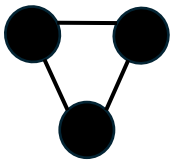

RAGgady

Michael Blithe

SWENG 837

Individual Final Project

System Context:

RAGgady is a software system designed to provide retrieval augmented generation (RAG) to end users using their own data. Generative AI (Gen AI) has changed how many people work and manage their lives. From questions about cooking, to summarizing emails, to generating practice questions to study for an exam, gen ai is here to stay. The problem with generative AI is that these models are trained on a large segment of the public internet or other text data and have no insight into the lives of almost anyone. However, users collect a significant amount of data about themselves and data that is relevant to their lives. This is a treasure trove to improve the lives of those users through generative AI and RAGgady aims to make this more accessible. In addition, these generative AI systems can invoke customizable tools which can extend their interactions to the real world or other external systems.

Key Functionality:

- Support tool calling
- Support ingestion of the following formats: PDF, Markdown, text files, HTML, Image files (PNG and JPEG)
- Automated Sync of directories to system search
- Available API for integration with multiple Gen AI providers including self-hosted ones
- Support for billing based on volume of data and number of requests

Target Users:

- Business users: including manuals, project proposal data, code bases, etc.
- Personal users: including their personal notes, calendar, email, text messages

Business Goals:

- Create cashflow positive business model
- Maintain appropriate pricing for business and personal users
- Provide seamless integration with generative AI providers and document sources
- Ensure system is transparent to users
- Provide method for users to get most of the data they have available to boost their productivity and improve their lives.
- Personal data will be retained for up to 6 months and then deleted.

Non-Functional Requirements:

- The system shall support files up to 10 megabytes
- The system shall ingest a single file in under 5 seconds
- The system shall be capable of storing at least a Terabyte of data per user
- All data in the system shall be encrypted at rest and in transit
- Data shall only be accessible to authenticated users who are authorized by the owner of data container
- The system shall maintain metrics including mean and max ingest time, and mean and max search times
- The system shall maintain up to 100 concurrent requests
- The system shall maintain an uptime of 99.9%
- A single replica of all data shall be maintained

Maintainability:

- Unit tests code coverage must be above 80%
- End to end tests must be written for each major use case
- Functions must meet 90% coverage in terms of documentation
- Automated SAST scans such as Fortify and SonarQube must be employed, and all high and critical findings addressed.
- Cyclomatic complexity will not exceed 10

Actor Descriptions

Type	Actor	Goal Description
Primary	Personal User	The personal user is a single user who controls their own data. The data is owned by a single user, searched by a single user and tools invoked are owned by that user.
Primary	Business User	Business Users are similar to personal users. However, data can be shared between Business Users with role-based access control (RBAC). This may include multiple teams and organizations within a single company.
Primary	System Administrator (Sys admin)	SysAdmins are responsible for ensuring that the system remains healthy. They will monitor the system and make any required corrections to the system to ensure stability. In addition, they will manage user accounts and provider integrations.
Supporting	Generative AI provider	This includes provider such as Anthropic, OpenAI, or Ollama for local hosting. These provide large language

		models as an API which can be invoked and provided data.
Supporting	3rd Party Data Provider	A data storage provider such as Google Drive, or Microsoft one Drive, or the User's own computer. This provides the data to RAGgady
Offstage	Payment Processor	To ensure billing a payment processor is used to charge customer credit cards
Offstage	Government Regulator	Storage of customer data includes policies such as General Data Protection Regulation (GDPR) and other such regulations that RAGgady must observe. This may also include data residency requirements.

Figure 1: Use Case Diagram

1. Upload Document:

- **Scope**: RAGgady platform
- **Level**: User Goal
- **Primary Actor**: Personal User/Business User
- **Stakeholders and Interests:**
    - **Personal User:** The personal user wants the specific file to be available for their use when they query their generative AI system.

    - **Business User:** The business user wants themselves and their team to be able to query including the new file for the generate AI system.
    - **Sys Admin:** The sys admin wants to ensure the ingestion does not impact system stability
    - **Gen AI Provider:** The Gen AI Provider wants as much relevant context to the request that is available
    - **3rd Party Data Provider:** The data provider will provide the file that will be ingested
- **Preconditions:**
    - User is authenticated to the system
    - User is authorized to upload the file to their data container
- **Success Guarantee:**
    - Embeddings are generated and available
    - The file is stored in the data container
    - Search queries can utilize the data
    - The user is notified of the successful ingestion
- **Main Success Scenario:**
    - **Step 1:** The user selects the "Upload File" Button
    - **Step 2:** The system validates that the size of the file is appropriate, and the format is one of the accepted formats
    - **Step 3:** The system provisions storage space for the file and the embeddings
    - **Step 4:** The system extracts any relevant images or text content from the uploaded file
    - **Step 5:** The system generates and stores embeddings in the embedding database
    - **Step 6:** The system stores the original file
    - **Step 7:** The system notifies the user that the ingestion was successful
- **Extensions:**
    - **2a.** The file exceeds the file size limit
        - System displays an error to the user
        - System resets to base state
    - **2b.** The file format is one which is not accepted
        - System displays an error to the user
        - System resets to base state
    - **4a.** File text/image extraction fails
        - System displays an error to the user
        - System resets to base state
- **Special Requirements:**

- Uploaded files will not exceed 10 megabytes
- Uploadable formats will include PDF, Markdown, text files, HTML, Image files (PNG and JPEG)
- Processing shall be completed in under 5 minutes
- All transmission of data will use TLS
- A progress bar shall be displayed
- All data will be stored as encrypted data at rest

