SMU
SINGAPORE MANAGEMENT UNIVERSITY

School of
Computing and
Information Systems

SENTINEL

CS480 Final Report
Project Sentinel

Team Sentinel

30 November 2025

Team Members:
- Muhammad Haris bin Mohd Shariff (harisms.2022@scis.smu.edu.sg) – Software Engineer / Lead
- Ang Rui Yan (ruiyan.ang.2022@scis.smu.edu.sg) – Software Engineer / Point of Contact
- Chloe Ong Weiting (chloeong.2022@scis.smu.edu.sg) – Security Engineer
- Khoo Jing Ren (jrkhoo.2022@scis.smu.edu.sg) – Security Engineer
- Liao Weishun (weishunliao.2022@scis.smu.edu.sg) – Software Engineer
- Lim Wei Jie (weijielim.2022@scis.smu.edu.sg) – Software Engineer
- Oh Sheow Woon (swoh.2022@scis.smu.edu.sg) – Software Engineer

Faculty Supervisor:
- Muhammad Mohsin

Sponsor and/or Clients:
Morgan Stanley
- Bai, Kyrie (Kyrie.Bai@morganstanley.com) - Security Development (Vice President) - Project's Product Owner
- Ng, Kevin K (Kevin.K.Ng@morganstanley.com) - Commodities Technology (Vice President) - Project's Mentor / Supervisor

Table of contents

1. Introduction **4**
- 1.1. Our Sponsor 4
- 1.2. The Problem 4
- 1.3. Requirement Refining 4
- 1.4. Finalized Requirements 5

2. Project Management **5**
- 2.1. Epics 7
- 2.2. User Stories 7
- 2.3. Sprint Details 8
- 2.4. Team Roles and Contributions 9
- 2.5. Sponsor Interaction 10
- 2.6. Supervisor Interaction 10
- 2.7. Telegram PR Bot 11

3. Sentinel Overview **12**
- 3.1 System Architecture 12
- 3.2 Core Business Entities 14

4. Sentinel Frontend **15**
- 4.1. React, TypeScript and Vite 15
- 4.2. Clean architecture and organisation 15
- 4.3. Authentication 16
- 4.4. State Management 16
- 4.5. Data Validation 17
- 4.6. Integration Testing 17
- 4.7. CI 17
- 4.8. Pages 18
  - 4.8.1. Login 18
  - 4.8.2. Dashboard 18
  - 4.8.3. Chat 19
- 4.9. Internationalisation 20

5. Sentinel Backend (Bastion) **20**
- 5.1. Language and dependencies 20
- 5.3 Orchestration of internal services 21
  - 5.3.1 Integration with Sentinel Agent 21
  - 5.3.2 Integration with IGM system (Midpoint) 21
  - 5.3.3 Integration with Authentication Provider (Keycloak) 22
  - 5.3.4 Integration with Documentation Source (Bookstack) 22
- 5.4 Chat services 22
  - 5.4.1 HTTP Server 22
  - 5.4.2 Persistence (audit trail) 22
- 5.5 Continuous Integration 23

6. Sentinel Agent **23**
- 6.1. LangGraph, Python and OpenAI 24
- 6.2. System Architecture 24

6.2.1. Inter Process Communication 24
6.2.2. ReAct Agent 25
6.2.3. System Messages 25
6.2.3. RAGFlow pre fetch 26
6.2.4. Structured Outputs 26
6.2.5. Guardrails 26
6.3. Model Testing 28
6.4. System Message Testing 28
6.5. Guardrails Testing 29
7. MidPoint MCP 30
7.1. Overview 30
7.2. HTTP Server and Tool Contract Design 30
7.3. Role Search and MidPoint-Specific Implementation Details 31
7.4. Integration with LangGraph 31
7.5. Testing and Validation 32
7.6. Continuous Integration and Supply Chain Security 32
8. Sentinel Simulated Environment 32
8.1. Overview 32
8.2. MidPoint Identity Governance and Administration System 33
8.3. Keycloak Authentication and Identity Bridging 33
8.4. Grafana Application Integration 34
8.5. Bookstack Knowledge Repository and RAGFlow Integration 34
8.6. Document Retrieval 36
8.6.1. RAGFlow 36
8.6.2. Reranker 37
9. Future Work 40
9.1. Integration with Production IGA and Enterprise Systems 40
9.2. Expansion of Knowledge Sources and RAG capabilities 40
9.3. Advanced Retrieval Strategies Beyond Reranking 40
9.4. Advanced Guardrails and Continuous Risk Monitoring 40
9.5. Evaluating with Real Users and Human in the Loop Workflows 41
9.6. Policy Validator Agent 41

1. Introduction

1.1. Our Sponsor

Morgan Stanley is a leading global financial services firm, operating in over 40 countries and serving clients worldwide, including corporations, governments, institutions and individuals. The firm operates through three main business segments, institutional securities, wealth management and investment management.[1] As of 2025, Morgan Stanley reported record revenues of $18.2 billion in the third quarter and total client assets of nearly $9 trillion, reflecting its strong position in global financial markets.[2]

1.2. The Problem

Morgan Stanley's teams operate across multiple modular business functions, resulting in information disentanglement. Information on the required policies or roles is fragmented across different systems and documentation sources, making it difficult for requestors to determine what access they need or how to obtain it. Non-technical teams struggle to navigate the process, leading to delays or unnecessary escalations. These delays often result in the granting of broad and higher levels of privilege permissions. This violates the principle of least privilege and increasing security risks.

This project addresses the bottleneck by providing an AI-driven, policy-aware assistant that can identify the cause of blocked requests, guide users towards compliant solutions and reduce the friction to request without expanding privilege scope.

1.3. Requirement Refining

The project requirements were iteratively refined though multiple meetings and sprint reviews with our sponsor, Kyrie Bai. Originally, the project scope included two proof-of-concepts (POCs): (1) Access Remediation Agent, and (2) Policy Validator Agent. After further discussion with the sponsor, we collectively agreed to prioritise POC 1, the access remediation agent, as the core deliverable for the project. POC 2, the policy validator agent, will not be part of the project deliverables. This decision would allow our team to focus our efforts on tackling the most pressing issue that the sponsor is currently facing.

Another point that we were able to clarify was what the agent's role should be. Based on our discussion, the agent should be able to assist users in drafting resource access requests, while still enforcing human approval before submission.

It was clarified that replicating the sponsor's internal Identity Governance and Administration (IGA) system was unnecessary. While mocking the IGA interactions was initially suggested, the team proposed building a simulated IGA environment to allow Sentinel's Agent to be tested under realistic conditions. The environment was designed to be modular, enabling the sponsor to later integrate their

---

[1] "Business Segments." Morgan Stanley. Accessed November 2025. https://www.morganstanley.com/about-us-governance/businesssegments.

[2] Bautzer, Tatiana, and Arasu Kannagi Basil. "Morgan Stanley Profit Beats Estimates on Dealmaking Boost, Shares Soar." Reuters, October 16, 2025. https://www.reuters.com/business/finance/morgan-stanley-profit-jumps-investment-banking-revival-2025-10-15/.

internal Retrieval Augmented Generation (RAG) systems and connect Sentinel through a Model Context Protocol (MCP) bridge to their actual IGA infrastructure with minimal changes.

For temporary and permanent access, it was decided that we would be handling temporary access for users, as that was the demographic that would benefit the most from Sentinel, as well as aligning to the principle of least privilege. This is because temporary users are much less likely to be familiar with what permissions they need, and will most likely face more issues requesting access.

Last but not least, we agreed on including a knowledge base which the agent can draw from, in order to support the answers returned to users. This would allow the model to have access to up to date information when replying to users.

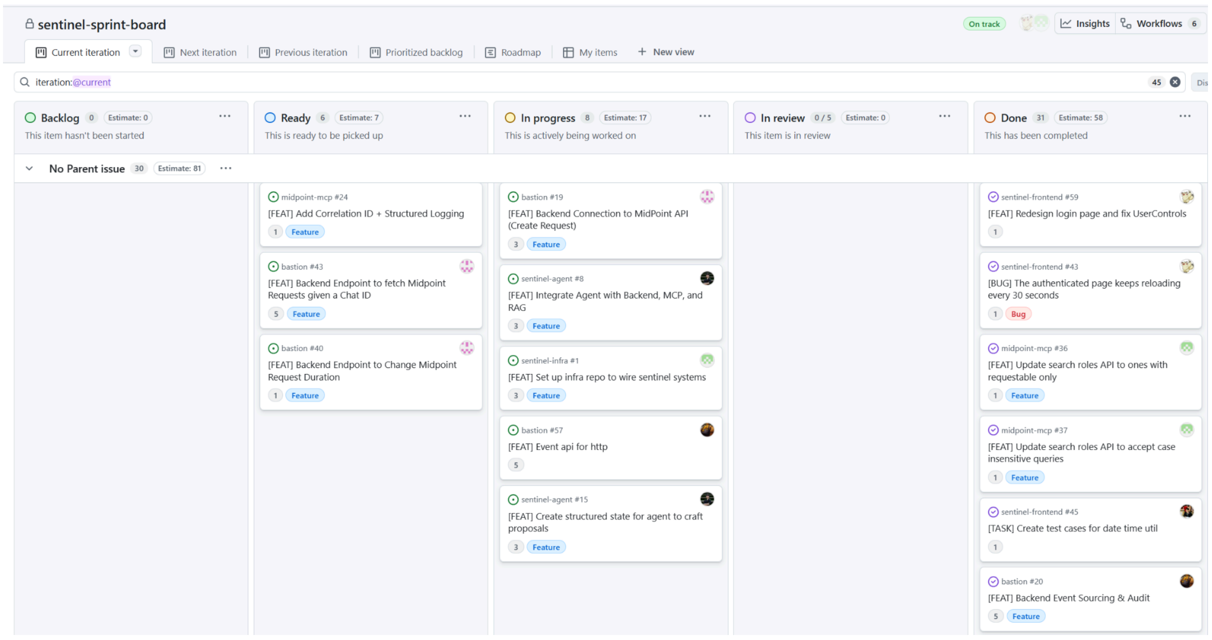
1.4. Finalized Requirements

After much discussion our finalised main deliverable is an end to end access remediation AI application, which assists Morgan Stanley employees in navigating complex policy documents and internal references, and the list of requirements for the system is as follows:

1. Allows users to interact with the AI agent to clarify any queries relating to permissions and access to Morgan Stanley systems
2. AI Agent should be able to reply with references to up to date information based on company internal documentation
3. Users should be able to re-visit sessions and continue conversations from where they left off
4. AI Agent behavior should be restricted based on cybersecurity principles
5. Any additional systems which Sentinel needs to interact with can be simulated, using alternatives of our choice (Eg, Document store, IGA system, etc.)
6. Systems should be modular, allowing them to be switched out and independently maintained.
7. Systems should be infrastructure agnostic, allowing them to be deployed in any environment.
8. Given the nature of the company, all interactions and events should be logged for auditing and monitoring
9. System functionality should be clearly documented to allow for future extension of the prototype if needed

2. Project Management

Our team adopted the Scrum methodology, an agile framework, for project management. The entire Sentinel application is managed using Github Projects, which is linked directly to a Github Organisation for Sentinel. We chose Github Projects as it allows us to easily manage the individual repositories for each epic from one central project page, with built in integrations directly with the repositories themselves. This approach enabled transparent workflow visualization and efficient task assignment.



sentinel-sprint-board

iteration:@current 45

Backlog 0 Estimate: 0
This item hasn't been started

Ready 6 Estimate: 7
This is ready to be picked up

In progress 8 Estimate: 17
This is actively being worked on

In review 0 / 5 Estimate: 0
This item is in review

Done 31 Estimate: 58
This has been completed

No Parent issue 30 Estimate: 81

Ready:

- midpoint-mcp #24 — [FEAT] Add Correlation ID + Structured Logging — 1 Feature
- bastion #43 — [FEAT] Backend Endpoint to fetch Midpoint Requests given a Chat ID — 5 Feature
- bastion #40 — [FEAT] Backend Endpoint to Change Midpoint Request Duration — 1 Feature

In progress:

- bastion #19 — [FEAT] Backend Connection to MidPoint API (Create Request) — 3 Feature
- sentinel-agent #8 — [FEAT] Integrate Agent with Backend, MCP, and RAG — 3 Feature
- sentinel-infra #1 — [FEAT] Set up infra repo to wire sentinel systems — 3 Feature
- bastion #57 — [FEAT] Event api for http — 5
- sentinel-agent #15 — [FEAT] Create structured state for agent to craft proposals — 3 Feature

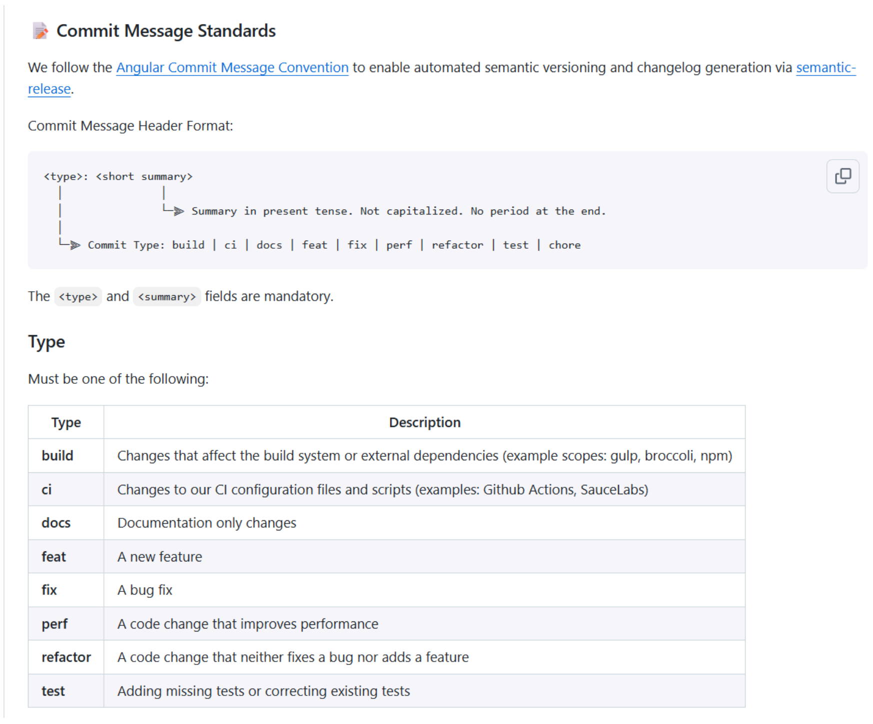
Done:

- sentinel-frontend #59 — [FEAT] Redesign login page and fix UserControls — 1
- sentinel-frontend #43 — [BUG] The authenticated page keeps reloading every 30 seconds — 1 Bug
- midpoint-mcp #36 — [FEAT] Update search roles API to ones with requestable only — 1 Feature
- midpoint-mcp #37 — [FEAT] Update search roles API to accept case insensitive queries — 1 Feature
- sentinel-frontend #45 — [TASK] Create test cases for date time util — 1
- bastion #20 — [FEAT] Backend Event Sourcing & Audit — 5 Feature

Fig 2.1: Sentinel sprint board on Github Projects

For each repository, we adopted trunk based development as our branching strategy. Under this approach, every team member worked on short lived feature branches that were merged back as soon as the feature was verified and reviewed by the main team member in charge of the respective area.

