed through regulatory alignment. Given these controlled risks, it's rated *4*.
Ethical	9	**High Impact**: Ensuring reliable, unbiased AI outputs and proper user data protection is paramount. The *9* underscores how ethical considerations drive both development and public acceptance.
Safety	8	**Significant Impact**: The platform includes disclaimers for legal/regulatory content, checks for malicious uploads, and maintains thorough audit trails. These measures warrant an *8*, reflecting a strong focus on user safety.
Sustainability	6	**Moderate to High Impact**: Long-term code maintenance, modular upgrades, and resource monitoring enable enduring stability. A *6* rating captures its importance, though it is somewhat balanced by other design factors.

7.2. Ethics and Professional Responsibilities

The development and deployment of SVQ.ai, an AI-powered platform designed to interact with complex and often sensitive documents, inherently carry significant ethical considerations and professional responsibilities. As engineers and developers, the team acknowledges the imperative to uphold the highest standards of integrity, accountability, and care throughout the project lifecycle. Our commitment extends beyond mere functionality to encompass the **trustworthiness, fairness, and societal impact of the system.**

7.2.1. Upholding Data Privacy and Confidentiality

A primary professional responsibility lies in safeguarding the confidentiality and privacy of user data. Given that users may upload sensitive regulatory, legal, or proprietary documents, robust security measures are not just features but ethical obligations. SVQ.ai is designed with a database-per-user architecture using Turso to ensure strict data isolation. Furthermore, all uploaded documents are encrypted using AES-256 both at rest in object storage and during transit, ensuring that user data is protected from unauthorized access. Access control, managed via Keycloak using OIDC standards, further reinforces this commitment by ensuring only authenticated and authorized users can access their respective data and functionalities.

7.2.2. Ensuring Accuracy, Reliability, and Transparency

Professionals using SVQ.ai rely on its outputs for potentially critical tasks; therefore, ensuring the accuracy and reliability of the information provided is paramount. The use of Retrieval-Augmented Generation (RAG) is a deliberate choice aimed at grounding responses in the provided source documents, thereby minimizing AI "hallucinations" and factual inaccuracies. A key feature supporting reliability and ethical transparency is back-referencing, which links generated responses directly to the specific sections of the source documents. This allows users to verify the information themselves, fostering trust and enabling informed judgment. We also recognize the responsibility to be transparent about the system's limitations, incorporating disclaimers where appropriate, particularly regarding interpretations that might constitute legal or expert advice.

7.2.3. Addressing Bias and Promoting Fairness

AI systems can inherit biases present in data or models. The team holds a professional responsibility to proactively address and mitigate potential biases within SVQ.ai to ensure fairness. While the RAG approach inherently focuses on

user-provided documents rather than broad pre-trained knowledge alone, ongoing monitoring and evaluation are necessary. The benchmarking process developed for the RAG model includes evaluating responses for adherence and accuracy, which helps in identifying potential issues. Continuous efforts will be made to refine the system to provide objective and impartial information retrieval and generation based solely on the provided document context.

7.2.4. Professional Accountability and Continuous Improvement

While SVQ.ai is designed to be a powerful assistive tool, we maintain that it does not replace professional judgment or expertise. Users retain ultimate responsibility for how they interpret and utilize the information provided by the system. Our professional responsibility includes clearly communicating the tool's capabilities and limitations. Furthermore, the team is committed to continuous improvement, not only in functionality but also in ethical practice. This involves staying informed about evolving AI ethics standards, incorporating user feedback, performing regular maintenance and security audits, and adhering to rigorous testing protocols outlined in this report to ensure the system remains robust, secure, and ethically aligned.

7.3. Judgements and Impacts to Various Contexts

The development of SVQ.ai required careful judgements balancing technical capabilities, user needs, and broader contextual impacts. Evaluating these impacts across economic, societal, environmental, and global contexts is crucial for responsible innovation. The choices made in designing SVQ.ai reflect deliberate considerations of these factors, aiming to maximize benefits while mitigating potential risks.

7.3.1. Impact on Professional Workflows and Economic Contexts

SVQ.ai is anticipated to have a significant impact on professionals who regularly engage with dense regulatory, legal, or technical documents. By automating aspects of information retrieval and providing context-aware answers through its RAG-powered chatbot interface, the platform can drastically reduce the time spent on manual document review. This efficiency gain represents a key economic impact, potentially lowering operational costs for businesses and increasing productivity in fields like compliance, law, and research. However, this also necessitates a judgement about the workforce: while SVQ.ai is designed as an assistive

tool, widespread adoption could influence the nature of roles focused purely on information retrieval. The B2C marketability focus also implies a judgement aiming for broad accessibility, potentially impacting the competitive landscape of regulatory technology tools.