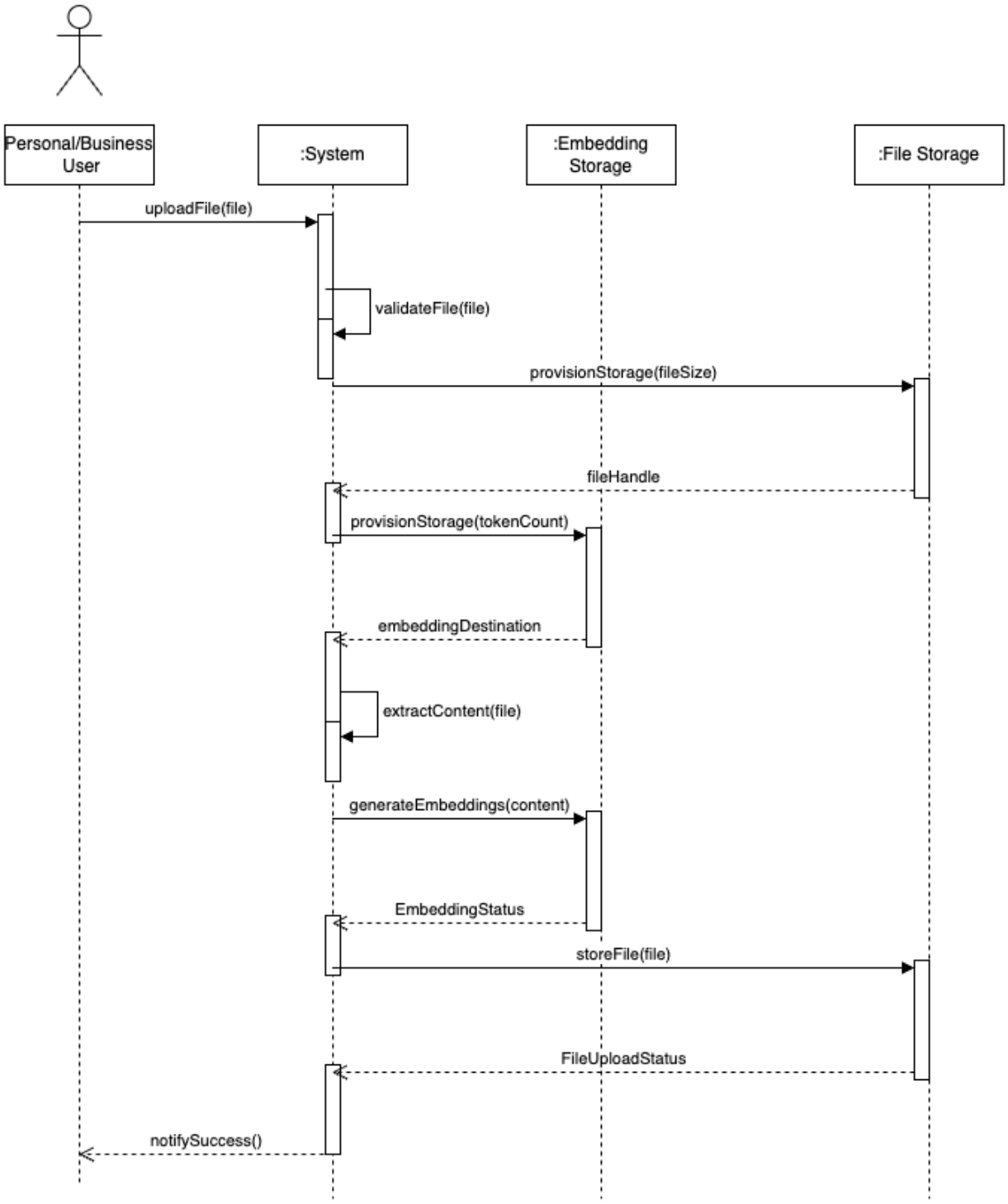

Figure 2: Sequence Diagram Use Case 1

2. Perform Search:

- **Scope**: RAGgady platform
- **Level**: User Goal

- **Primary Actor**: Personal User/Business User
- **Stakeholders and Interests:**
    - **Personal User:** The user would like relevant data to be passed from their personal archives to the destination system
    - **Business User:** The user would like relevant data to be passed from their team or organizations archives to the destination system
    - **Sys Admin:** The sys admin wants to ensure the search does not impact system stability
    - **Gen AI Provider:** Wants relevant content to make their Gen AI response to be as useful to the client as possible
    - **3rd Party Data Provider:** Wants to make sure the user has access to their data
- **Preconditions:**
    - User is authenticated to the system
    - User is authorized to upload the file to search their data container
- **Success Guarantee:**
    - A search response is returned with any relevant matches based off available user data
    - The response is cached for future use
    - System stability is maintained
- **Main Success Scenario:**
    - **Step 1:** User submits a search request
    - **Step 2:** System converts search query to embedding
    - **Step 3:** System performs search using computed embeddings
    - **Step 4:** System produces search response
    - **Step 5:** System caches search request and response
    - **Step 6:** System captures search time as metric
    - **Step 7:** System returns response to client
- **Extensions:**
    - **4a.** No response found
        - Not considered an error but response is returned with no content
    - **2a.** Embedding computation fails
        - System displays an error to the user
        - System resets to base state
- **Special Requirements:**
    - All requests will be encrypted using TLS
    - Search will occur in less than 2 seconds

- Search query will not exceed 2048 tokens
- A maximum of 5 relevant data points will be returned

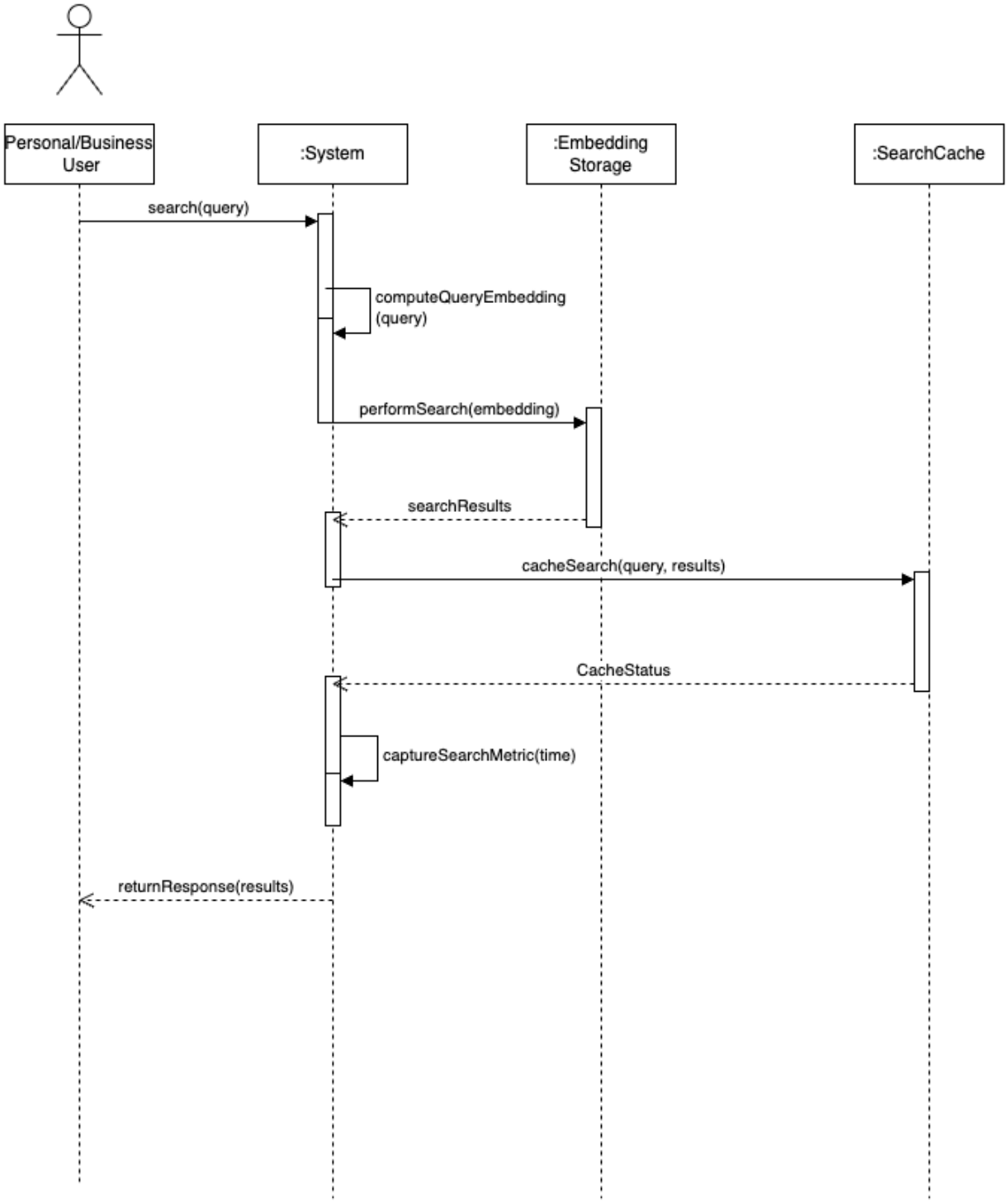

Figure 3: Sequence Diagram Use Case 2

3. Tool Invoke:

- **Scope**: RAGgady platform
- **Level**: User Goal
- **Primary Actor**: Personal User/Business User
- **Stakeholders and Interests:**
    - **Personal User:** The personal user wants some tool to be invoked via natural language to augment the capabilities of generative AI
    - **Business User:** The business user wants some tool to be invoked via natural language to augment the capabilities of generative AI
    - **Sys Admin:** The sys admin wants to ensure that security is maintained, and the tool does not have negative side effects on the rest of the system
    - **Gen AI Provider:** Wants to invoke the tool based on the needs of the user
    - **3rd Party Data Provider:** Wants the user's data to be available during the tool invocation
- **Preconditions:**
    - User is authenticated to the system
    - User is authorized to upload the file to their data container
    - A tool has been added from the tool storefront to the user's context
- **Success Guarantee:**
    - The tool is invoked, and its response provided back to the LLM
    - The tools execution does not negatively impact other users
- **Main Success Scenario:**
    - **Step 1:** The user submits a request to the LLM which requires a tool be invoked
    - **Step 2:** The gen AI provider sends a request to the system
    - **Step 3:** The system loads the appropriate tool
    - **Step 4:** The system invokes the tool
    - **Step 5:** The system records the tool invocation and runtime as a metric
    - **Step 6:** The tool's output is sent back to the gen AI provider
    - **Step 7:** The tools runtime is destroyed
- **Extensions:**
    - 4a: Tool invocation fails
        - System returns an error code to the invoking LLM
        - Tool logs are captured for analysis
        - Runtime is destroyed
        - System resets to base state
    - 3a: Tool does not exist

        - A message indicating the list of available tools and requested tool is returned to the requestor
        - System resets to base state
- **Special Requirements:**
    - Tools run in sandbox
    - Only authorized tools can be loaded from the storefront
    - Metrics are captured on tool use
    - Tools will execute in under 1 minute or be terminated
    - Tool logs will be captured