We also ensured alignment across all repositories by using the same contribution guidelines, outlined within the Sentinel organization github page, which dictates our branching strategy, pull request process and commit practices.



Commit Message Standards

We follow the Angular Commit Message Convention to enable automated semantic versioning and changelog generation via semantic-release.

Commit Message Header Format:

```
<type>: <short summary>
  │              │
  │              └─⫸ Summary in present tense. Not capitalized. No period at the end.
  │
  └─⫸ Commit Type: build | ci | docs | feat | fix | perf | refactor | test | chore
```

The `<type>` and `<summary>` fields are mandatory.

Type

Must be one of the following:

Type	Description
build	Changes that affect the build system or external dependencies (example scopes: gulp, broccoli, npm)
ci	Changes to our CI configuration files and scripts (examples: Github Actions, SauceLabs)
docs	Documentation only changes
feat	A new feature
fix	A bug fix
perf	A code change that improves performance
refactor	A code change that neither fixes a bug nor adds a feature
test	Adding missing tests or correcting existing tests

Fig 2.2: Excerpt of contribution guidelines

2.1. Epics

Our “epics” are large objectives representing major deliverables, which can be broken down into smaller, actionable stories. We separated “epics” logically by their role within Sentinel as a whole, with each having clearly defined functionality that fulfils a part of the product requirements.

Sentinel comprises of 5 main “epics”, which will be elaborated upon more in later sections:

1. Sentinel Frontend
2. Bastion (Backend)
3. Sentinel Agent
4. MidPoint MCP
5. Sentinel Simulated Environment

2.2. User Stories

Our user stories provide detailed, actionable tasks from the perspective of system users or agents, with explicit acceptance criteria for the POC context.

User stories are classified into a few categories, which is represented in the title of the user story:

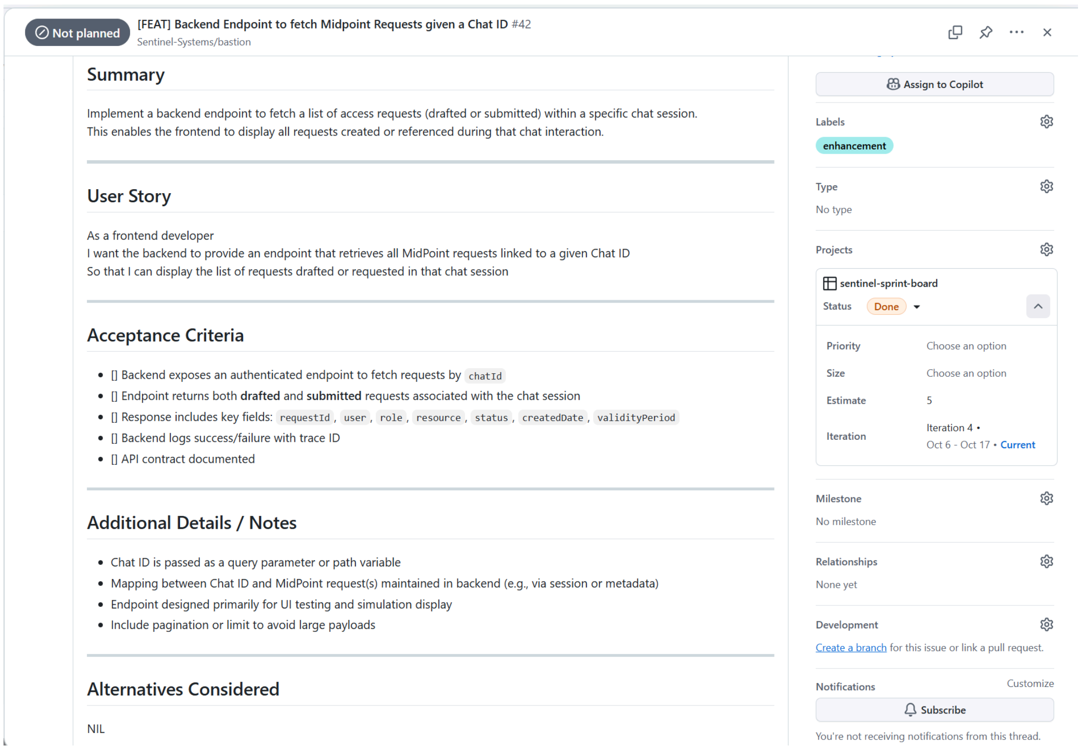
1. FEAT: A new feature adding functionality to Sentinel
2. TASK: A user story that does not explicitly add functionality to Sentinel, but is required for another user story
3. BUG: Issues identified within existing functionality

When written, each user story is structured according to the following template (*Fig 1.1*):

1. Summary: Provides a short description of the entire story
2. User Story: Describes what a specific user wants to do, and their expectations
3. Acceptance Criteria: Defines clear conditions that must be met for the story to be considered completed
4. Additional Details/Notes: Outlines specific implementation restrictions or assumptions we may have, or any extra information required for implementation
5. Alternatives Considered: Describes alternative methods to fulfil the objective of the user story

Each user story is also assigned story points, which we collectively decide during our sprint planning meetings, using scrum poker. To align our story point scales, our team uses a scale based on the fibonacci sequence, and linked to how long we believe the story will take to complete:

- 1 Point: 0.5 day - 1 day
- 2 Points: 1 day - 2 days
- 3 Points: 2 days - 3 days
- 5 Points: 3 days - 5 days
- 8 Points: 5 days - 8 days
- 13 Points: 8 days - 10+ days
- 21 Points: More than 1 sprint (14 days)


Not planned [FEAT] Backend Endpoint to fetch Midpoint Requests given a Chat ID #42
Sentinel-Systems/bastion

Summary

Implement a backend endpoint to fetch a list of access requests (drafted or submitted) within a specific chat session.
This enables the frontend to display all requests created or referenced during that chat interaction.

User Story

As a frontend developer
I want the backend to provide an endpoint that retrieves all MidPoint requests linked to a given Chat ID
So that I can display the list of requests drafted or requested in that chat session

Acceptance Criteria

- [] Backend exposes an authenticated endpoint to fetch requests by `chatId`
- [] Endpoint returns both **drafted** and **submitted** requests associated with the chat session
- [] Response includes key fields: `requestId`, `user`, `role`, `resource`, `status`, `createdDate`, `validityPeriod`
- [] Backend logs success/failure with trace ID
- [] API contract documented

Additional Details / Notes

- Chat ID is passed as a query parameter or path variable
- Mapping between Chat ID and MidPoint request(s) maintained in backend (e.g., via session or metadata)
- Endpoint designed primarily for UI testing and simulation display
- Include pagination or limit to avoid large payloads

Alternatives Considered

NIL

Labels: enhancement
Type: No type
Projects: sentinel-sprint-board
Status: Done
Priority: Choose an option
Size: Choose an option
Estimate: 5
Iteration: Iteration 4 • Oct 6 - Oct 17 • Current
Milestone: No milestone
Relationships: None yet
Development: Create a branch for this issue or link a pull request.
Notifications: You're not receiving notifications from this thread.

Fig 1.1: User Story Example

2.3. Sprint Details

Following the Scrum methodology, we split our development up into 2 week sprints, with each focusing on a specific area of development. We begin sprints with 3 goals:

1. Sprint 1: Sun, Aug 24 - Fri, Sep 5
   Research, Foundational building of knowledge and comprehension
2. Sprint 2: Sun, Sep 7 - Thu, Sep 18
   Finish the setup of the simulated environment (IAM, IDP, Wiki)
   Build the ingestion pipeline
   Begin backend agent workflows
3. Sprint 3: Sun, Sep 21 - Thu, Oct 2
   Agent integration with Backend, RAG & MCP
   Frontend integration with Backend
   Authentication for Frontend
4. Sprint 4: Sun, Oct 5 - Thu, Oct 16
   Report write up for mid-term presentation
   Preparation of Demo for mid-term presentation
   FE - BE - AGENT - MCP - MIDPOINT- GRAFANA flow
5. Sprint 5: Sat, Oct 18 - Fri, Oct 31
   Extend from mid-term
   Guardrail implementation
6. Sprint 6: Sun Nov 3 - Fri, Nov 21
   Reranker research, Final e2e integrations, Guardrail integration

On the day before starting each sprint, our team conducts a sprint planning meeting, deciding on which user stories to work on, estimate the efforts for each ticket, and allocate user stories to team members.

Considering the fact that our team members also have other commitments, we chose not to have daily standups, as each member may not be able to work on their tasks every day. In place of this, we conduct a mid sprint review on the middle Friday of the sprint, to check the progress on individual user stories and if there are any blockers which may prevent us from completing the user story. We also individually update each other on a day to day basis if we are facing any impediments, or if there are unexpected events.

On the day after each sprint, we conduct our sprint retrospective to understand the progress we made during the sprint, as well as to determine if there are any areas of improvement for the next sprint.

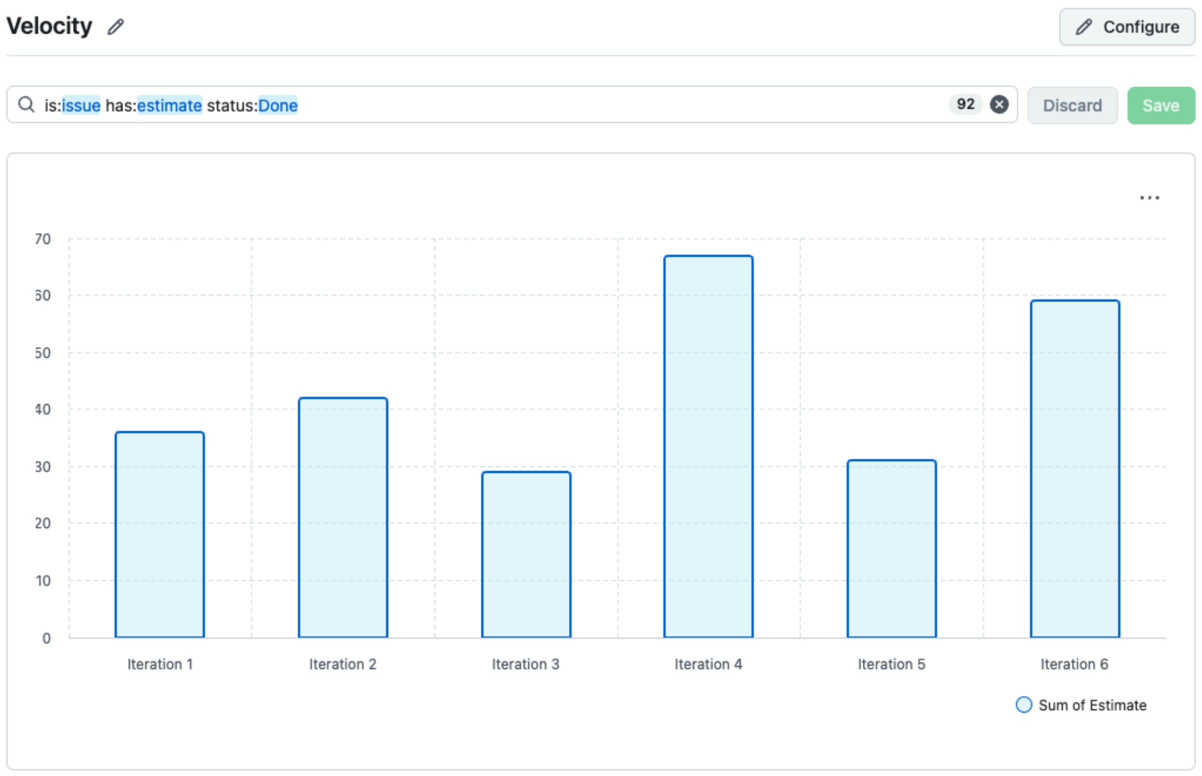


Fig 2.2: Sprint velocities

2.4. Team Roles and Contributions

Each team member focused on certain areas of development, but worked together to support each other as needed.

1. Sentinel Frontend: Rui Yan, Chloe
2. Bastion (Backend): Sheow Woon, Lucas
3. Sentinel Agent: Wei Jie, Jing Ren
4. MidPoint MCP: Haris
5. Sentinel Simulated Environment: Haris, Jing Ren, Chloe, Lucas

2.5. Sponsor Interaction

Throughout the duration of the project, we maintained consistent communication with our sponsor to ensure alignment of expectations and transparency in our progress. The sponsor and supervisor are updated after every sprint in email summarising our progress, tickets completed and any queries or issues we faced. On top of that, calls are scheduled in order to have direct feedback on our progress. The team also conducts mid-weekly refinements in order to update both the sponsor and supervisor on our mid-week progress within the sprint board itself.

For example, during our first iteration, we presented the Sentinel's frontend flow in Figma, depicting how the user would interact with the application, to determine if what we wanted to create was in line with the sponsor's expectations and requirements. Our team received positive feedback regarding this, and discussions moved towards how the frontend and backend would interact. Rui Yan, our main point of contact with the sponsor, also met in person with them to gather feedback on Sentinel's earlier prototypes in Figma and initial frontend implementation.