7.3.2. Societal Impact and Information Accessibility

A core judgement driving SVQ.ai's development is the potential societal benefit of making complex information more accessible. By simplifying interaction with difficult texts, the platform could empower smaller organizations, non-profits, or individuals who lack resources for extensive legal or regulatory analysis. The inclusion of features like back-referencing reflects a judgement prioritizing transparency and user trust, which is vital for societal acceptance. Conversely, there's an awareness of potential negative impacts. Over-reliance on the tool without critical engagement, despite built-in safeguards like accuracy goals and disclaimers, could lead to misinterpretations. Ensuring equitable access and promoting digital literacy among users are necessary considerations to maximize positive societal impact and mitigate risks.

7.3.3. Engineering Judgements and Their Consequences

Several key engineering judgements shape SVQ.ai's impact. The choice of a serverless architecture was a judgement prioritizing scalability and cost-effectiveness over potential complexities in managing distributed functions. This impacts the economic viability and environmental footprint (potentially lower energy use) but requires robust monitoring and maintenance strategies. Implementing a multi-tenant, database-per-user model with Turso was a critical judgement balancing cost against the paramount need for data security and isolation. Similarly, adopting the RAG architecture was a judgement favouring accuracy and grounding in source material over potentially faster but less reliable generative approaches, directly impacting the system's trustworthiness. Security measures like AES encryption and Keycloak authentication represent judgements to adhere to industry best practices, impacting usability slightly but significantly bolstering security and user confidence.

7.3.4. Broader Environmental and Global Contexts

While specific design choices like serverless aim for efficiency, the team acknowledges the broader environmental context: AI development and operation contribute to global energy consumption. Continuous

optimization efforts are part of the engineering judgement to minimize this footprint. In a global context, SVQ.ai could facilitate easier understanding of international regulations or standards if applied to relevant documents. However, factors like language support (tested for multi-language documents in 11001) and varying data privacy laws globally require ongoing judgement and adaptation for wider applicability.

7.4. Teamwork Details

7.4.1. Contributing and functioning effectively on the team

The development of SVQ.ai relied on strong collaboration and efficient teamwork. By adopting an agile approach, the team maintained continuous communication, iterative development cycles, and seamless integration of new features. Project coordination was optimized through a structured workflow using tools such as Youtrack for tracking progress, and GitHub for source code management. Regular sync-up meetings facilitated progress reviews, problem-solving, and collective decision-making, ensuring that every team member's contributions were effectively incorporated. This dynamic and well-organized approach enabled the team to stay aligned with SVQ.ai's objectives while fostering innovation and efficiency.

Rowaha: Rowaha played a critical role in designing, implementing and deploying the SVQ.ai core server subsystem. His contributions include implementing robust security measures, user session management, documents' encryption and multi-tenant database, hence safeguarding the application's integrity and user data.

Zahaab: Zahaab worked on the research, design, implementation and testing of the SVQ.ai's RAG model. He has worked on the implementation of the demo model, the research and implementation of the testbench and evaluation criteria, and also the testing of the initial demo model of the RAG system.

Yassin: Yassin was responsible for market research and analysis, including researching potential customers and existing competitors. He was also responsible for research and testing of the SVQ.ai's RAG model.

Ghulam: Ghulam was responsible for the UI/UX design and frontend implementation of the project. He created an intuitive user interface with responsive design principles and a document interaction interface with specialized components. He was also involved in RAG pipeline research and implementation.

Mehshid: Mehshid was responsible for the UX/UI design and frontend implementation of the project. She was also responsible for all the Reports and documentation of the project.

7.4.2. Helping creating a collaborative and inclusive environment

Throughout the project, our team nurtured a collaborative and inclusive environment. We actively promoted open communication, frequently exchanging ideas and offering constructive feedback. This approach not only supported individual growth but also enhanced the project's development by incorporating diverse perspectives.

Additionally, we emphasized knowledge sharing, with team members–based on their expertise–leading training sessions on topics such as RAG, AI model implementation, and frontend development frameworks. By fostering a culture of mutual respect, active listening, and continuous learning, we created a space that encouraged innovation, creativity, and the open exchange of ideas, ultimately driving the project's success.

7.4.3. Taking lead role and sharing leadership on the team

Our team embraces a dynamic approach to leadership, encouraging each member to take the lead in areas that align with their core strengths and interests. In this project, **Rowaha** has assumed primary responsibility for developing the backend architecture and managing repository maintenance. By streamlining backend operations and coordinating updates, he ensures that the foundation of our system remains stable and efficiently organized.

On the frontend, **Mehshid** and **Ghulam** are jointly responsible for designing and implementing the user-facing components. While they collaborate to create a seamless user experience, Mehshid also manages meeting minutes, recording key decisions and discussions to keep the entire team aligned as well as preparing all the reports.