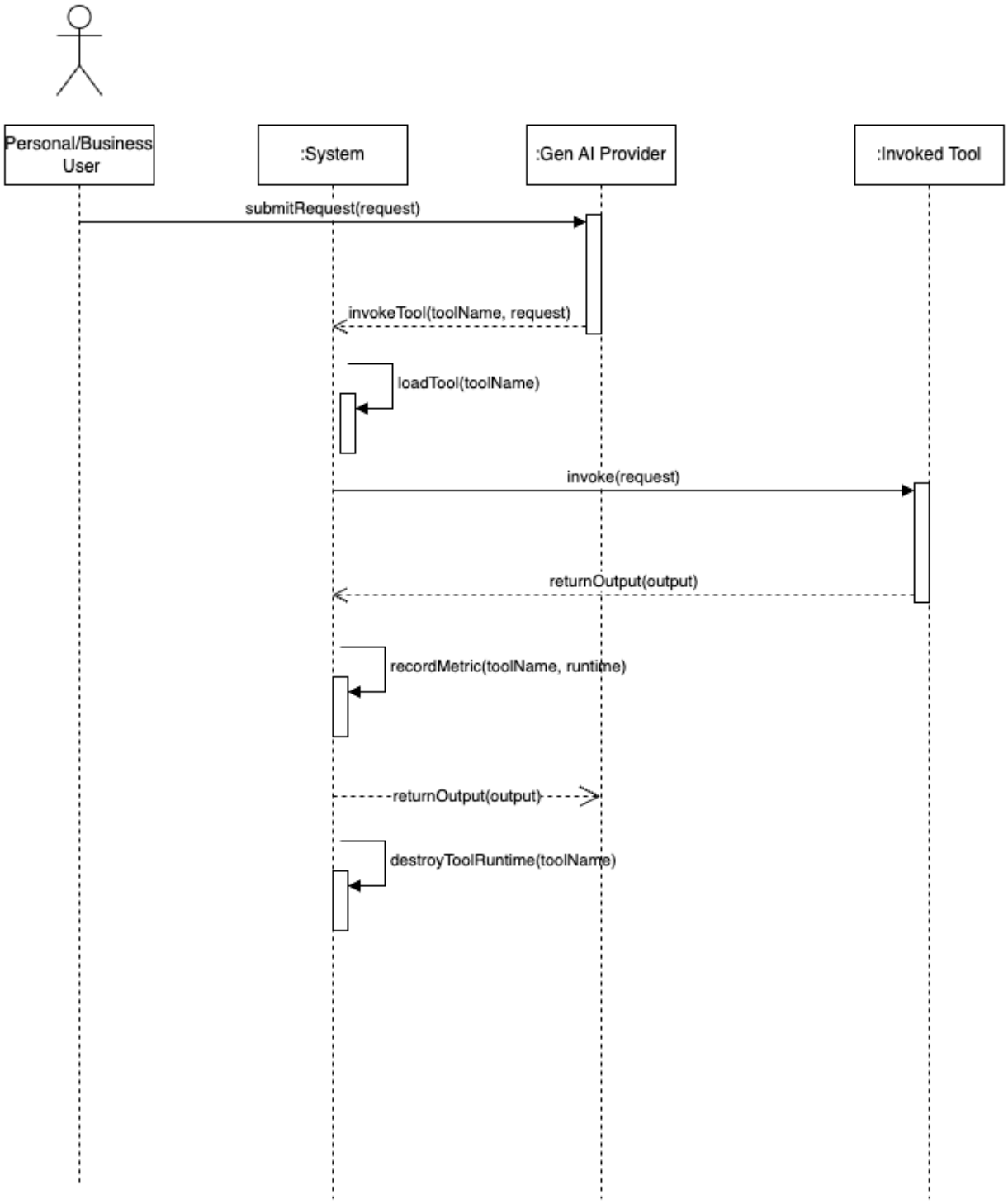

Figure 4: Sequence Diagram Use Case 3

4. Grant Access to Data Container:

- **Scope**: RAGgady platform
- **Level**: User Goal

- **Primary Actor**: Business User
- **Stakeholders and Interests:**
    - **Personal User:** N/A, Personal users have access to their own data container by default
    - **Business User:** Buisness user would like access to their shared data container for the team or organization
    - **Sys Admin:** The system admin wants to ensure that only authorized users have access to the appropriate container
    - **Gen AI Provider:** Needs secure access to user's data
    - **3rd Party Data Provider:** Needs a secure location to provide user's files to
- **Preconditions:**
    - User is authenticated to the system
    - User is authorized to make changes to data container
- **Success Guarantee:**
    - Additional user can now access the data container for either read only or read/write based on configured access
- **Main Success Scenario:**
    - **Step 1:** User goes to the share page of the data containers
    - **Step 2:** User selects the appropriate data container
    - **Step 3:** User selects the additional users to share the container with
    - **Step 4:** User selects permission level (read or write) for the new user
    - **Step 5:** System updates user permissions
    - **Step 6:** System logs the request
    - **Step 7:** Initiating user receives a notification
- **Extensions:**
    - **5a:** User is already added
        - System logs request
        - System reports that the user already exists
        - System resets to base state
    - **1a: User is a personal user**
        - Share page is not available to personal users
        - System resets to base state
- **Special Requirements:**
    - RBAC must be enforced at all times
    - TLS must be used for all requests
    - Audit logging must capture all changes

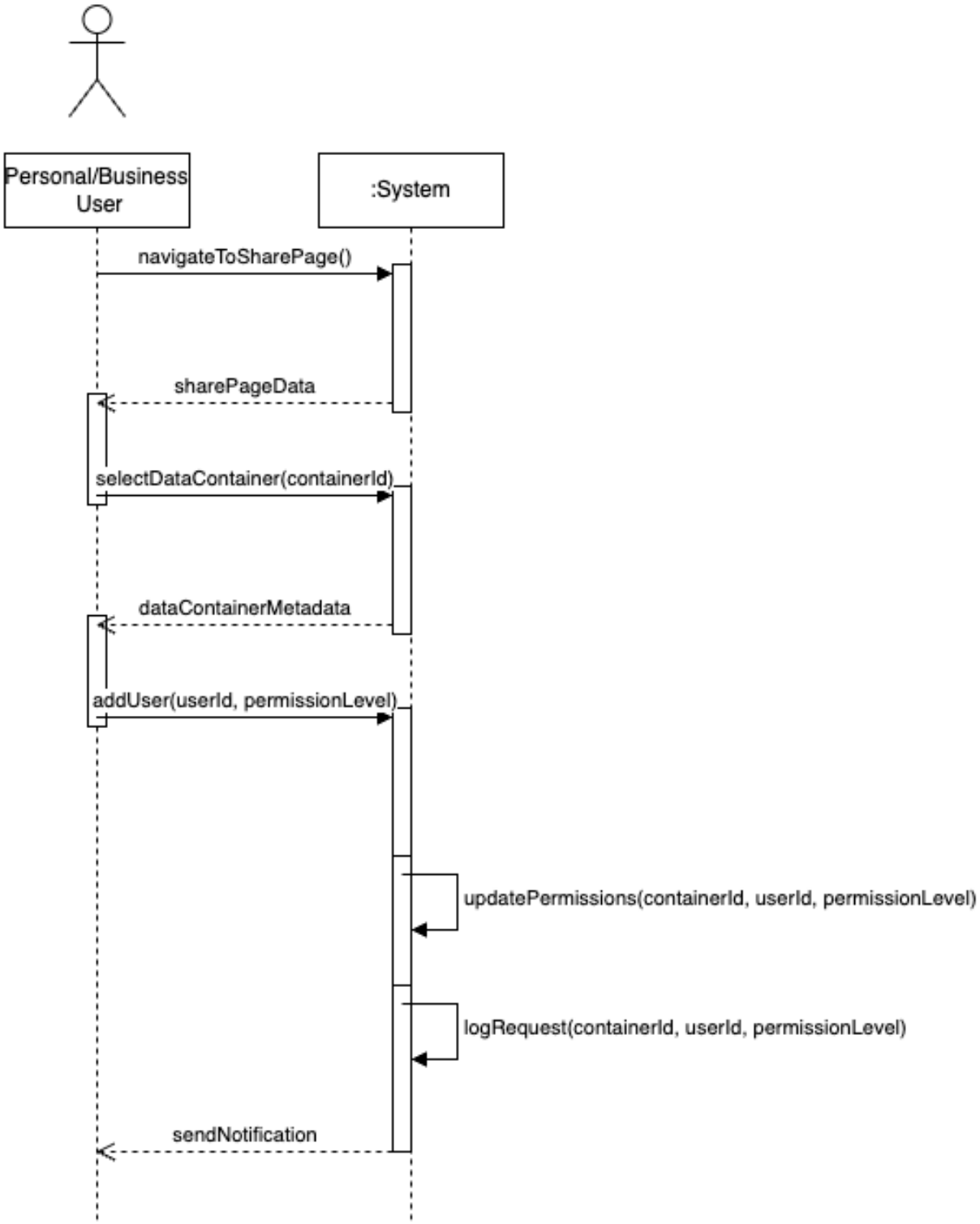

Figure 5: Sequence Diagram Use Case 4

5. Review System Metrics:

- **Scope**: RAGgady platform
- **Level**: User Goal
- **Primary Actor**: Sysadmin
- **Stakeholders and Interests:**
    - **Personal User:** The user wants to ensure that the system is usable and stable at all times
    - **Business User:** The user wants to ensure that the system is usable and stable at all times
    - **Sys Admin:** The system admin wants to verify the system is operating properly and take corrective action if required
    - **Gen AI Provider:** The Gen AI provider wants to make sure their requests to the RAG system are handled appropriately
    - **3rd Party Data Provider:** The data provider wants to make sure their files are handled properly
- **Preconditions:**
    - User is authenticated to the system
    - User is authorized to review metric data
    - A problem exists affecting system performance
- **Success Guarantee:**
    - Sys admin is able to verify key performance indicators (KPIs)
    - Sys admin is able to review system logs
- **Main Success Scenario:**
    - **Step 1:** The system notifies the sys admin that there is an issue
    - **Step 2:** Sysadmin requests system metrics
    - **Step 3:** System retrieves and displays key performance indicators (KPIs) and logs
    - **Step 4:** Sysadmin reviews the metrics and logs to verify system performance and stability
    - **Step 5:** Issues are identified sys admin takes corrective action.
    - **Step 6:** System updates metrics in real time
    - **Step 7:** Sys admin continues to monitor KPIs as they work to stabilize the system
- **Extensions:**
    - **4a:** System is stable and performing within in spec
        - No action required
        - Sys admin monitors situation