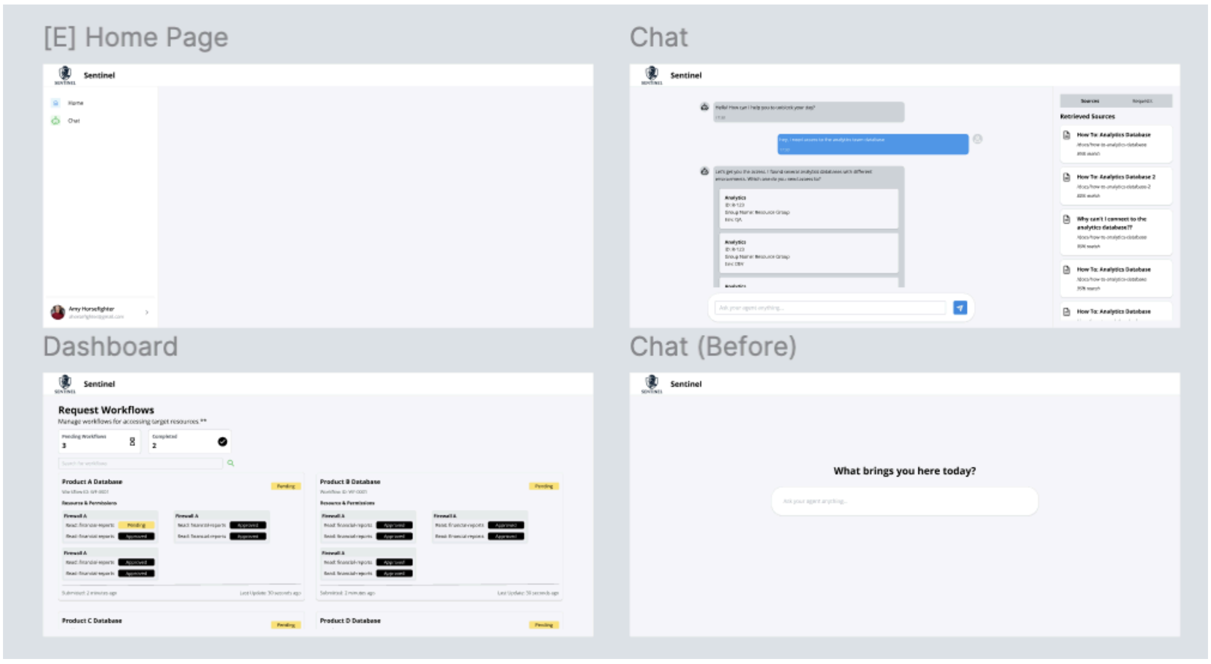
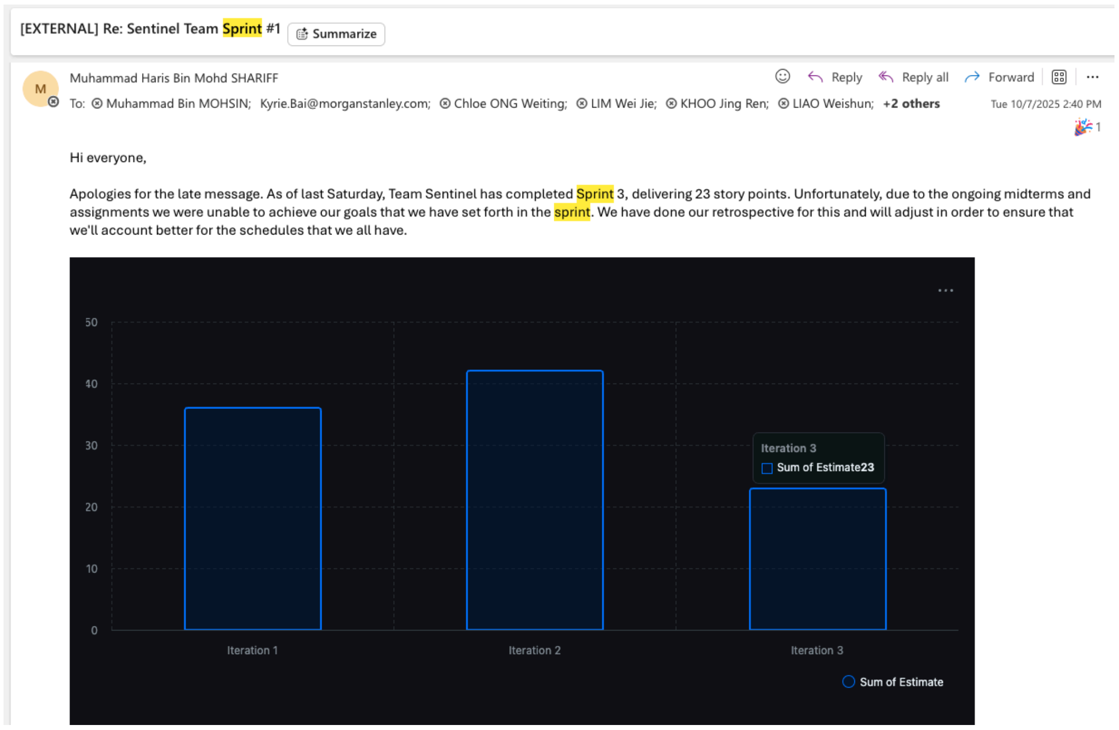


Fig 2.3: Figma prototype in week 2

Another example of feedback was when the sponsor encouraged research in the RAG reranker. As the reranker was an area of interest for the organization, this led us to plan for the future tickets on this.

2.6. Supervisor Interaction

The supervisor is updated after each sprint through both email and follow-up calls outlining progress, blockers and any queries raised during the sprint. Mid-week updates are shared via our GitHub Project boards. The supervisor has also expressed concerns about including the second POC in our submission, which led us to revisit the project scope and align with the sponsor on submitting a single POC instead. Throughout each week, the supervisor has also provided guidance on the functional flows to be implemented in the agent and the specific tests we should conduct to validate its performance.



[EXTERNAL] Re: Sentinel Team Sprint #1

Muhammad Haris Bin Mohd SHARIFF

To: Muhammad Bin MOHSIN; Kyrie.Bai@morganstanley.com; Chloe ONG Weiting; LIM Wei Jie; KHOO Jing Ren; LIAO Weishun; +2 others

Tue 10/7/2025 2:40 PM

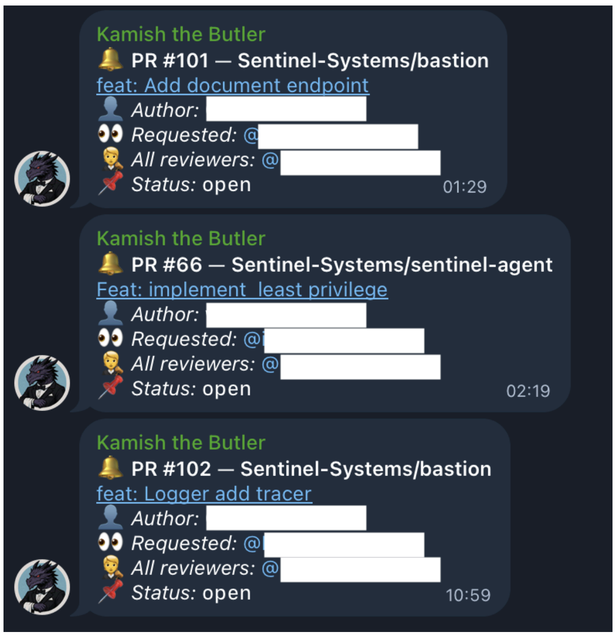
Hi everyone,

Apologies for the late message. As of last Saturday, Team Sentinel has completed Sprint 3, delivering 23 story points. Unfortunately, due to the ongoing midterms and assignments we were unable to achieve our goals that we have set forth in the sprint. We have done our retrospective for this and will adjust in order to ensure that we'll account better for the schedules that we all have.

Fig 2.4: Email Outreach every Sprint

2.7. Telegram PR Bot

The Telegram PR Bot is an internal automation tool that our team built to bridge GitHub and Telegram to streamline the code review process. When a new pull request (PR) is created or updated on GitHub, the bot is triggered via GitHub Actions. It automatically tags the assigned reviews with our mapping to the Telegram username and topic, and includes the direct link to the PR. This bot enhances internal developer experience by reducing context switching and improving visibility of ongoing reviews. Our team can be informed in real time through Telegram. This prevents overlooked reviews, and contributes to a more efficient and connected engineering workflow.



Kamish the Butler
🔔 PR #101 — Sentinel-Systems/bastion
feat: Add document endpoint
👤 *Author:* [redacted]
👀 *Requested:* @[redacted]
🤵 *All reviewers:* @[redacted]
📌 *Status:* open
01:29

Kamish the Butler
🔔 PR #66 — Sentinel-Systems/sentinel-agent
Feat: implement_least privilege
👤 *Author:* [redacted]
👀 *Requested:* @[redacted]
🤵 *All reviewers:* @[redacted]
📌 *Status:* open
02:19

Kamish the Butler
🔔 PR #102 — Sentinel-Systems/bastion
feat: Logger add tracer
👤 *Author:* [redacted]
👀 *Requested:* @[redacted]
🤵 *All reviewers:* @[redacted]
📌 *Status:* open
10:59

Fig 2.5: A screenshot of Telegram PR message by the bot, tagging the reviewers.

3. Sentinel Overview

3.1 System Architecture

Sentinel is designed as a modular and infrastructure-agnostic ecosystem that brings together multiple independent services into a unified access remediation workflow. The system architecture reflects a strong separation of concerns, where each component fulfills a distinct responsibility while contributing to the overall access remediation experience. This modular design ensures that individual services can be replaced, upgraded, or extended without affecting the larger system. The architecture also mirrors real enterprise environments, where identity governance, authentication, application access, and documentation exist across multiple platforms.[3]

At a high level, Sentinel consists of five major subsystems: the Sentinel Frontend, the Bastion Backend, the Sentinel Agent, the MidPoint MCP service, and the Sentinel Simulated Environment. These components communicate with one another using HTTP, gRPC, and Model Context Protocol (MCP), forming a cohesive workflow that can intake user prompts, retrieve contextual knowledge, query identity governance metadata, generate structured access proposals, and send them back through the frontend for display.

The system follows a clear operational flow. When a user accesses the Sentinel web application, authentication is performed through Keycloak, which issues the required tokens for secure interactions. Once authenticated, the user can create or open chat sessions with the agent. The frontend sends each new user message to Bastion, which manages persistence through its hybrid event-sourced database model. Bastion then forwards the message to the Sentinel Agent through a gRPC channel. The agent processes the request, retrieves information from RAGFlow or MidPoint MCP when necessary, and returns both the natural-language response and any structured proposal content. Bastion stores the results and streams the reply back to the frontend through a dedicated Server-Sent Events pipeline. The frontend renders the message to the user, offering a seamless and interactive experience.

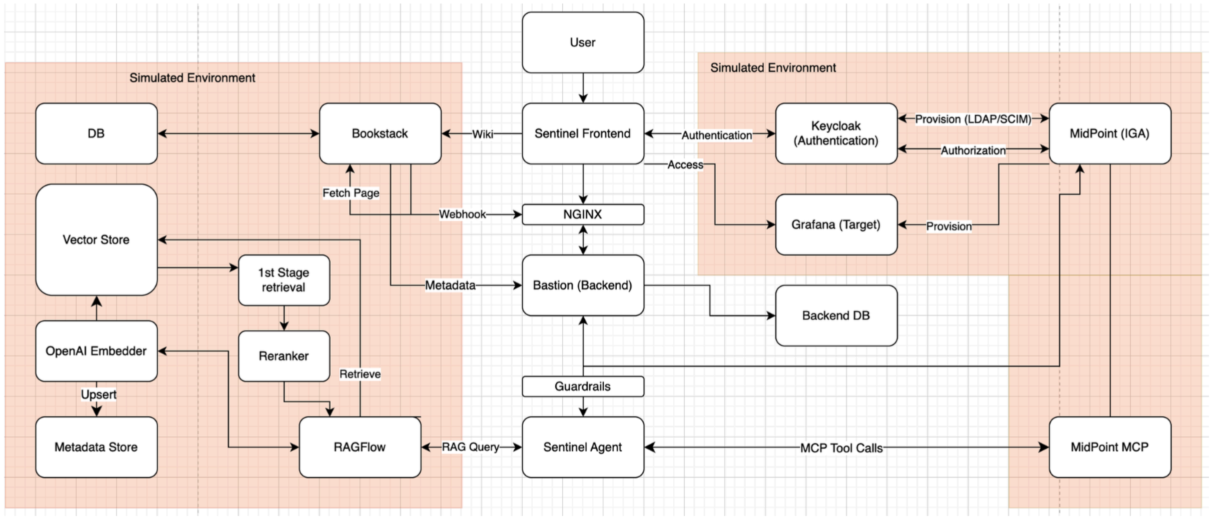


Fig 3.1.1: System Architecture

[3] Guntakandla, Anusha Reddy. "Modular Architecture: A Scalable and Efficient System Design Approach for Enterprise Applications." World Journal of Advanced Research and Reviews 26, no. 1 (April 30, 2025): 3114–26. https://doi.org/10.30574/wjarr.2025.26.1.1340.

The overall flow of a user interaction is as follows: (assuming user needs access to a resource)

1. Users open up the Sentinel WebApp
2. Users authenticate themselves to the system with Keycloak SSO
3. Users create a new chat and raises access related query
4. Sentinel Frontend sends access query to Bastion
5. Bastion stores the chat in a database, and forwards the access query to Sentinel Agent
6. Sentinel Agent receives the access query processes it
7. Sentinel Agent requests information from RAGFlow/Sentinel MCP as required
8. RAGFlow/Sentinel MCP returns results
9. Sentinel Agent uses access query, chat history and additional information to generate response
10. Bastion stores tool calls and response in database, and sends reply to Sentinel Frontend
11. Sentinel Frontend displays reply on user interface