Meanwhile, **Zahaab** and **Yassin** focus on the RAG model, taking the lead in selecting the best approach and developing a test bench for comparing model performances. Their work ensures the system remains both accurate and efficient, driven by clear performance metrics.

All reports are a shared responsibility: each team member writes sections related to their domain, while **Rowaha** oversees alignment to ensure that the information is consistent and free of discrepancies. This model of distributed leadership leverages each person's strengths, promotes collective accountability, and bolsters overall project cohesion.

7.4.4. Meeting objectives

The Smart Vector Query (SVQ) final report presents the development of a sophisticated platform that harnesses Retrieval-Augmented Generation (RAG) to simplify interaction with complex regulatory documents. This evaluation examines the project's achievement of its core objectives—drawing on system architecture, implemented features, and rigorous testing protocols—to demonstrate overall success.

Objective 1: Advanced Document Navigation & Querying
SVQ fully realized its primary goal by implementing a cutting-edge RAG pipeline, featuring intelligent chunking, embedding generation, knowledge graph integration, and context-aware response generation. Through an intuitive chatbot interface, users can engage in natural, conversational queries that seamlessly access and interpret their documents.

Objective 2: High Accuracy and Reliability
To ensure consistent precision, the RAG system was designed to ground every response directly in source material, minimizing hallucinations. Transparency and traceability were enhanced via back-referencing, enabling users to verify answers against specific document sections. A comprehensive benchmarking framework measuring Context Relevance, Completeness, and Adherence was established to rigorously evaluate system performance.

Objective 3: Robust Security and Data Privacy
Security underpins the SVQ design through a multi-tenant Turso architecture, which enforces strict data isolation. Documents are protected with AES-256 encryption both at rest and in transit, while Keycloak configured with OIDC standards manages secure access control. Dedicated security testing validated these measures, confirming data confidentiality and regulated access.

Objective 4: Intuitive User Interface and Experience
A modern, responsive web interface built with React, TypeScript, and the ShadCN component library delivers a polished user experience. Clear usability

goals guided the design process, and planned usability testing will verify the platform's ease of navigation and the intuitiveness of its document-querying features.

Objective 5: Scalable and Performant Architecture
SVQ's serverless backend enables dynamic resource scaling in response to demand, while Turso's edge-hosted databases ensure low-latency data access. Performance testing under simulated concurrent loads and with large document sets confirmed that the system meets predefined responsiveness and stability benchmarks.

Objective 6: Long-Term Supportability and Maintainability
A comprehensive documentation strategy and a detailed maintenance plan underscore the platform's future viability. Proactive monitoring, regular updates, structured data management, ongoing security audits, and a robust user support framework will facilitate effective long-term maintenance and continuous improvement.

In conclusion, the evidence presented in this report demonstrates that the Smart Vector Query project has substantially met its defined objectives. The team has engineered a complex, RAG-powered platform that excels in security, usability, performance, and maintainability, reflecting a comprehensive and successful execution of the project's goals.

7.4.5. New Knowledge Acquired and Applied

The Smart Vector Query (SVQ.ai) project demanded a significant expansion of the team's technical skillset, requiring the acquisition and practical application of knowledge across several cutting-edge domains. Foremost among these was the deep dive into advanced Artificial Intelligence, particularly Retrieval-Augmented Generation (RAG) methodologies. Team members gained expertise in the entire RAG pipeline, including sophisticated document chunking strategies, the generation and management of high-dimensional vector embeddings (utilizing vector databases like Nano) , and leveraging Large Language Models (LLMs) for generating contextually grounded responses. A crucial part of this involved learning and applying novel techniques for benchmarking RAG systems, establishing specific evaluation metrics such as Context Relevance, Context Utilization, Completeness, and Adherence.Backend development necessitated mastering modern cloud-native technologies. The team implemented a serverless architecture, gaining practical experience in managing and orchestrating

individual functions. Designing a secure multi-tenant system involved learning about distributed databases like Turso and implementing effective data isolation strategies. Significant knowledge was acquired in Identity and Access Management (IAM), specifically applying Keycloak, OpenID Connect (OIDC) [8], and JWT standards to secure API endpoints and user sessions. Furthermore, the team applied robust cryptographic practices, implementing AES-256 encryption to ensure the confidentiality of user documents.On the frontend, the project involved applying knowledge of React and Typescript, along with modern UI component libraries like ShadCN, to create a responsive, intuitive, and aesthetically pleasing user interface. Effective state management techniques using libraries such as MobX-Keystone were also implemented. Beyond specific technologies, the team enhanced its understanding and application of comprehensive software engineering practices. This included designing and executing rigorous testing plans covering functional, integration, performance, security, and usability aspects, utilizing version control systems like GitHub, employing project management tools like Youtrack for coordination, and integrating considerations for broader engineering design factors such as economic viability, ethical implications, and long-term sustainability into the development process.