- **Special Requirements:**
    - RBAC is enforced at all times
    - Metrics are updated at 1hz
    - All logs are viewable and filterable by severity
    - All metric data is saved to a long-term archive

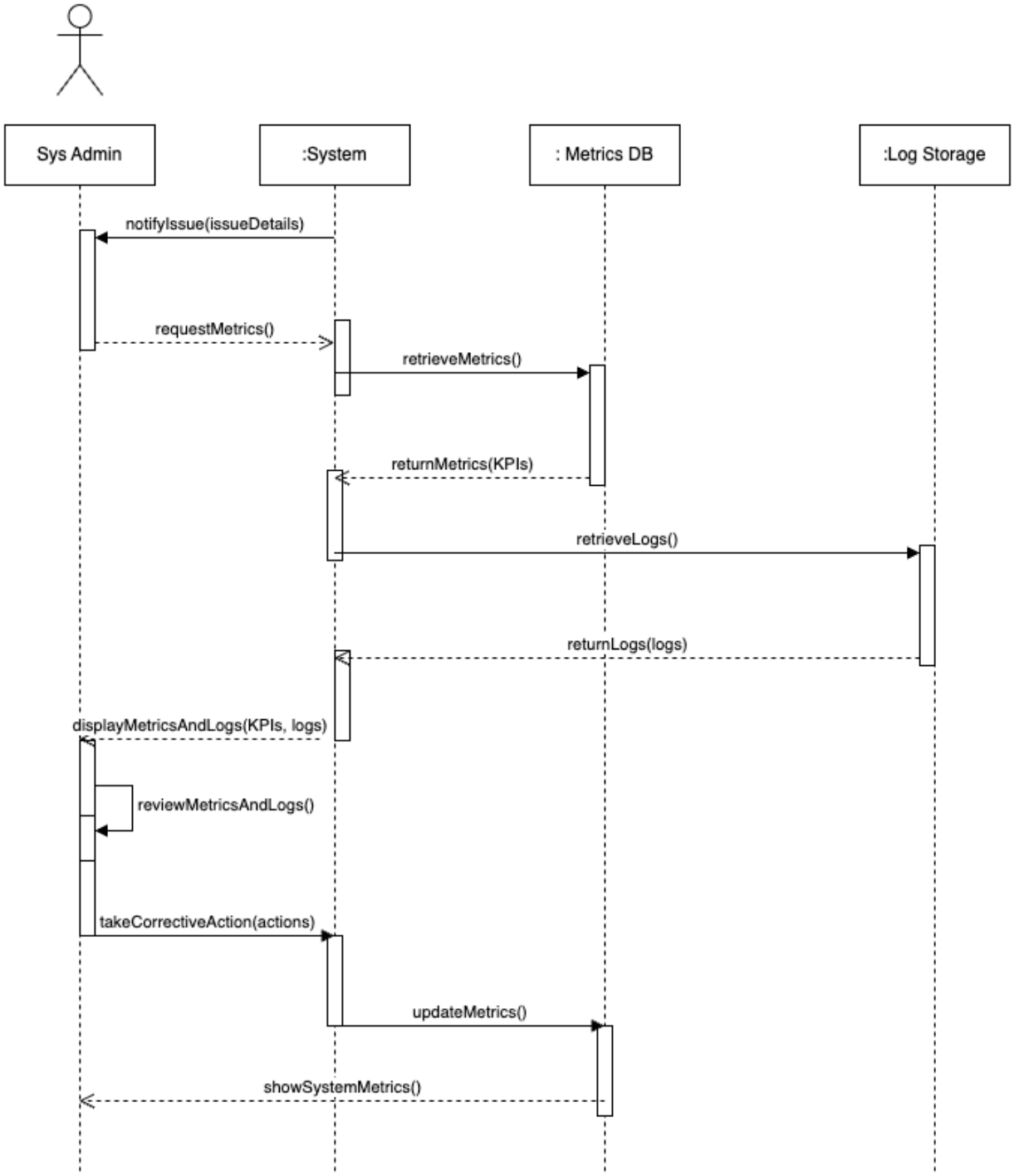

Figure 6: Sequence Diagram Use Case 5

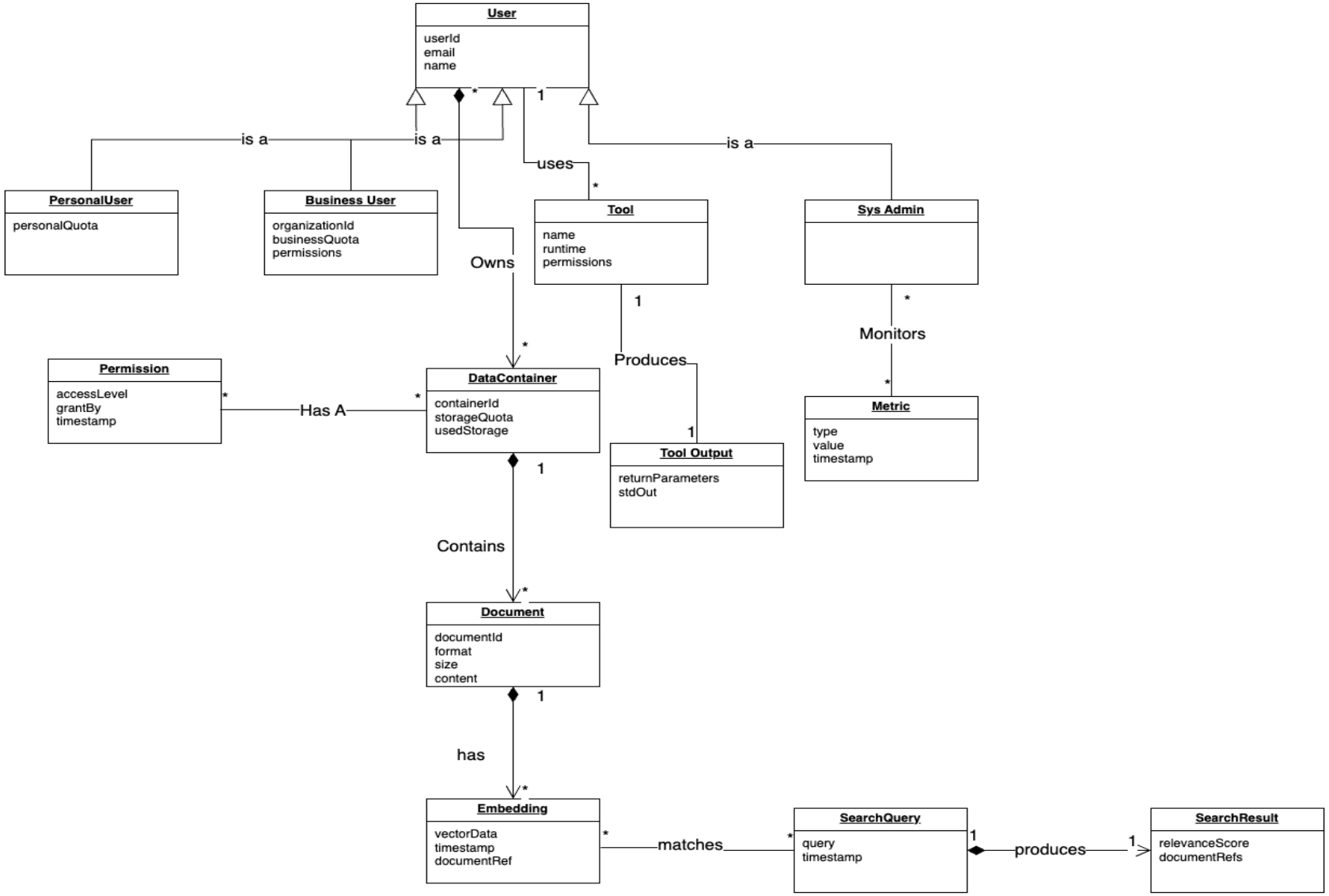

Figure 7: Domain Model

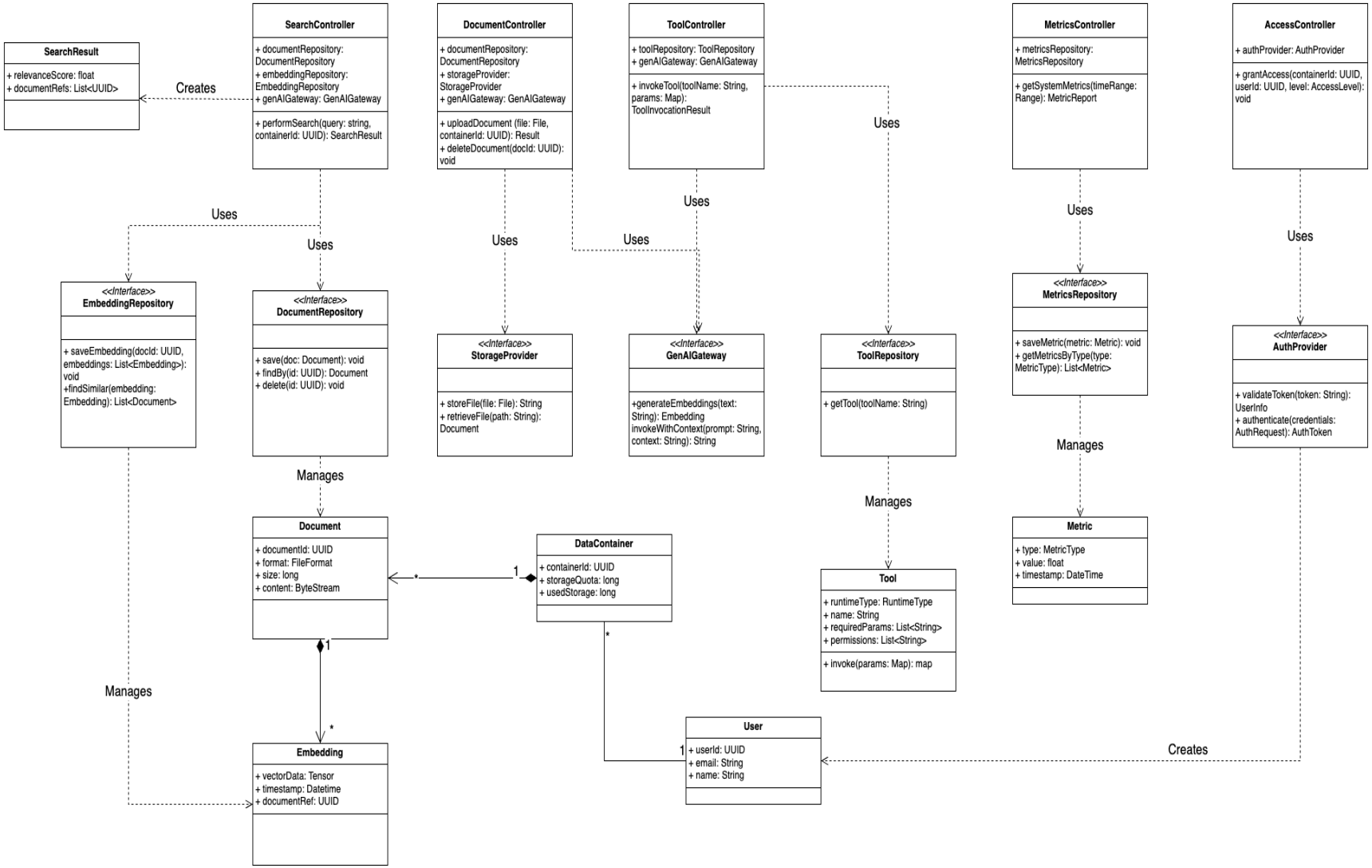

Figure 8: Class Diagram

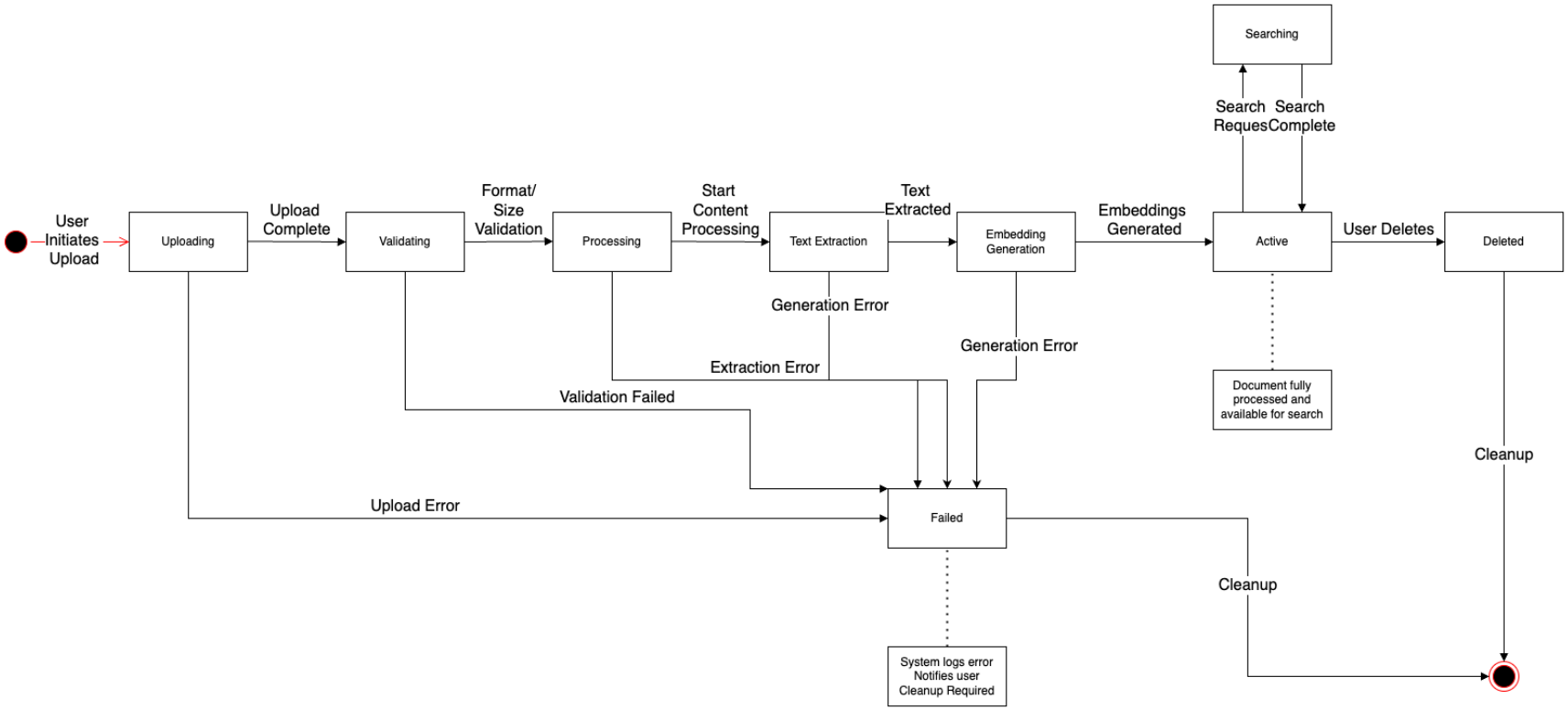

Figure 9: State Diagram

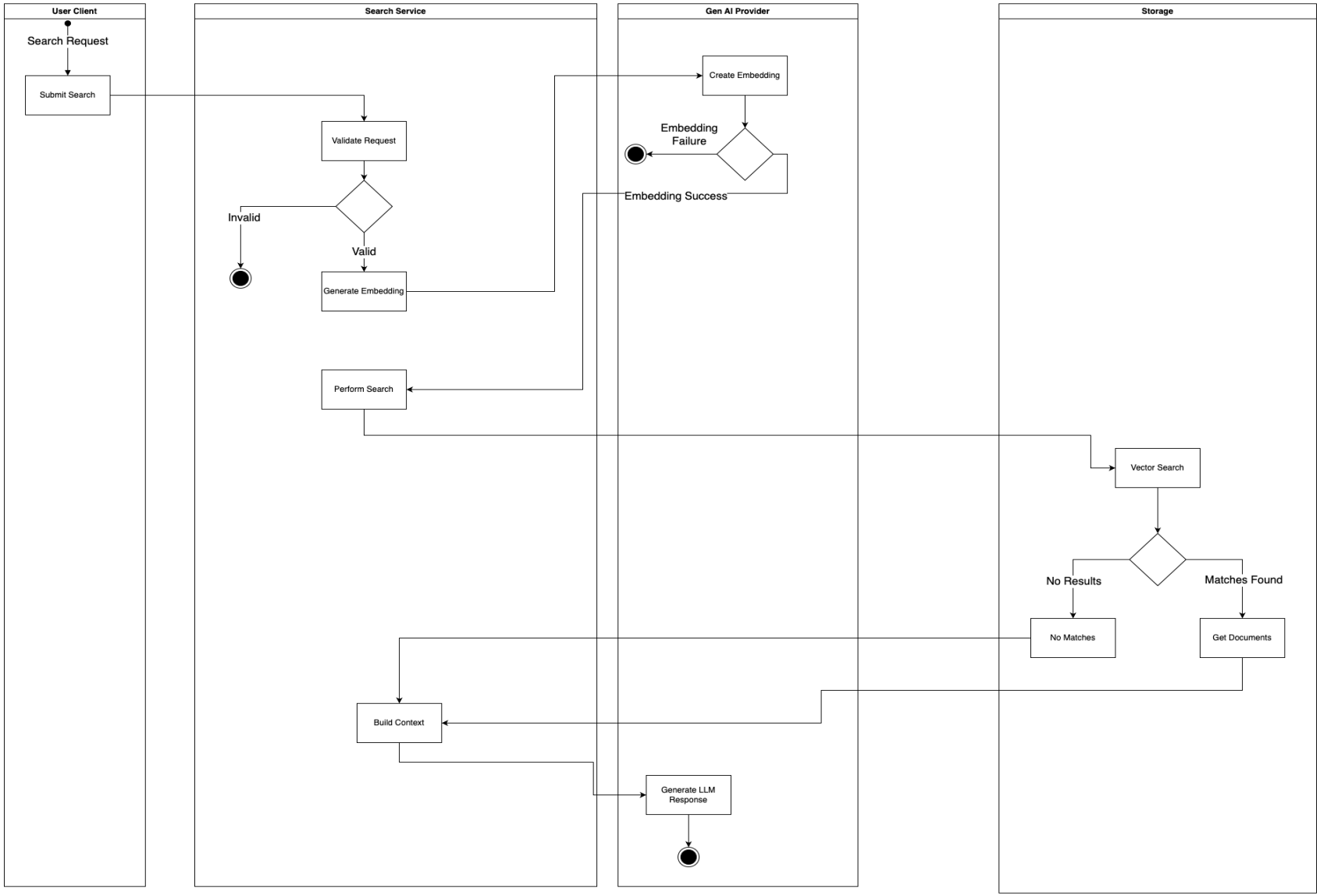

Figure 10: Activity Diagram

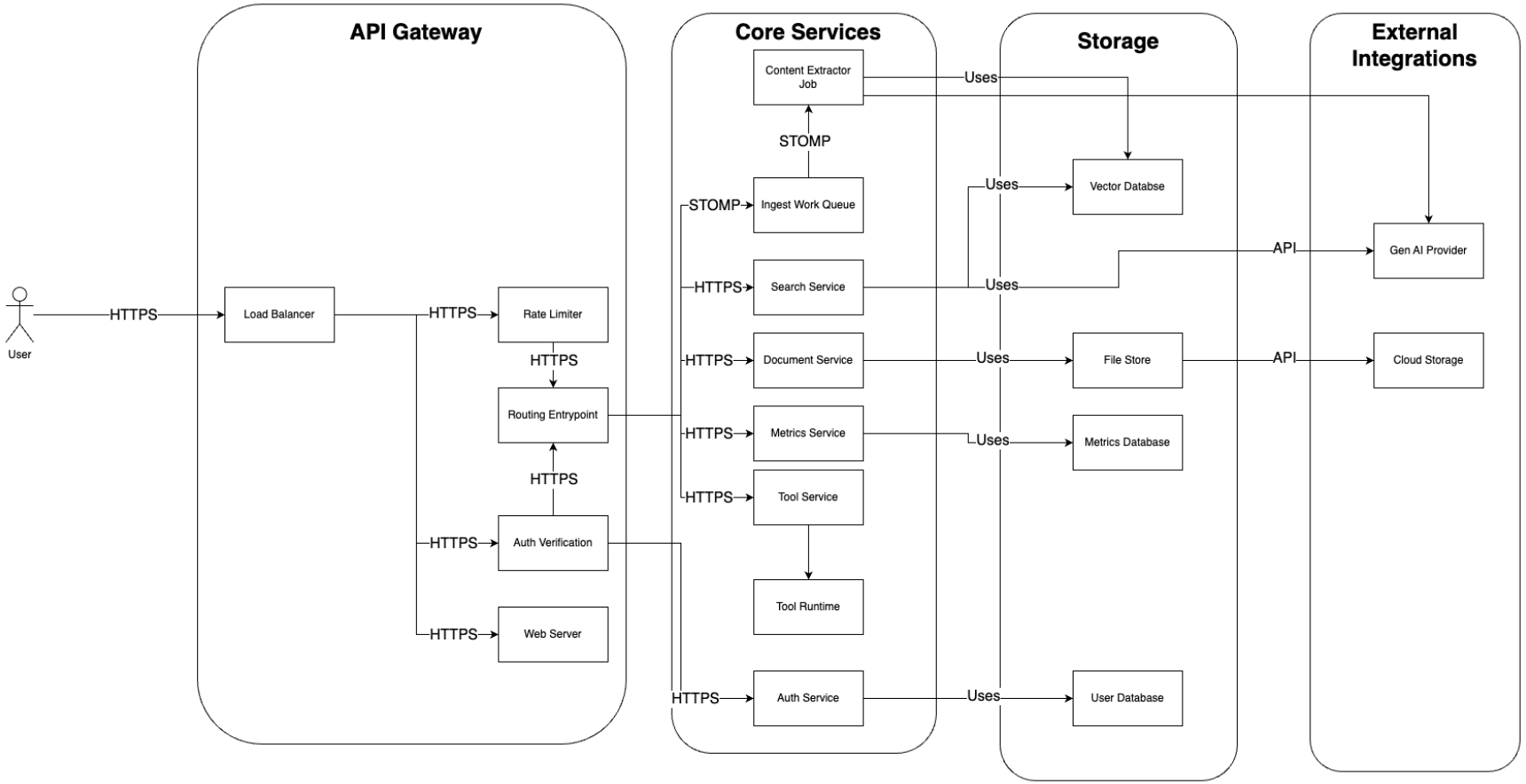

Figure 11: Component Diagram

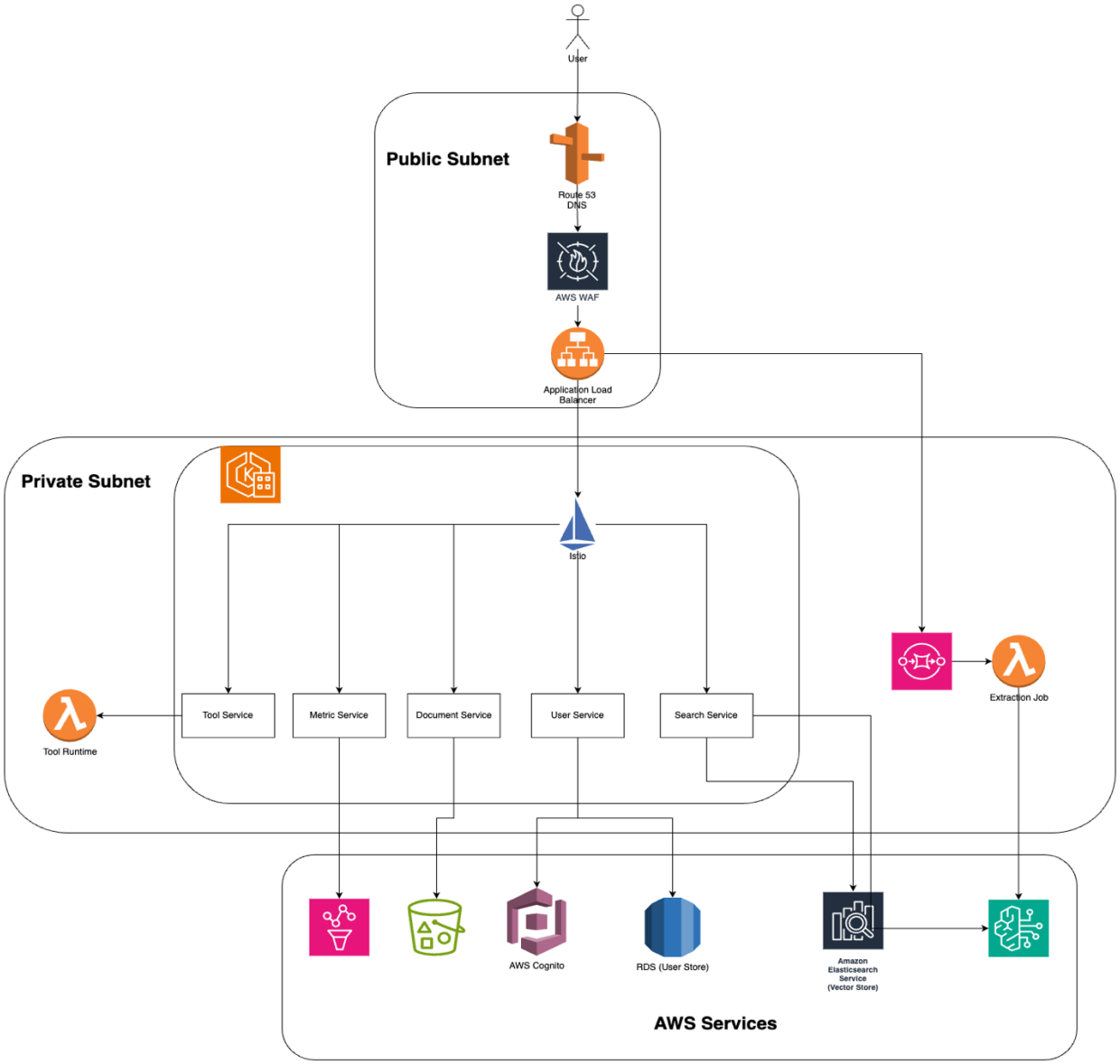

Figure 12: Deployment Diagram

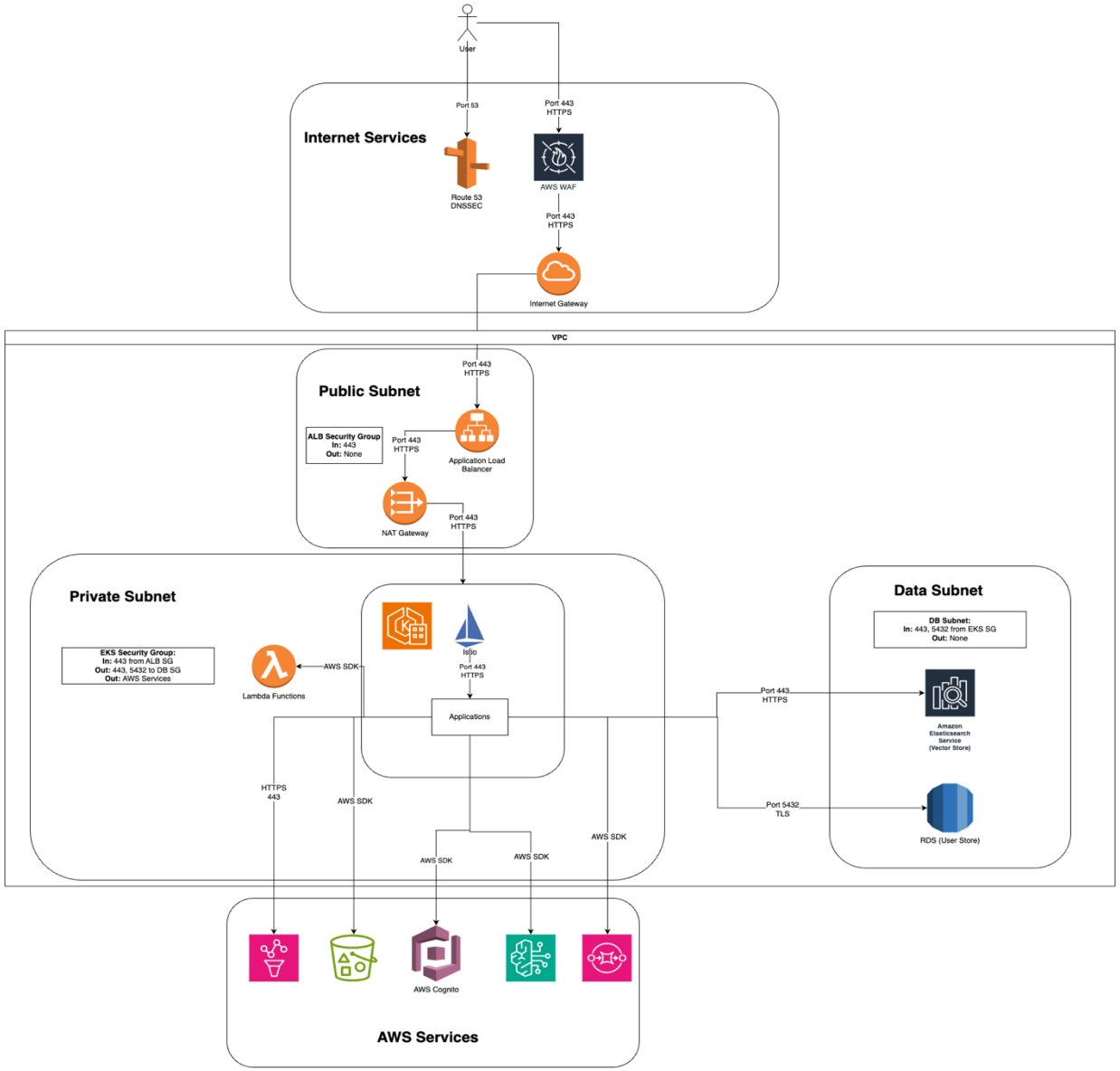

Figure 13: Security Diagram

Design Patterns

GRASP:

GRASP was used in the layout of the classes. The first principle used was the creator pattern. This is handled by the controllers which have all the required information to instantiate the primary classes of the system. This was due to the controllers already having the required information needed to construct the classes as well as the relatively flat class hierarchy. Next the controller pattern was used which separates each use case into a controller, this ensures high cohesion and low coupling by decoupling the classes from any user interface as well having only the required methods and fields to handle that particular use case. Indirection was also utilized in the form of repository interfaces and external system functionality such as Generative