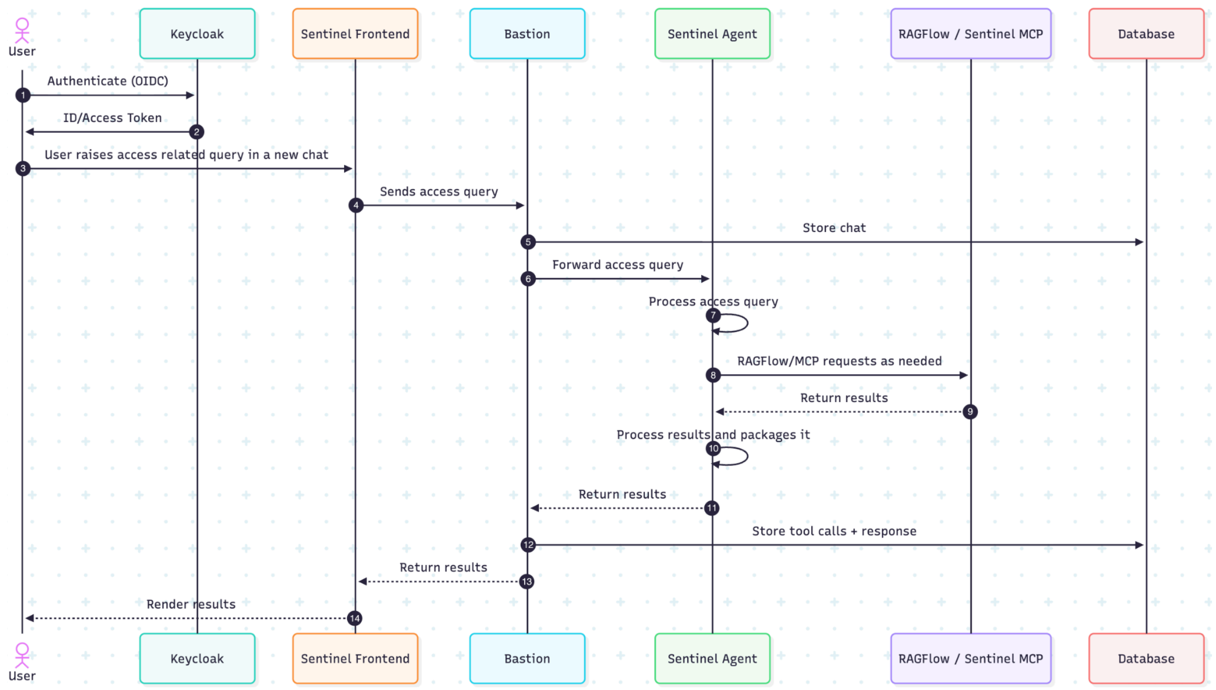


Fig 3.1.2: Agent request flow diagram

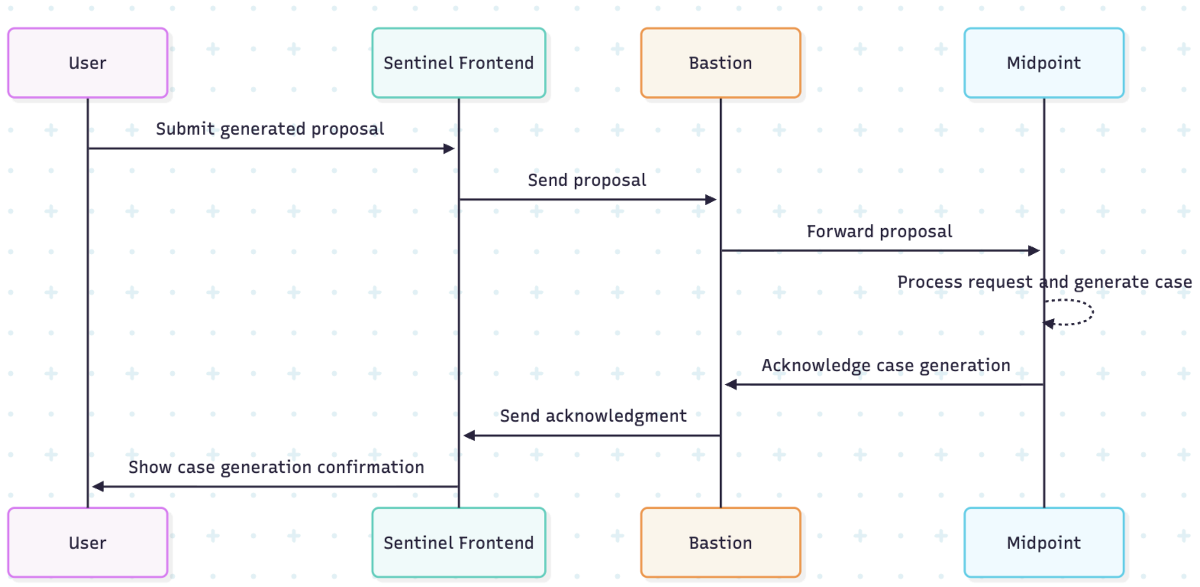


Fig 3.1.3: Proposal request flow diagram

3.2 Core Business Entities

To support consistent communication between services, Sentinel defines four core entities that serve as the backbone of the system's data model.

- Chat
  Represents a full conversation session between the user and the agent. It contains metadata such as session identifiers, creation timestamps, and ownership details. Chats persist across user visits, allowing users to return to previous sessions without losing context.

- Message
  Represents a single exchange within a chat. Each message is tied to a chat and stored chronologically. Messages may originate from either the user or the agent. For agent responses, messages may also contain streaming metadata or associated proposal identifiers.

- Proposal
  Represents a structured access request drafted by the agent. A proposal includes fields such as the target resource, requested role, justification, and expiration details. Proposals are treated as first-class objects within the system to allow versioning, editing, and eventual submission to MidPoint.

- Document
  Represents the source which the RAG module retrieved from. It consists of metadata such as the title, the broad category it belongs to and the document itself. These documents are stored in a simulated wiki Open Source Software (OSS), Bookstack which can be swapped out to our client's preferred wiki.

These entities are shared across the frontend, backend, and agent to maintain uniformity. Bastion manages this structured data through relational storage combined with an event-sourced audit log that records every state transition.

4. Sentinel Frontend

Sentinel Frontend is a single page web application, written with React and in TypeScript and built with Vite. It is the interface which users will interact with, allowing them to authenticate to the system and access the Sentinel system.

4.1. React, TypeScript and Vite

In discussions with the sponsor, we determined that the company already uses React extensively, thus adopting it ensures consistency and allows for easier maintainability of our application in the future by the sponsor. React itself is also a mature library for building interactive user interfaces, making it ideal for scalable application development[4].

TypeScript allows for static typing, enhancing code quality by catching errors early during development and improving maintainability through clearer contracts and autocompletion in editors[5].

Vite complements this stack by providing a fast and efficient build tool optimized for libraries like React[6]. It allows for faster development startup and optimized production builds such as automatic code splitting and tree shaking. Furthermore, native support for TypeScript also helps to simplify setup.

4.2. Clean architecture and organisation

Our project uses a feature-based modular architecture that groups related components, logic, data fetching, and state management within feature boundaries. This isolates shared utilities such as date-time, services such as authentication and navigation stores, and common UI elements such as a badge component to represent different environments. This design aligns with software engineering principles that improves maintainability for the client, allows for scalable features, and the ease of reading as the developer[7].

We are able to achieve this via with the following decisions made:

- Each feature directories contain components, hooks that are used for multiple API calls, feature-specific stores that manage client states, and types,
- Shared module directory for cross-feature utilities,
- Layout, page and routes are separated from the UI components and feature logic. The pages put features together based on layout, and individual components follow Don't Repeat Yourself (DRY) principle where they are focused and reusable[8].

---

[4] Angular Minds - React development Services, "Why React Will Stay Popular in 2025 and Beyond," Angular Minds, November 22, 2024, https://www.angularminds.com/blog/why-react-is-so-popular.
[5] Paul Bratslavsky, "Top 6 Benefits of Implementing Typescript," Strapi, February 26, 2025, https://strapi.io/blog/benefits-of-typescript.
[6] "Getting Started | Vite," Vite, accessed November 29, 2025, https://vite.dev/guide/.
[7] Mayank, "Design Principles in System Design," GeeksforGeeks, July 23, 2025, https://www.geeksforgeeks.org/system-design/design-principles-in-system-design/.
[8] baeldung, "Dry Software Design Principle | Baeldung on Computer Science," ed. Michal Aibin, Baeldung, June 17, 2023, https://www.baeldung.com/cs/dry-software-design-principle.

- Feature-level hooks handle data fetching using React Query to hold server state, isolating API concerns in one place. Feature-level stores on the other hand, handle client state but they do not hold server state again. Pages or parent components consume these hooks rather than mixing API and UI logic.

Sentinel Frontend is also designed following some software engineering best practices:

- DRY
  With the centralised common utilities in shared modules, the logic is not duplicated and there is no redundancy. This reduces maintenance overhead when it comes down to bug fix or changes. Axios authentication logic only needs to be defined once and it reduces inconsistencies and implementations across features.

- Separation of concerns
  Codes are organised by feature and reduce coupling across features. Such as stores and hooks having their own purpose. Visual components and logic are separated and this layering enforces structure.

- SOLID principles
  While SOLID principles are meant for Object-Oriented Programming, we applied some principles to our case for UI. Each module has its single responsibility for example an API is made to call the backend endpoint, the hook is used for data fetching and caching, and a component is meant to display the UI. With this separation, we are able to extend functionalities easily.

4.3. Authentication

Users are authenticated using OAuth 2.0 with Proof Key for Code Exchange (PKCE) to provide secure authentication, suited for single page applications that cannot securely store a client secret[9]. To simulate the sponsor's single sign on (SSO) system, we use Keycloak as our identity provider, managing user authentication via a centralized login system. Critical authentication related data, including user identity, active sessions and realm configurations are stored securely in a PostgreSQL database. Overall, it strengthens security by mitigating common OAuth vulnerabilities and provides a smooth user experience.

4.4. State Management

The state management in the frontend follows a clear separation of concerns between client side and server side state to ensure real time data hydration and rendering. Zustand is used for managing UI and local client state, while TanStack Query handles server state. The server state also acts as a single source of truth, with optimistic UI updates that are rolled back on error to maintain consistency.

A complex example of this is the chat streaming feature, where TanStack Query fetches messages hierarchically from the server. Messages sent by the user are shown optimistically in the UI for immediate responsiveness, while background revalidation syncs this with the actual server state. Streaming updates from the server are rendered token by token as well.

---

[9] Okta, "Single-Page Apps," OAuth 2.0 Simplified, December 16, 2021, https://www.oauth.com/oauth2-servers/single-page-apps/.

Our client states, managed with Zustand, stores information specialized by domain:

- Auth store: Manages session and user identity
- Chat store: Coordinates active chats and streaming state
- Proposal store: Handles requests that are drafted from the backend
- Document store: Manages documents, chat and message associations, and UI highlighting.

4.5. Data Validation

We use Zod schemas in order to enforce business rules at runtime, complementing TypeScript's compile time checks. Validation is applied consistently across system boundaries, such as at API responses, form inputs and request payloads. These schemas encode business logic directly, such as enforcing character limits and restricting environment values, preventing scattered rules throughout the codebase while providing users with immediate, clear feedback.

4.6. Integration Testing

Integration testing verifies that core user workflows such as login, chat viewing, label editing and chat streaming function correctly in a realistic environment. A mock backend was created from the API endpoint specifications and run as a lightweight Javascript server for simplicity.

Using Docker Compose, the testing environment spins up Keycloak, Postgres and the mock backend alongside the frontend container. In the CI pipeline, Docker Compose initializes the stack, installs Playwright with Chromium dependencies, executes the test suite, stores the results and then tears down the environment.

Tests were built using Playwright Test, which automates browser interactions, to validate end-to-end behaviour. The Playwright VSCode extension was used to record UI actions into code, speeding up test creation and ensuring scenarios closely reflect actual user interactions.

4.7. CI

Pre-commit automation through Husky helps to run quality checks on modified files, improving commit speed and catching errors early with ESLint auto fixes and Prettier formatting for consistency. Commitlint enforces conventional commits, enabling automated changelog generation, semantic versioning and clearer code reviews.

The CI pipeline, built on Github Actions, runs parallel tasks for linting, type checking and testing to minimize build times. It scans dependencies for vulnerabilities and verifies builds prior to publishing, ensuring secure delivery. On merge to main, continuous delivery automatically builds and uploads the container image to Sentinel's GHCR.

4.8. Pages

4.8.1. Login

This is the page that unauthenticated users would be greeted with. Our frontend is integrated with Keycloak which bases its authorisation rules on Midpoint, our IGA as part of the simulated environment. The Keycloak integration can be swapped out with our client's choice of OIDC.

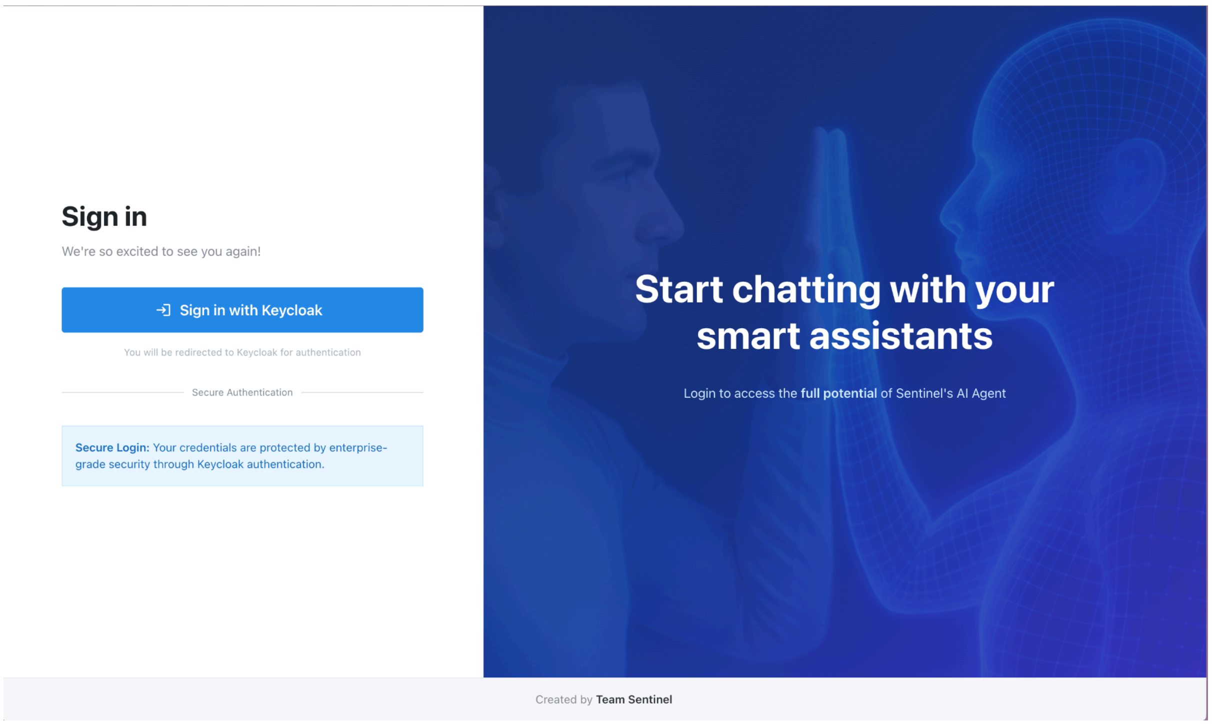
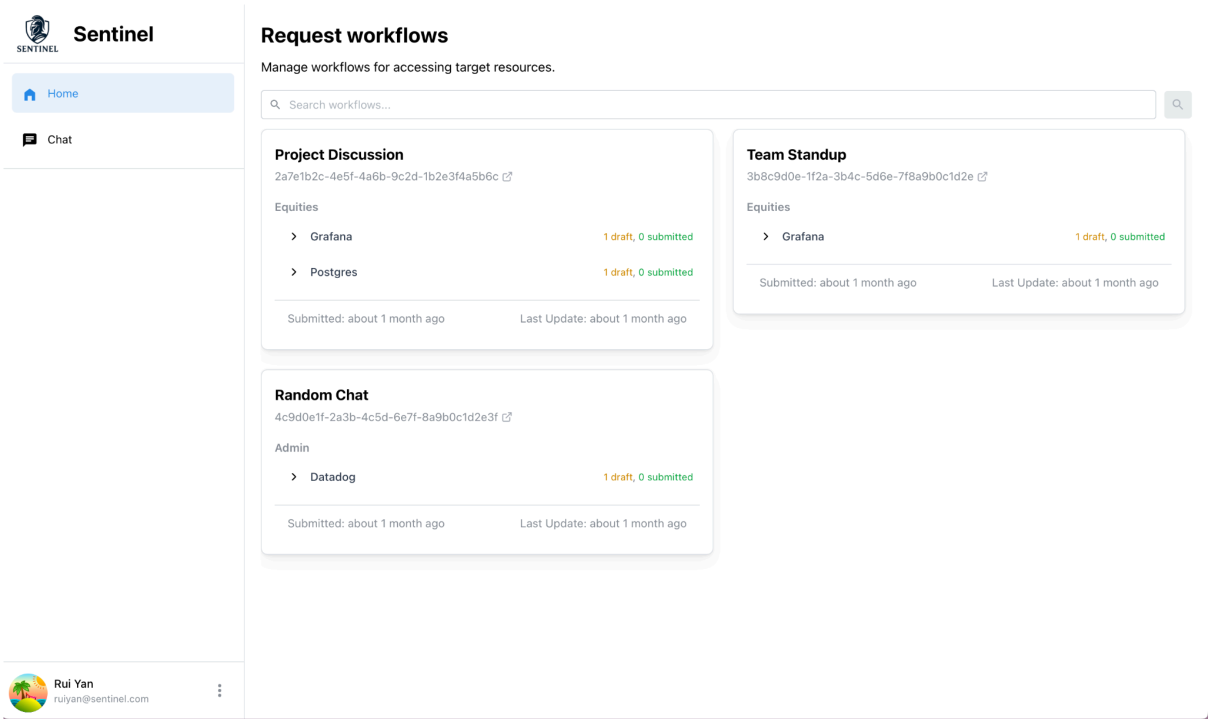