8. Conclusion and Future Work

The SVQ.ai platform effectively demonstrates the power of combining RAG technology, LLMs, and vector databases to transform how professionals interact with complex documents. By prioritizing accuracy, reliability, security, and scalability, SVQ.ai provides a robust solution for querying and extracting insights from regulatory, legal, and technical documents. The platform's architecture, leveraging a client-server model with a serverless backend and a React + TypeScript frontend, ensures both performance and a user-friendly experience.

To further enhance SVQ.ai and expand its capabilities, the following future work directions are identified:

1. **Enhanced Query Understanding**: Implement advanced Natural Language Understanding (NLU) to improve the interpretation of complex and nuanced queries, ensuring more precise and contextually relevant results.
2. **Expanded Document Modalities**: Extend the platform to support a broader range of document formats beyond text, including tables, images, and charts, to provide a more comprehensive information retrieval experience.

3. **Proactive Personalization**: Integrate user feedback mechanisms and behavior analysis to personalize responses and recommendations, thereby increasing user engagement and satisfaction.
4. **Seamless Enterprise Integration**: Develop APIs and connectors to facilitate smooth integration with existing enterprise systems and workflows, enhancing the platform's utility within organizational contexts.
5. **Continuous RAG Optimization**: Continuously refine the RAG models and underlying algorithms to minimize hallucinations, improve response accuracy, and reduce latency, ensuring SVQ.ai remains a cutting-edge solution for AI-driven document analysis.
6.

9. Glossary

9.1. Definitions, acronyms, and abbreviations

LLM (Large Language Model): An AI-based model designed to understand, process, and generate human-like text based on input data.

Retrieval-Augmented Generation (RAG): enhances AI text generation by retrieving relevant information from external sources, ensuring more accurate and context-aware responses.

Advanced Encryption Standard (AES): a symmetric encryption algorithm that secures data using fixed key sizes. It encrypts data in blocks through multiple rounds of substitution and permutation, ensuring strong security and efficiency.

OpenID Connect (OIDC): is an authentication protocol built on OAuth 2.0 that enables secure user sign-in across applications. It provides identity verification through ID tokens, allowing seamless and decentralized authentication.

Time to Interactive (TTI): a web performance metric that measures how long it takes for a page to become fully interactive. It indicates when users can reliably interact with the page without delays.

Multi-tenant architecture: a software model where a single application instance serves multiple customers (tenants), each with isolated data and configurations.

Query Datasource (QD): a domain-specific term used to extensively refer to a collection of documents for a user.

B2C (Business-to-Consumer): A commerce model where businesses sell products or services directly to consumers.

9.2. Evaluation Criteria and Rating Scale

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: Maintain 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.

10. References

[1] P. Lewis *et al.*, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," *arXiv.org*, Apr. 12, 2021. https://arxiv.org/abs/2005.11401

[2] Y. Han, C. Liu, and P. Wang, "A Comprehensive Survey on Vector Database: Storage and Retrieval Technique, Challenge," arXiv.org, Oct. 18, 2023. https://arxiv.org/abs/2310.11703

[3] hatchet-dev, "GitHub - hatchet-dev/hatchet: Run Background Tasks at Scale," *GitHub*, Apr. 15, 2025. https://github.com/hatchet-dev/hatchet (accessed May 02, 2025).

[4] E. Rescorla and T. Dierks, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246, Aug. 2008. [Online]. Available: https://tools.ietf.org/html/rfc5246

[5] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," *arXiv:1702.08734 [cs]*, Feb. 2017, Available: https://arxiv.org/abs/1702.08734

[6] Keycloak Team, Keycloak Documentation, "Securing Applications and Services Guide," [Online]. Available: https://www.keycloak.org/documentation.html

[7] National Institute of Standards and Technology, "FIPS PUB 197: Advanced Encryption Standard (AES)," U.S. Dept. of Commerce, 2023. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

[8] D. Hardt, "The OAuth 2.0 Authorization Framework," RFC 6749, Oct. 2012. [Online]. Available: https://tools.ietf.org/html/rfc6749

[9] R. Friel, M. Belyi, and A. Sanyal, "Ragbench: Explainable Benchmark for Retrieval-Augmented Generation Systems," [Online]. Available: https://arxiv.org/pdf/2407.11005.

[10] P. Sbarski, S. Kroonenburg, and A. Bernert, "Serverless Architectures on AWS," 2nd ed. Shelter Island, NY: Manning Publications, 2020.

[11] "Transformers," *huggingface.co*. https://huggingface.co/docs/transformers/en/index

[12] "Faiss," *GitHub*, Jan. 12, 2023. https://github.com/facebookresearch/faiss