AI, or Cloud storage. This allows for loose coupling with those systems and evolution of multiple different backend implementations that can accommodate different providers. The repositories are also an example of pure fabrication, these classes were constructed to ensure that the other classes here highly cohesive and loosely coupled. These classes would contain any database logic removing that code from the higher-level classes. This makes it easier to maintain and more highly cohesive.

SOLID:

The first solid principle used is the Single-responsibility principle, each of the classes utilized are created for a single purpose. Repositories for storing and retrieving the data, controllers for accepting data from the UI and processing the use case and the core classes themselves to hold the data required to process the system. The open-closed principle and Liskov substitution principle were not applied. The system's class structure remains flat, this was intentional due to the small number of classes with limited crossover in terms of fields and methods. The interface segregation principle was observed. Each method for the interface is cohesive and focused on its scope. In particular, the repository or external service components provide methods which will almost always be used together. The dependency inversion principle was observed via these interfaces as well, the classes the controllers depend on are interfaces, this allows for the high-level interface to depend on an abstraction not a concrete implementation.

Gang of Four:

From the Gang of Four, the singleton pattern was utilized, each of the interface classes including the repositories, auth service, storage provider and gen ai gateway should use singletons. These classes are more expensive to instantiate due to their reliance on

sockets for external communication, therefore having a singleton allows for optimal use of these resources. The bridge pattern was also used for working with the gen ai provider. Each provider may have a different API or mechanism for how authentication works. The generic interface allows for adaptation to each of these without change to the primary application code. The tool implementation is a form of the command pattern. This includes a set of instructions (the tool) and a set of parameters to be executed within the tool's code. This allows for the tool to be processed in a separate sandbox and not interfere with other user's code or security.

Microservice:

This uses several microservice design patterns, the first is the API Gateway pattern. While it does not AWS API gateway it does use Application Load Balancer (ALB) and Web Application Firewall (WAF) to perform the same pattern. It uses the ALB as an entry point which filters traffic and balances it between N Kubernetes clusters, where the routing is performed. Meanwhile, WAF provides security filtering to the applications entry point. These also both serve as ambassadors for the ambassador pattern. Next, it follows the Database per Service Pattern, where each service has its own database, and no hidden interfaces exist between them via the database layer. The document service uses S3, Search Service uses OpenSearch, User Service uses RDS, and the Metric Service Prometheus. Next, it uses an event driven pattern, the SQS queue along with the lambda functions for processing provides ad hoc and independent scaling not coupled to the rest of the system. While not diagramed, support for the circuit breaker pattern could easily be added at the service level when interacting with Bedrock/Gen AI as well as the other AWS service interactions. Within Kubernetes there is also service discovery, this is built in and encrypted using Istio, which provides the service mesh. Kubernetes allows for pods to reach each other through their DNS names within the cluster and load balances between replicas. Finally, Sagas could be used to coordinate the orchestration in the document service as it: Uploads to S3, queues the extraction job, generates the embeddings and stores the resulting vectors in OpenSearch and performs rollback on failure.

Scalability:

100 concurrent users:

100 users could be achieved with the initial design. This would be achieved with a few replicas of each service pod in the EKS cluster and a single primary RDS server with a replica. OpenSearch will use 2-3 data nodes for vector storage and WAF and an application

load balancer can be used to balance requests between the EKS clusters. The caching will ensure that API times are minimal, and flexibility exists to grow the data nodes for OpenSearch and RDS if search times need to be improved.