Sign in
We're so excited to see you again!
Sign in with Keycloak
You will be redirected to Keycloak for authentication
Secure Authentication
Secure Login: Your credentials are protected by enterprise-grade security through Keycloak authentication.
Start chatting with your smart assistants
Login to access the full potential of Sentinel's AI Agent
Created by Team Sentinel

Fig 4.8.1: Frontend login page

4.8.2. Dashboard

Dashboard page presents a clean interface to display the various chats the user has, along with the systems, resources, environment and roles that have been applied for. They are grouped logically as dropdowns. The status allows the requestors to understand which proposals have been submitted to the IGA.


Sentinel

Home

Chat

Rui Yan
ruiyan@sentinel.com

Request workflows

Manage workflows for accessing target resources.

Search workflows...

Project Discussion
2a7e1b2c-4e5f-4a6b-9c2d-1b2e3f4a5b6c

Equities

Grafana	1 draft, 0 submitted
Postgres	1 draft, 0 submitted

Submitted: about 1 month ago   Last Update: about 1 month ago

Team Standup
3b8c9d0e-1f2a-3b4c-5d6e-7f8a9b0c1d2e

Equities

Grafana	1 draft, 0 submitted

Submitted: about 1 month ago   Last Update: about 1 month ago

Random Chat
4c9d0e1f-2a3b-4c5d-6e7f-8a9b0c1d2e3f

Admin

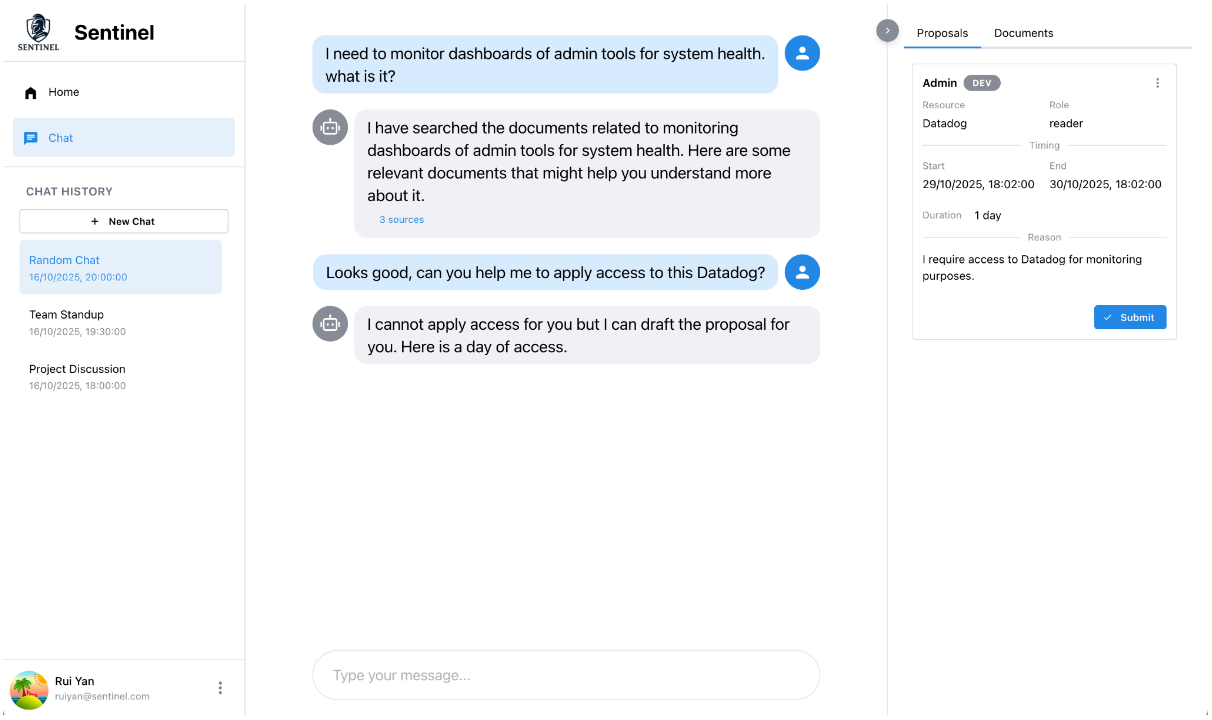
Datadog	1 draft, 0 submitted

Submitted: about 1 month ago   Last Update: about 1 month ago

Fig 4.8.2: Dashboard page that shows the status of the proposals made in each chat sort by systems

4.8.3. Chat

A neat interface that presents the chat as the highlight and where the requestors would interact with. Proposals are drafted access requests that exist on Sentinel services but not in the IGA. Only when the users click "Submit" after reviewing it, Sentinel Backend service would send a POST request to the IGA's endpoint to create it. This gives the requestors the full control to submit the requests.


Sentinel

Home

Chat

CHAT HISTORY

New Chat

Random Chat
16/10/2025, 20:00:00

Team Standup
16/10/2025, 19:30:00

Project Discussion
16/10/2025, 18:00:00

Rui Yan
ruiyan@sentinel.com

I need to monitor dashboards of admin tools for system health. what is it?

I have searched the documents related to monitoring dashboards of admin tools for system health. Here are some relevant documents that might help you understand more about it.
3 sources

Looks good, can you help me to apply access to this Datadog?

I cannot apply access for you but I can draft the proposal for you. Here is a day of access.

Type your message...

Proposals   Documents

Admin DEV

Resource	Role
Datadog	reader

Timing

Start	End
29/10/2025, 18:02:00	30/10/2025, 18:02:00

Duration 1 day

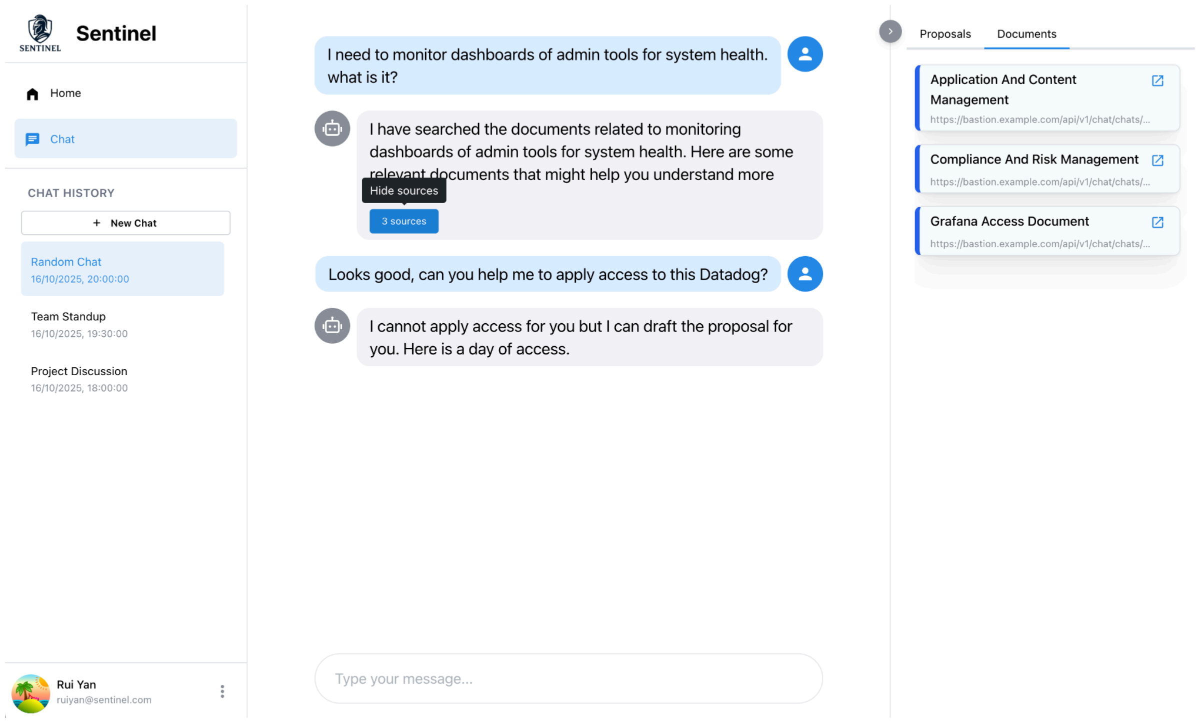
Reason

I require access to Datadog for monitoring purposes.

Submit

Fig 4.8.3.1: Proposal panel uncollapsed

As requestors tend to question the source and would like to see concrete evidence from the agent, requestors can click on “Sources” which will switch the tab to “Documents” with highlights of the specific documents that they referred to.


I need to monitor dashboards of admin tools for system health. what is it?

I have searched the documents related to monitoring dashboards of admin tools for system health. Here are some relevant documents that might help you understand more

Looks good, can you help me to apply access to this Datadog?

I cannot apply access for you but I can draft the proposal for you. Here is a day of access.

Application And Content Management
https://bastion.example.com/api/v1/chat/chats/...

Compliance And Risk Management
https://bastion.example.com/api/v1/chat/chats/...

Grafana Access Document
https://bastion.example.com/api/v1/chat/chats/...

Fig 4.8.3.2: Documents panel with highlights as the source is clicked from the message

4.9. Internationalisation

Internationalisation (i18n) support was incorporated to ensure that our product is able to scale to a multinational environment, since our client is a multinational company. While English is the primary language used, the system is designed so that additional languages can be easily extended without restructuring existing components. The text copywriting is separated to different files that allows our client to add or update languages with low effort. This provides long-term scalability and accessibility to diverse users.

5. Sentinel Backend (Bastion)

Bastion is the core backend server written in Go. It has two main purposes: to abstract away calls to internal services for calling by the frontend, and to handle all chat-related services. This includes handling the orchestration of service calls, and persistence of the chat resources with audit trails.

5.1. Language and dependencies

Go was chosen for its strong performance and built-in concurrency model, making scaling straightforward and easy[10]. The robust and production-ready standard library reduces the need to add external dependencies, lowering maintenance and security overhead.

[10] Olszak, Jacek, and Karolina Rusinowicz. “Why Golang May Be a Good Choice for Your Project.” CodiLime, January 21, 2022. https://codilime.com/blog/why-golang/.

For persistence, an SQL database was deemed suitable due to the domain objects having clear relationships with each other that could be modelled. PostgreSQL was chosen due to its good performance and reliability, as well as its strong support for JSON data via JSONB[11].
A cache was also required for storing short-lived data, such as session data. Redis was chosen for its simple key-value interface and fast speed in storing and retrieving data[12].
5.2 Code architecture
The code base utilises a hexagonal (ports and adapters) architecture[13], aligned with Domain-Driven Design (DDD) principles. By separating *domains*, *ports*, *infrastructure* and *services* into distinct packages, there is a clear boundary between the core business logic and the external implementations.

The *ports* package also defines clear and stable interfaces, allowing infrastructure components such as the database, cache and IAM systems, to be swapped out without affecting the domain logic. This gives the sponsor flexibility to integrate it with their preferred infrastructure.

The architecture also improves testability, by enabling the domain logic and individual adapters to be tested independently. Mock implementations are also used for testing things such as *services*, simplifying tests and improving development speed.

5.3 Orchestration of internal services

Bastion communicates between several internal services:

- Sentinel Agent
- IGM system, Midpoint
- Authentication provider, Keycloak
- Documentation source, Bookstack

5.3.1 Integration with Sentinel Agent

Communication with Agent is done via gRPC, using protobuf files to define the contracts. Bastion calls the Agent service as part of the chat flow, after a user sends a message. Upon receiving the message, Bastion will query and construct the context of the chat accordingly, which includes the message history and draft proposals.

5.3.2 Integration with IGM system (Midpoint)

Certain operations, such as modifying proposal fields, or submitting the access requests require knowledge of the IGM constraints and domain rules. To perform these tasks, it is essential that Bastion is able to access the IGM system itself, as it will serve as the first point of validation for user inputs, before sending any data to the IGM system itself. Beyond that, Bastion will also use it for the querying of user identity & account metadata, retrieval of available resources, as well as to perform actions on behalf of users.

[11] Dudycz, Oskar. “PostgreSQL JSONB - Powerful Storage for Semi-Structured Data.” PostgreSQL JSONB - Powerful Storage for Semi-Structured Data, April 21, 2025. https://www.architecture-weekly.com/p/postgresql-jsonb-powerful-storage.
[12] Fanchi, Christopher. “Redis: What It Is, What It Does, and Why You Should Care.” Backendless, May 22, 2023. https://backendless.com/redis-what-it-is-what-it-does-and-why-you-should-care/.
[13] Dhruv, Stuti. “Hexagonal Architecture: Principles and Benefits - 2025.” Aalpha, June 14, 2025. https://www.aalpha.net/blog/hexagonal-architecture/.

5.3.3 Integration with Authentication Provider (Keycloak)

Authentication for requests from the frontend is handled using JWTs. The JWTs are issued via the chosen auth provider, which is then attached to all requests in the "Authorization" header. Upon receiving requests to protected routes, Bastion will then verify the JWTs with the keys provided. We have also provided JWK workers which will query for the latest JWKs from the auth provider, on a schedule, ensuring that the latest keys are available, in line with best security practices.[14]

5.3.4 Integration with Documentation Source (Bookstack)

Due to the structure of the Agent's response, it may include the document sources in which the data is relevant. If provided with the relevant adapters, Bastion will query metadata from the sources before sending it as part of the response to the frontend, enabling a richer UI/UX which displays more information about the sources besides just the URL itself.

5.4 Chat services

5.4.1 HTTP Server

Meant solely for the frontend, Bastion provides REST endpoints for all its services. This includes simple interactions such as CRUD actions, to complex tasks, such as sending a new message, triggering multiple steps or calling external services.

All endpoints use a consistent format for ease of identification: */api/{version}/{domain}/{resource}*. Note that the swagger JSON file, as well as a swagger UI endpoint is provided. The swagger JSON file can be found at *internal/infra/http/server/static/swagger.json*.

Bastion also provides a streaming endpoint for the chat response, utilising Server-Sent Events (SSE) as the method for streaming. The agent's text response is sent back to the frontend in small chunks as events, which provides a smooth chat-like experience rather than waiting for the entire payload. SSE was chosen as it provides a simple way to stream from the server to client, as streaming in the other direction is not required[15]. Additionally, less resources are required on the server side to maintain the connection as compared to web sockets, which is more resource heavy.

5.4.2 Persistence (audit trail)

As stated above, the persistence layer uses SQL for storing of the resource data. Each resource has its own table and relationships are mapped via foreign keys. However, having the latest state is not enough as we need to track the actions taken for audit purposes. Hence, we decided to go with a hybrid approach, using both regular tables as well as event tables to track events in an event-sourced manner.

Every resource has its own dedicated event table which is populated on every data modification. Deletion across all tables is disabled, and only soft-deleting is allowed by setting the *deleted_at* column. This ensures that any and all modifications are persisted, providing a clear audit trail if needed.

[14] De Ryck, Philippe. "The Hard Parts of JWT Security Nobody Talks About." Pragmatic Web Security, March 13, 2020. https://pragmaticwebsecurity.com/articles/apisecurity/hard-parts-of-jwt.html.
[15] Martin, Eve. "WebSockets vs Server-Sent Events: Key Differences and Which to Use in 2024." Ably Realtime, September 26, 2024. https://ably.com/blog/websockets-vs-sse.

5.5 Continuous Integration

Bastion uses a multi-stage pipeline hosted on GitHub Actions for continuous integration.

Creation and commits to pull requests trigger the CI, which includes both a format check enforced via *gofmt* as well as testing. The testing suite is built on Go's built-in testing framework and includes both unit and integration tests, using mock infrastructure where required, or *Testcontainers* to spin up ephemeral 3rd party dependencies.

Merging of pull requests into main then triggers the building of the container image which then gets uploaded to the GitHub Container Registry (GHCR). The Software Bill Of Materials (SBOM) is then generated, which allows for auditing of softwares and dependencies used, should there be vulnerabilities or legal issues. The images are also then scanned for vulnerabilities using Trivy, before finally being signed using Cosign to make it tamper-proof.

6. Sentinel Agent

Sentinel Agent is the core AI component of Project Sentinel built on LangGraph, processing user queries received from Bastion and responding accordingly.

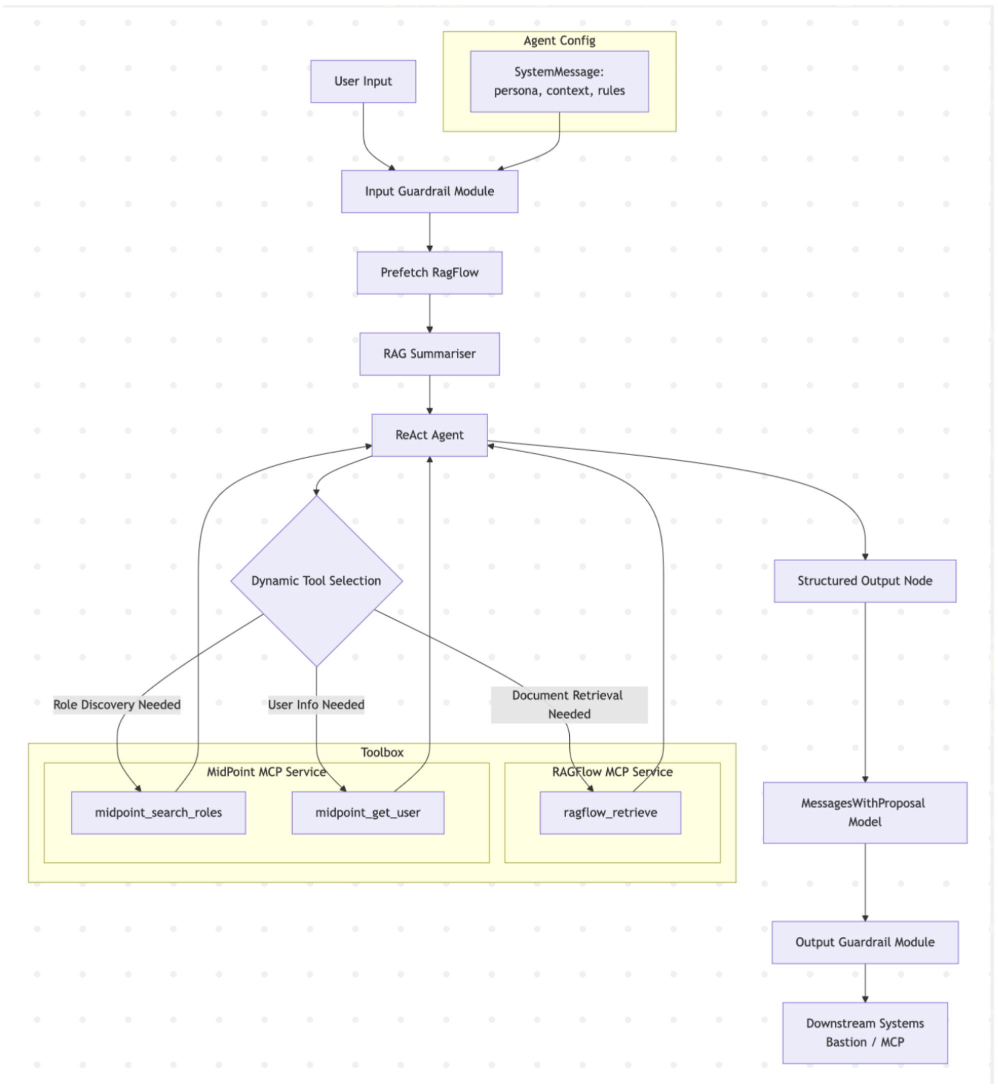


Fig 6.1: Agent Graph

For audit purposes, and given the regulatory requirements of the sponsor, every single action is logged centrally, such as tool calls, requests and responses.

6.1. LangGraph, Python and OpenAI

LangGraph was chosen due to its active maintenance, strong community support, and extensive documentation, which made integration smooth. The team also had prior experience using LangGraph for chatbot development and LangChain tool integration, providing a solid foundation for adoption. Another key reason was scalability. Since the project is expected to handle a high number of agent requests due to the company's push for temporary permissions, the agent pipeline needed to run efficiently under heavy loads. LangGraph, being code-based, can be deployed on servers or cloud instances, allowing horizontal and cloud-native scaling through autoscaling or container orchestration.

We designed the solution to be fully model-agnostic, allowing users to swap in any LLM they prefer. LangGraph's broad support for modern models makes this seamless, and if a team brings their own privately trained model, LangGraph's built-in wrappers make integration straightforward.

Apart from LangGraph, our team also considered several alternatives, which we rejected due to various reasons:

- N8n: Scaling relies on queue mode or multiple instance setups and is limited by the platform's architecture. Its browser-based workflow editor can become sluggish or unstable under heavy workloads, as observed by one team member with experience working with this tool. It is less suited for large-scale, dynamic, AI-driven pipelines.
- CrewAI: Offers limited customisation for structured outputs and lacks modular integration for tools such as MCP and RAGFlow.

Here, Python was chosen as our preferred language due to its extensive support for machine learning, being the de facto standard language for many AI and machine learning libraries[16], which allows for rapid prototyping. Its simplicity enhances collaboration among team members with varying expertise, making development smoother as well.

We selected OpenAI as our primary model provider because of their industry-leading natural language capabilities, strong contextual reasoning, and reliable generation quality.[17] Their ongoing research, mature tooling, and focus on safety make integration smooth and predictable. Furthermore, our team is already familiar with their APIs and ecosystem, allowing us to build, iterate, and troubleshoot quickly without additional overhead.

6.2. System Architecture

6.2.1. Inter Process Communication

Similar to Bastion, Sentinel Agent employs protocol buffers and gRPC for inter process communication, using the same defined core entities "chat", "message", and "proposal". On top of

[16] Elliott, Thomas. "The State of the Octoverse: Machine Learning." The GitHub Blog, February 17, 2022. https://github.blog/news-insights/octoverse/the-state-of-the-octoverse-machine-learning/.
[17] OpenAI. "Measuring the Performance of Our Models on Real-World Tasks." OpenAI. Accessed November 2025. https://openai.com/index/gdpval/.

being more performant than standard communication protocols such as HTTP[18], it also allows us to work on both Bastion and Agent concurrently without issues, as both sides simply have to implement the contract.

6.2.2. ReAct Agent

Our main reasoning module adopts the ReAct (Reasoning + Acting) architecture[19] with internal prompt chaining utilising multiple LLM Nodes, enabling the agent to reason over user inputs and invoke tools in sequence to produce structured outputs such as access proposals. When the agent is invoked, it is able to dynamically select between multiple tools available to it based on the user query, calling them to request for additional information, supplementing its response:

- RAGFlow MCP: Exposes the “ragflow_retrieve” tool, which returns documents relating to the request
- Midpoint MCP: Exposes the “midpoint_search_roles” tool, which returns the requested role from the Midpoint IGA system, and the “midpoint_get_user” tool, which returns the requested user from the Midpoint IGA system, along with the permissions they currently have

6.2.3. System Messages

For this project, the system prompt is kept as a separate, stable configuration that defines the agent’s role, high-level objectives, safety rules, and response style. It is written using an XML style structure so that each part of the behavior specification is clearly tagged, for example separating sections for role, constraints, tools, and output format. Structured prompting makes the system message easier to maintain over time and reduces ambiguity for the model, since it can clearly see which parts are instructions rather than conversational content or retrieved knowledge[20]. This approach also aligns with OpenAI’s own guidance on using XML like tags to clearly delineate instructions and reference content within prompts[21], helping the model understand where each component begins and ends and producing more reliable, structured outputs for downstream processing.

When sending a query to the model, the user message, recent chat history, and supplementary data are combined into a single XML structured payload that sits alongside the system prompt. The user’s current question is wrapped in its own tag, while previous turns are serialized as a sequence of tagged messages, and any retrieved or tool generated context is placed in clearly labeled sections. This structured packaging helps the model distinguish instructions from dialogue and background context, improving grounding while keeping the core system prompt clean and reusable across different conversations.

[18] Berisha, Blend. “Impact of Protocol Selection on Performance and Scalability in Microservices: A Comparison of GRPC, Rest, and Graphql.” University of Prishtina, June 2025. https://www.researchgate.net/publication/392507557_Impact_of_Protocol_Selection_on_Performance_and_Scalability_in_Microservices_A_Comparison_of_gRPC_REST_and_GraphQL.

[19] Bergmann, Dave. “What Is a React Agent?” IBM, November 20, 2025. https://www.ibm.com/think/topics/react-agent#:~:text=IBM%20Think-,ReAct%20agent%20overview,and%20toward%20complex%20problem%2Dsolving.

[20] He, Jia, Mukund Rungta, David Koleczek, Arshdeep Sekhon, Franklin X Wang, and Sadid Hasan. “Does Prompt Formatting Have Any Impact on LLM Performance?” arXiv.org, November 15, 2024. https://arxiv.org/abs/2411.10541.

[21] OpenAI. “Prompt Engineering - Openai API.” OpenAI. Accessed November 2025. https://platform.openai.com/docs/guides/prompt-engineering.

6.2.3. RAGFlow pre fetch

In practice however, we noticed that the model showed a preference for our Midpoint MCP tools, and often did not retrieve any documentation from the RAGFlow MCP. We suspect this was because Midpoint MCP provides much more concise and directly relevant data, making it easier for the LLM to reason with, while RAGFlow MCP returns textual documentation without inherent structure or purpose labels, making the purpose of such documentation not as easily understood. This behavior seems to align with ReAct's natural bias, as the agent optimises for the most deterministic and information-dense path when multiple tools can satisfy a query.

Thus, we implemented a prefetch RAGFlow module, which uses the user query to perform an initial retrieval of relevant documents before the main reasoning step, retrieving a broad set of potentially useful documents from our knowledge base, which is then passed together with the user query to our main reasoning module.

However, one thing we noticed was that at times, this raw context could be extremely large, to the point that it was overshadowing the system prompt and agent instructions. Thus, we added a RAG summariser that condenses the retrieved documents into a focused summary capturing only the key facts and references needed for the task. This concise summary is then passed into the ReAct agent alongside the system prompt, preserving the agent's behavior while still grounding it in up to date, relevant knowledge from the underlying documents. If the agent decides that further information is still needed while processing the query, it still has access to the RAGFlow MCP to retrieve any additional information.

6.2.4. Structured Outputs

We used Pydantic schemas to define the exact JSON structures that our agent must return, such as fields for Proposals and Messages. By generating strict schemas from these models and validating every model response against them, we converted the free form natural language outputs of our agent into predictable, typed data that can be consumed safely by downstream services and UI components. This approach also aligns well with modern structured output features in LLM APIs, which are designed to respect JSON schemas and significantly reduce parsing errors.[22]

However, maintaining these complex schemas while also tracking long running conversational state inside the same service quickly became difficult. Every change to the structured output format required careful coordination with how state was stored and mutated over time, which introduced tight coupling and made debugging harder. To address this, we decided to keep the agent service stateless, with state being managed externally by Bastion, and necessary context is included in the requests instead. This separation simplifies schema evolution, makes the agent service easier to test and scale horizontally, and avoids subtle bugs that can occur when mixing schema heavy logic with in memory conversation state.

6.2.5. Guardrails

In line with the security requirements from the sponsor, guardrails in the system were implemented using the LLM Guard framework, with a clear separation between input and output filtering.

[22] OpenAI. "Structured Model Outputs - Openai API." OpenAI. Accessed November 2025. https://platform.openai.com/docs/guides/structured-outputs.