To achieve an RTO of less than 4 hours and RPO of 1 hour, and maintain a simple architecture, infrastructure as code (IaC) will be used in the form of a CDK application. This will allow the environment to be more portable across regions. This will be leveraged by applying this architecture to both regions, with an active/passive model. The passive environment will contain a read replica of RDS, have S3 multi-region support and receive snapshots of the OpenSearch cluster every 30 minutes. This ensures < 1 hour RPO and RTO, should be on the order of minutes. Additional nodes can be added dynamically due to EKS Fargates dynamic nature, and vertical scaling of the RDS and OpenSearch nodes can be utilized to keep costs manageable. A Route 53 DNS record with health checks can be used to perform the swap between the two regions without relying on a single point of failure load balancer. This swap can also use a Lambda script to automatically scale up the resources within the Kubernetes cluster and vertically scale OpenSearch and RDS to allow for higher performance without needing to re-replicate the data.

10,000 concurrent users:

10,000 users have significantly higher needs. An ElasticCache cluster will be employed to provide cross AZ rapid caching of commonly accessed documents and search terms/results. RDS will utilize vertical scaling to achieve the throughput required. Read replicas will be employed to ensure performance. OpenSearch will add additional nodes to improve read performance, and vertical scaling to ensure that ingestion is timely. OpenSearch utilizes a single active node for writes which necessitates vertical scaling. With multiple EKS clusters spread across multiple AZs a load balancer will distribute evenly between the 3 sites and autoscaling policies within Kubernetes will be used to ensure performance is maintained. The additional caching and scalability will ensure that with the exception of RDS all services in the architecture can be horizontally scaled to meet demand. This with the utilization of a cache will ensure high performance for the API.

An RTO of less than 4 hours and a RPO of 1 hour is achieved by default in this architecture within a region, to achieve it across regions a similar strategy to the 100-user architecture can be employed, a scaled down secondary region can receive data through an async replication of RDS, and snapshots of S3 and OpenSearch just as in the 100 user architecture. When a failure occurs a lambda script can automatically scale these up and route 53 can detect the health failure and begin sending new DNS records to clients.

100,000 concurrent users:

The same EKS architecture with multiple clusters spread across AZs, will be enhanced to be spread across multiple regions. Each of these regions will contain an ElasticCache cluster with region specific relevant data. Global Accelerator will be used along with Route 53 to ensure that traffic is spread evenly between these EKS clusters. RDS will be replaced with DynamoDB Global, and S3 Cross-Region replication will be used for relevant documents. The Transition to DynamoDB was due to the requirement of multiple regions in this architecture. RDS would require all writes to be sent back to the primary region and replicated back out. This has loose consistency guarantees, on the other hand, DynamoDB does not have this issue and can function as a multi-write cluster and can use Dax for caching. This will enable the high performance required across multiple regions. Given the inherent multi-region support in this architecture no specific needs are required to achieve an RTO of 4 hours and an RPO of less than 1 hour. This system is designed to be global by default and failures will be handled by routing more data to another region and that region will employ its autoscaling to achieve demand, the biggest risk is OpenSearch due to the slow Horizontal scalability.

Cost Management

Primary Region Costs

Service	Monthly ($)	Configuration summary
Amazon Simple Queue Service (SQS)	0	Standard queue requests (1 million per month)
AWS Lambda	326.67	Architecture (x86), Architecture (x86), Invoke Mode (Buffered), Amount of ephemeral storage allocated (512 MB), Number of requests (1000000 per month)
Amazon RDS for PostgreSQL	600.7	Storage amount (100 GB), Storage volume (General Purpose SSD (gp3)), Nodes (1), Instance Type (db.m4.xlarge), Utilization (On-Demand only) (100 %Utilized/Month), Deployment Option (Multi-AZ), Pricing Model (OnDemand), Total Size of Backup Processed for Export (GB) (100 per month)
Amazon Managed	384.2312	Average Number of Dashboard users per day (10), Number of Prometheus rules (10), Average number of queries per day

Service for Prometheus		per dashboard user (2400), Average samples per query for Monitoring queries (100000), Average samples per query for Alerting queries (100000), Average active time series (150000), Total Monitoring Samples (73008000000), Total Alerting Samples (43800000000), Number of collectors (1)
Application Load Balancer	28.11	Number of Application Load Balancers (1)
Amazon OpenSearch Service	1382.73	Number of instances (3), Storage for each Amazon OpenSearch Service instance (General Purpose SSD (gp3)), UltraWarm storage cost (0), Number of nodes (0), Instance type (ultrawarm1.large.search), Utilization (On-Demand only) (100 %Utilized/Month), Pricing strategy (OnDemand), Nodes (3), Instance type (c4.2xlarge.search), Utilization (On-Demand only) (100 %Utilized/Month), Instance Node Type (Compute optimized), Storage Type (EBS Only), Pricing strategy (OnDemand), Nodes (0), Instance type (r5.2xlarge.search), Utilization (On-Demand only) (100 %Utilized/Month), Instance Node Type (Memory optimized), Storage Type (EBS Only), Pricing strategy (OnDemand), Managed storage size (1000 GB), Storage amount per volume (gp3) (200 GB)
Amazon Cognito	5	Optimization Rate for Token Requests (0), Optimization Rate for App Clients (0), Advanced security features (Enabled), Number of monthly active users (MAU) (100)
S3 Standard	23.04	S3 Standard storage (1000 GB per month), GET, SELECT, and all other requests from S3 Standard (100000)
Data Transfer	0	
Amazon EKS	73	Number of EKS Clusters (1), Number of hybrid nodes (1 per month)
Amazon EC2	424.448	Tenancy (Shared Instances), Operating system (Linux), Workload (Consistent, Number of instances: 4), Advance EC2 instance (t4g.xlarge), Pricing strategy (On-Demand Utilization: 100 %Utilized/Month), Enable monitoring (disabled), EBS Storage amount (100 GB), DT Inbound: Not selected (0 TB per month), DT Outbound: Not selected (0 TB per month), DT Intra-Region: (0 TB per month)

Failover Region base costs

Service	Monthly($)	Configuration summary

AWS Fargate	0	Operating system (Linux), CPU Architecture (x86), Average duration (1 minutes), Number of tasks or pods (01 per month), Amount of ephemeral storage allocated for Amazon ECS (20 GB)	
Application Load Balancer	22.27	Number of Application Load Balancers (1)	
Amazon OpenSearch Service	334.96000000000004	Number of instances (2), Storage for each Amazon OpenSearch Service instance (General Purpose SSD (gp3)), UltraWarm storage cost (0), Nodes (2), Instance type (r4.large.search), Utilization (On-Demand only) (100 %Utilized/Month), Instance Node Type (Memory optimized), Storage Type (EBS Only), Pricing strategy (OnDemand), Nodes (0), Instance type (r5.2xlarge.search), Utilization (On-Demand only) (100 %Utilized/Month), Instance Node Type (Memory optimized), Storage Type (EBS Only), Pricing strategy (OnDemand), Number of nodes (0), Instance type (ultrawarm1.large.search), Utilization (On-Demand only) (100 %Utilized/Month), Pricing strategy (OnDemand), Storage amount per volume (gp3) (200 GB)	
S3 Standard	23	S3 Standard storage (1000 GB per month)	
Data Transfer	0		
Amazon RDS for PostgreSQL	80.85	Storage amount (100 GB), Storage volume (General Purpose SSD (gp2)), Nodes (1), Instance Type (db.t4g.medium), Utilization (On-Demand only) (100 %Utilized/Month), Deployment Option (Single-AZ), Pricing Model (OnDemand)	

Cost Rationale and Discussion

These represent rough estimates that are realistic for a relatively moderate user count, this makes a few modifications to the starting architecture to optimize cost including using EKS with EC2 nodes rather than Fargate due to Fargate's unfavorable cost structure for long running tasks. This achieves the goals of an 8 hour RTO and 2 hour RPO as well as a less than 24 hour RTO and 4 hour RPO while still having less than 100ms response times. The one modification between the various tiers of RTO/RPO is the frequency of the snapshots between S3 and OpenSearch between the regions. This may affect the cost marginally over the course of a month. However, given the low cost of the passive site ($461/month) it is unlikely to be worthwhile to develop multiple tiers of

RTO/RPO recovery. This includes automatic failover within 5 minutes of detection. AWS Route 53's health check will identify a failure of the primary site and invoke a lambda to quickly vertically scale the OpenSearch, RDS instances as well as the Kubernetes workload. The data is being replicated regularly so this just involves assigning additional compute and is therefore extremely quick in terms of RTO. For large and unexpected increases in requests, this passive environment can be spun up as an active environment. The limitation of that approach is that the secondary site cannot process ingestion, and any ingest requests must go to the primary region and replicated back to the backup region therefore being eventually consistent. However, given the RAG use case this is likely still helpful compared to reduced performance. Finally, this cost can be further tuned using an AWS cost savings plan, reserved instances or spot instances.

Sample class

See: https://github.com/michaelblithe/raggady/tree/main/code for included classes and schemas.