Before even reaching the main reasoning model or the RAGFlow pre fetch module, all user inputs go through a dedicated input guardrail layer, so that any potentially harmful or non-compliant prompts are stopped as early as possible, reducing the risk of leaking internal context or misusing the retrieval and reasoning components. This layer uses a set of scanners focused on detecting unsafe or adversarial prompts before they reach any downstream components. In particular, it applies an invisible text detector to catch attempts to hide instructions using zero width characters or other obfuscation tricks, and a prompt injection detection model, deberta-v3-base-prompt-injection-v2. This model is tuned to recognize patterns such as attempts to override the system prompt, exfiltrate internal configuration, or redirect the agent away from its intended task[23], which makes it well suited to protecting our agent from being "hijacked".

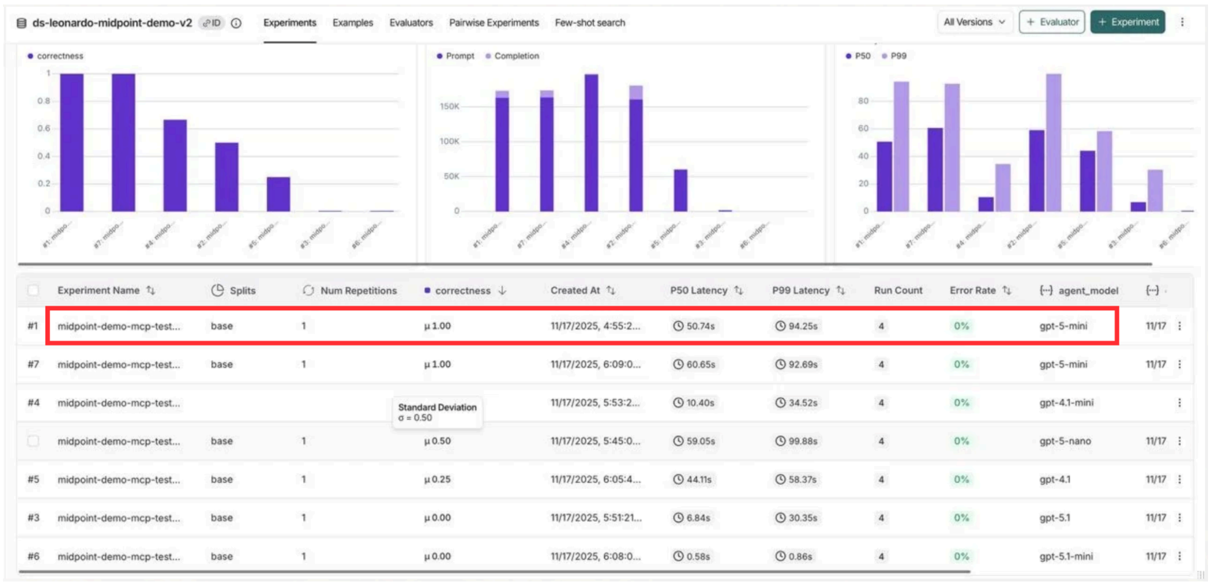
Just before the response is returned to the user, the output is scanned by the output guardrail layer. This two-layer design follows a "defense in depth" approach recommended in industry security guidance from major vendors[24], where both prompts and responses are actively monitored and sanitized, rather than relying only on model alignment. Here, every model response is checked for two main issues, model refusal and harmful language. The distilroberta-base-rejection-v1 model is used to detect when the LLM has refused a request[25], which is indicative of a prompt injection attack which made it past our initial screening, but was blocked by the model's own internal training. In parallel, toxicity is checked using the unbiased-toxic-roberta model, which ensures that model responses do not contain harmful or inappropriate content such as racism, hate speech, or sexual content.[26] This protection is especially important in an enterprise context, because a single toxic answer could damage the sponsor's reputation, breach internal policies, or violate regulatory and ethical requirements.[27]

All of these scanners are wired into the guardrail framework as modular components, and each can be enabled or disabled as needed, and new scanners can be added without code changes to the core agent. This modularity lets the team evolve safety policies over time while keeping the overall architecture stable and maintainable.

During design, other guardrail options were evaluated but ultimately not adopted. NVIDIA NeMo's safety tooling was considered, but it is tightly coupled to the NVIDIA ecosystem, which conflicted with the team's goal of keeping the overall system as model and vendor agnostic as possible. We also explored GuardrailsAI, however, its setup required installing and managing additional components through a separate package manager and wiring them into the stack, which introduced operational complexity compared to the relatively straightforward integration of the LLM Guard framework.

[23] HuggingFace. "Protectai/Deberta-V3-Base-Prompt-Injection-V2." Hugging Face. Accessed November 2025. https://huggingface.co/protectai/deberta-v3-base-prompt-injection-v2.
[24] Sethi, Hitendar, and Kay Clark. "Building Secure AI by Design: A Defense-in-Depth Approach." Palo Alto Networks Blog, November 25, 2025. https://www.paloaltonetworks.com/blog/network-security/building-secure-ai-by-design-a-defense-in-depth-approach/.
[25] HuggingFace. "Protectai/Distilroberta-Base-Rejection-V1." Hugging Face. Accessed November 2025. https://huggingface.co/protectai/distilroberta-base-rejection-v1.
[26] HuggingFace. "Unitary/Unbiased-Toxic-Roberta." Hugging Face. Accessed November 2025. https://huggingface.co/unitary/unbiased-toxic-roberta.
[27] Holweg, Matthias, Rupert Younger, and Yuni Wen. "The Reputational Risks of AI." California Management Review, January 24, 2022. https://cmr.berkeley.edu/2022/01/the-reputational-risks-of-ai/.

6.3. Model Testing



	Experiment Name	Splits	Num Repetitions	correctness	Created At	P50 Latency	P99 Latency	Run Count	Error Rate	agent_model	{--}
#1	midpoint-demo-mcp-test...	base	1	μ 1.00	11/17/2025, 4:55:2...	50.74s	94.25s	4	0%	gpt-5-mini	11/17
#7	midpoint-demo-mcp-test...	base	1	μ 1.00	11/17/2025, 6:09:0...	60.65s	92.69s	4	0%	gpt-5-mini	11/17
#4	midpoint-demo-mcp-test...			Standard Deviation σ = 0.50	11/17/2025, 5:53:2...	10.40s	34.52s	4	0%	gpt-4.1-mini	
	midpoint-demo-mcp-test...	base	1	μ 0.50	11/17/2025, 5:45:0...	59.05s	99.88s	4	0%	gpt-5-nano	11/17
#5	midpoint-demo-mcp-test...	base	1	μ 0.25	11/17/2025, 6:05:4...	44.11s	58.37s	4	0%	gpt-4.1	11/17
#3	midpoint-demo-mcp-test...	base	1	μ 0.00	11/17/2025, 5:51:21...	6.84s	30.35s	4	0%	gpt-5.1	11/17
#6	midpoint-demo-mcp-test...	base	1	μ 0.00	11/17/2025, 6:08:0...	0.58s	0.86s	4	0%	gpt-5.1-mini	11/17

Fig 6.3.1: Model Testing Results via Langsmith

While the system is built to be model-agnostic, we still ran a focused evaluation to identify the most suitable model for our POC. Using LangSmith, a platform for structured LLM experimentation, evaluation, and tracing. We benchmarked several models by running the same set of queries and comparing the outputs to “golden answers”. With this, we used an LLM-as-judge to score correctness, measured latency, and inspected tool call behaviour through LangSmith traces.

From our analysis, GPT-5-mini delivered the strongest overall performance: the highest correctness scores with consistently low latency. Some newer models produced higher-quality responses but were constrained by significantly tighter token limits (up to 10× smaller), making them unsuitable for our use case especially with tool calls like RAG that required high token counts. Older models in the GPT-4 and GPT-3 series had more generous token limits but consistently underperformed in accuracy and reasoning.

6.4. System Message Testing

When tuning the system message, we treated it like an engineering component rather than a static prompt. We experimented with different levels of detail, structure, and ordering to understand how each element influenced behaviour. This included defining the agent’s role, outlining its objectives, specifying how it should communicate, and setting explicit boundaries for what it must avoid doing. We also incorporated OpenAI’s prompting guidelines, which emphasise clarity, deterministic structure, and showing and not just telling the model what “good” looks like. Across iterations, we learned that overly long instructions led to drift, while overly short ones caused ambiguity. The most stable and accurate behaviour came from a concise, modular system message that included a clear role framing, a list of rules/restrictions, a step-by-step workflow for how the agent should reason, and a short example response. This structure gave the model consistent guidance without overwhelming it, leading to more predictable reasoning and better task adherence.

6.5. Guardrails Testing

Although the creators of the individual guardrail models have already conducted their own testing and validation, we wanted to evaluate the guardrails specifically in the context of our agent architecture and use case. The behavior of these scanners can vary depending on the type of queries users ask, the domain specific language used, and how the guardrails interact with the rest of our system, so context specific testing is essential to ensure they perform well in production.

To evaluate the implemented guardrails in our agent, we simulated a combination of benign and malicious prompts to test the overall effectiveness of our configuration. We created 100 benign prompts based on our model's expected use case. For malicious inputs, we used a dataset from a study on jailbreaking LLMs, which collected jailbreak prompts from real environments in the wild, simulating real potential attacks.[28] From this malicious dataset, we randomly select 100 entries. These 200 sample entries, 100 benign and 100 malicious, were run against our entire agent pipeline with guardrails enabled to test the overall effectiveness of our guardrail combination.

Because the guardrail models allow us to tune the acceptance threshold, we ran our tests at varying threshold levels to determine the optimal balance between security and utility. Lower thresholds reduce the rate of legitimate queries being blocked but allow more malicious prompts through, while higher thresholds block more malicious prompts but also increase the rate of legitimate queries being incorrectly flagged. We evaluated thresholds between 0.5 and 0.9 at intervals of 0.1.

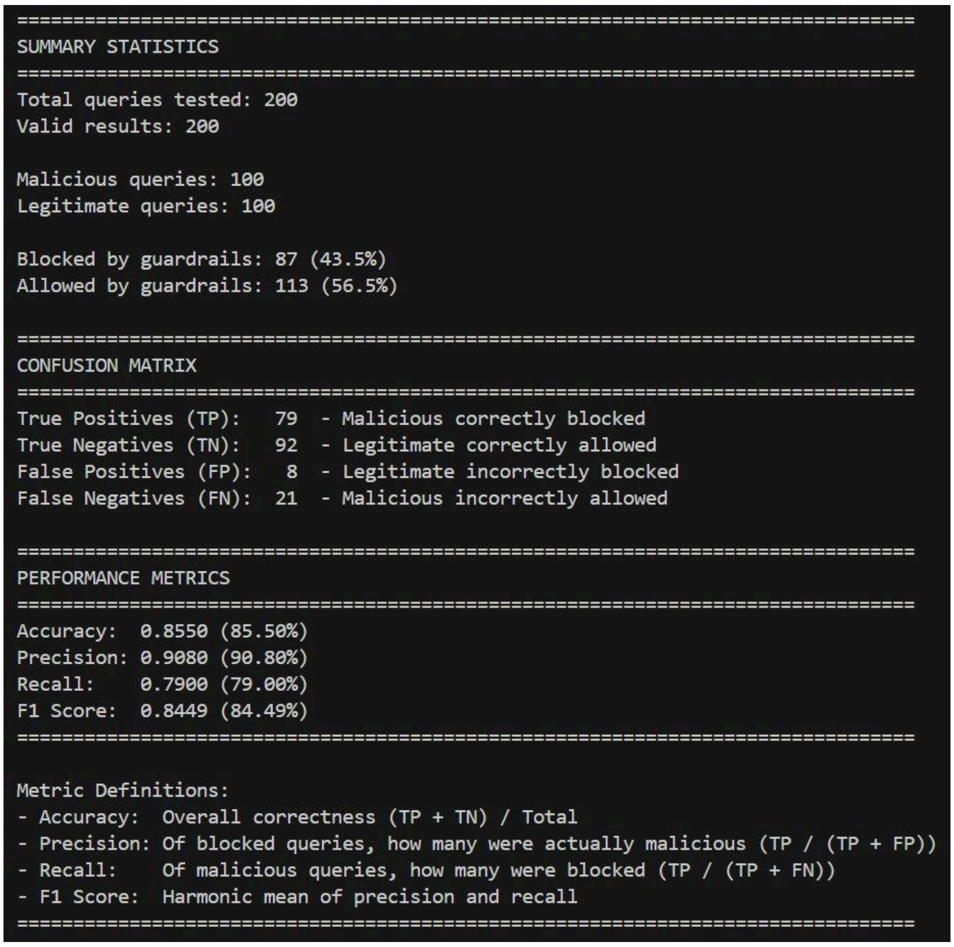
For interpreting results, we treated a blocked outcome as the positive class and an allowed outcome as the negative class. This means a True Positive is a malicious query correctly blocked, a True Negative is a legitimate query correctly allowed, a False Positive is a legitimate query incorrectly blocked, and a False Negative is a malicious query incorrectly allowed.

We evaluated four main metrics to assess performance across different thresholds.

- Accuracy measures overall correctness as the proportion of correct decisions, calculated as (TP + TN) divided by Total.
- Precision measures the proportion of blocked queries that were actually malicious, calculated as TP divided by (TP + FP).
- Recall measures the proportion of malicious queries that were successfully blocked, calculated as TP divided by (TP + FN).
- F1 Score provides a harmonic mean of precision and recall, giving a single balanced metric that accounts for both false positives and false negatives.

We recorded the detailed results for each threshold, and found that a threshold of 0.7 was best, as it provided the best balance between security and utility.

[28] Shen, Xinyue, Zeyuan Chen, Michael Backes, Yun Shen, and Yang Zhang. "'Do Anything Now': Characterizing and Evaluating in-the-Wild Jailbreak Prompts on Large Language Models." arXiv.org, May 15, 2024. https://arxiv.org/abs/2308.03825.



```
================================================================================
SUMMARY STATISTICS
================================================================================
Total queries tested: 200
Valid results: 200

Malicious queries: 100
Legitimate queries: 100

Blocked by guardrails: 87 (43.5%)
Allowed by guardrails: 113 (56.5%)

================================================================================
CONFUSION MATRIX
================================================================================
True Positives (TP):   79  - Malicious correctly blocked
True Negatives (TN):   92  - Legitimate correctly allowed
False Positives (FP):   8  - Legitimate incorrectly blocked
False Negatives (FN):  21  - Malicious incorrectly allowed

================================================================================
PERFORMANCE METRICS
================================================================================
Accuracy:  0.8550 (85.50%)
Precision: 0.9080 (90.80%)
Recall:    0.7900 (79.00%)
F1 Score:  0.8449 (84.49%)
================================================================================

Metric Definitions:
- Accuracy:  Overall correctness (TP + TN) / Total
- Precision: Of blocked queries, how many were actually malicious (TP / (TP + FP))
- Recall:    Of malicious queries, how many were blocked (TP / (TP + FN))
- F1 Score:  Harmonic mean of precision and recall
================================================================================
```

Fig 6.5.1: Guardrails test results

7. MidPoint MCP

7.1. Overview

MidPoint MCP functions as the integration layer that bridges Sentinel with the simulated identity governance system, MidPoint. Its primary role is to expose a standardized set of tools that the Sentinel Agent can invoke to retrieve identity, role, and entitlement information in a controlled and structured manner. This abstraction layer ensures that the agent does not interact with MidPoint directly. Instead, all interactions flow through MCP tool calls that enforce schema rules, security constraints, and compatibility with the agent's reasoning framework. By introducing this separation, the system remains modular and adaptable, allowing the underlying IGA implementation to be replaced or extended in the future without disrupting the agent's workflow.

MCP ensures that all data returned to the agent is controlled and schema-driven. Because the MCP server sits between the agent and the IGA APIs, it dictates exactly what information can be exposed, returning only the fields defined in the tool schema while filtering out sensitive or unnecessary data. This allows the system to maintain data hygiene and delivers only the specific identity, role, or entitlement information required for safe and effective reasoning.

7.2. HTTP Server and Tool Contract Design

The MidPoint MCP service is implemented as an HTTP server that exposes various tools used by the agent. The most important tool is a role search function that retrieves roles based on a provided prefix. The decision to use HTTP rather than alternative transports such as stdin or Server-Sent Events was influenced by several practical considerations. HTTP provides native support for bearer token authentication, is easy to integrate within microservice ecosystems, and aligns with the network patterns used by both Bastion and the broader Sentinel architecture. It also supports clear request response semantics, which simplifies debugging, monitoring, and testing.

Each MCP tool is constructed around a typed schema that defines the structure of both the input and output. This design ensures that all data returned to the agent adheres to strict validation rules. If a request fails schema validation or contains untrusted or unsafe content, the MCP service rejects it before it ever reaches MidPoint. By keeping the tool contract explicit and schema driven, the system prevents malformed input from influencing IGA behavior and ensures that the agent always receives well-formatted and predictable responses.

7.3. Role Search and MidPoint-Specific Implementation Details

The MCP server primarily exposes two tools: Role Search and User Lookup, each designed to provide the Sentinel Agent with the precise information required for identity-governance reasoning while presenting over-exposure to the underlying IGA dataset.

Role Search is intentionally constrained to only return requestable roles. The MCP service binds the query to a predefined subset of MidPoint's role model. This guarantees that the agent can retrieve only roles that are eligible for end-user access requests, preventing accidental requests to administrative, deprecated, or non-public roles. By enforcing this restriction at the MCP layer, the system maintains strong control over what the agent can discover, preserving governance boundaries and ensuring consistent behavior across all agent interactions.

User Lookup tool provides contextual identity information by retrieving a user's existing entitlements, roles, and attributes. This enables the agent to reason about whether an access request is appropriate based on what the user already has without exposing sensitive or unnecessary identity attributes. As with Role Search, the MCP server filters and shapes the response according to the tool schema, ensuring that only approved fields such as current entitlements relevant to the workflow are returned.

Together, these tools provide the minimal but complete set of capabilities the agent needs to perform IGA-related reasoning, while tightly controlling the scope of data it can access. This design reinforces the principle of least privilege and ensures that MidPoint remains protected behind the MCP abstraction layer.

7.4. Integration with LangGraph

During development, the team encountered several challenges related to integrating MCP with LangGraph. LangGraph predates the most recent standardization of the Model Context Protocol and implements its own simplified tooling structure. As a result, certain MCP features, such as fully typed structured outputs, were not directly supported. To address these constraints, the team carefully analyzed the behavior of LangChain MCP libraries, which share similar goals and are actively evolving toward MCP compliance.

By learning from these libraries, the team aligned MCP with LangGraph's expectations by mapping structured output fields into the unstructured output fields that LangGraph currently supports. Throughout this process, the team relied heavily on the MCP Inspector, a tool used for visually testing the tools without the need for an agent or a built UI. The resulting integration not only solved the immediate compatibility issues but also positioned the system for easy adaptation once LangGraph supports the full MCP standard.

7.5. Testing and Validation

The MidPoint MCP service underwent extensive testing to ensure reliability, security, and correctness. Unit tests validated the behavior of individual components, including authentication middleware, schema validation, response formatting, and sanitization logic. Integration tests verified the complete operational flow, from receiving an MCP request to fetching real data from MidPoint and returning structured responses. Special emphasis was placed on ensuring that the tool search function remained resistant to malformed input and did not expose internal details beyond what was explicitly permitted.

These tests also verified streaming behavior where applicable, confirmed consistent status codes, and ensured stable operation across different deployment configurations. By validating each component independently and in combination, the team ensured that MCP remained a dependable pillar of the Sentinel ecosystem.

7.6. Continuous Integration and Supply Chain Security

The MCP service employs a dedicated CI pipeline that automatically builds container images on each merge. As part of this process, the pipeline generates a Software Bill of Materials using Syft[29], which documents all dependencies included in the image. The container image and its attestation are then signed using Cosign[30], ensuring traceability and supply chain integrity. These practices mirror enterprise DevSecOps standards and ensure that each container image is traceable, integrity-checked, and backed by a verified record of its dependencies.

8. Sentinel Simulated Environment

8.1. Overview

The Sentinel Simulated Environment was created to reproduce the operational realities of an enterprise-scale identity governance ecosystem without requiring access to Morgan Stanley's confidential infrastructure. Since the core objective of Project Sentinel is to evaluate an AI agent's ability to navigate complex access workflows, the environment had to provide authentic identity data, realistic provisioning flows, consistent authentication processes, and meaningful documentation sources. The simulated ecosystem therefore integrates MidPoint for identity governance, Keycloak for authentication, Grafana as a representative target application, and Bookstack with RAGFlow for documentation and retrieval operations. Together, these systems establish a controlled but highly realistic platform in which the agent's reasoning can be tested under conditions that mirror a production environment.

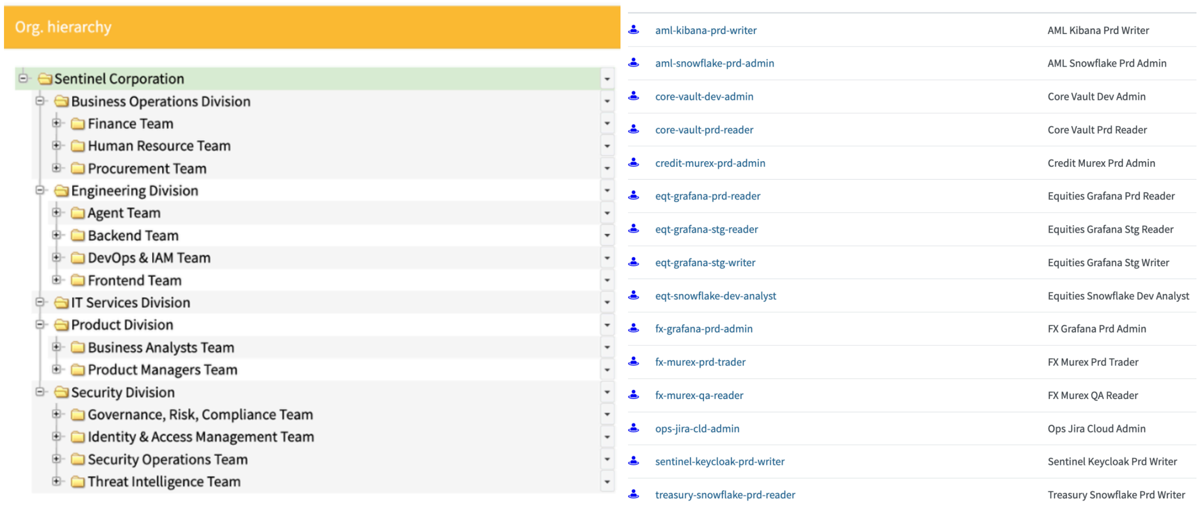
The use of a simulated environment also enabled rapid prototyping and testing. Each system could be configured, extended, or reset without risking real user accounts or organizational data. By carefully

[29] https://github.com/anchore/syft
[30] https://github.com/sigstore/cosign

tuning role hierarchies, connector configurations, and documentation structures, the team ensured that the simulated environment reflected the complexity and diversity typical of large corporate IGA landscapes.

8.2. MidPoint Identity Governance and Administration System

MidPoint serves as the backbone of the simulated IGA environment. It manages user accounts, groups, entitlements, role hierarchies, provisioning logic, and policy enforcement. The team configured MidPoint with a comprehensive hierarchy of business roles and application roles, including nested relationships and realistic entitlement structures. These roles were also annotated with purpose descriptions, requestability indicators, and mappings to application-level permissions.



Org. hierarchy

- Sentinel Corporation
  - Business Operations Division
    - Finance Team
    - Human Resource Team
    - Procurement Team
  - Engineering Division
    - Agent Team
    - Backend Team
    - DevOps & IAM Team
    - Frontend Team
  - IT Services Division
  - Product Division
    - Business Analysts Team
    - Product Managers Team
  - Security Division
    - Governance, Risk, Compliance Team
    - Identity & Access Management Team
    - Security Operations Team
    - Threat Intelligence Team

aml-kibana-prd-writer	AML Kibana Prd Writer
aml-snowflake-prd-admin	AML Snowflake Prd Admin
core-vault-dev-admin	Core Vault Dev Admin
core-vault-prd-reader	Core Vault Prd Reader
credit-murex-prd-admin	Credit Murex Prd Admin
eqt-grafana-prd-reader	Equities Grafana Prd Reader
eqt-grafana-stg-reader	Equities Grafana Stg Reader
eqt-grafana-stg-writer	Equities Grafana Stg Writer
eqt-snowflake-dev-analyst	Equities Snowflake Dev Analyst
fx-grafana-prd-admin	FX Grafana Prd Admin
fx-murex-prd-trader	FX Murex Prd Trader
fx-murex-qa-reader	FX Murex QA Reader
ops-jira-cld-admin	Ops Jira Cloud Admin
sentinel-keycloak-prd-writer	Sentinel Keycloak Prd Writer
treasury-snowflake-prd-reader	Treasury Snowflake Prd Writer

Fig 8.2.1: Organisation structure and application roles

MidPoint's internal connectors allowed the system to synchronize user identities and role assignments with both Keycloak and Grafana. For example, when a user was assigned an application role in MidPoint, provisioning policies triggered updates that created or modified corresponding accounts in Keycloak and Grafana. This behavior mirrored the real workflows of corporate identity governance, where access changes propagate across tightly integrated systems.[31] The richness of MidPoint's model enabled the agent to query meaningful role data through MCP, improving the reliability of the tool-based reasoning flow.

8.3. Keycloak Authentication and Identity Bridging

Keycloak functions as the identity provider for the Sentinel Frontend as well as for the simulated target applications. All user authentication flows are based on OAuth 2.0 with PKCE, which provides a secure mechanism for session management within browser-based applications.[32] By serving as the centralized identity provider, Keycloak ensures that identity attributes remain consistent across Sentinel, Grafana, and any other integrated systems.

[31] Evolveum. "Practical Identity Management with midPoint." docs.evolveum.com, November 19, 2024. https://docs.evolveum.com/book/practical-identity-management-with-midpoint.html
[32] Curity. "Proof Key for Code Exchange Overview." Curity Resources, January 23, 2025. https://curity.io/resources/learn/oauth-pkce/

The interplay between MidPoint and Keycloak is particularly important. MidPoint manages user identities and assigns roles, but Keycloak handles actual authentication and token issuance. The MidPoint Keycloak connector ensures that any new user or entitlement change in MidPoint is reflected in Keycloak's internal representation of users and their roles. This automatic propagation creates a seamless bridge between identity governance and authentication. As a result, when the agent advises a user to request a specific role, that role's assignment in MidPoint translates into real authentication and access privileges within the simulated environment.

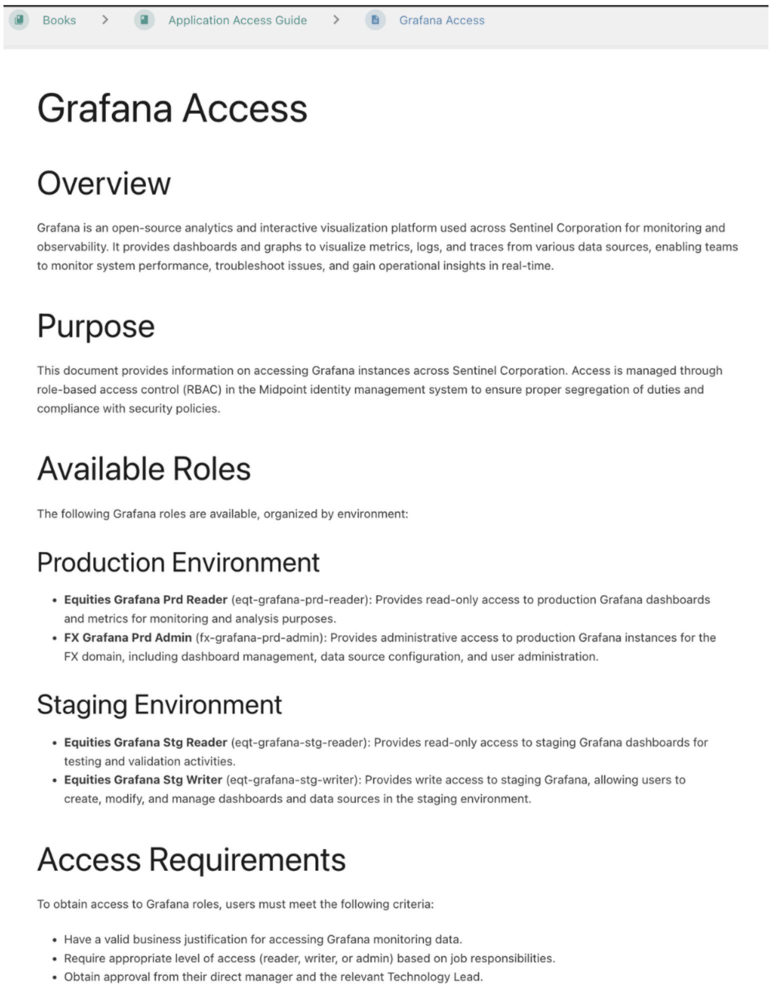
8.4. Grafana Application Integration

Grafana was selected as a representative target application to evaluate the agent's ability to reason about application-level access. It offers clear role definitions, intuitive access boundaries, and meaningful dashboard privileges that map neatly to least-privilege principles. Grafana was configured to accept Keycloak tokens for authentication through OpenID Connect, meaning users authenticate through Keycloak rather than natively in Grafana. This mirrors common enterprise SSO patterns.

MidPoint's provisioning logic was extended to automatically create or update Grafana accounts based on role assignments. For example, assigning the "eqt-grafana-prd-reader" role in MidPoint would result in provisioning the corresponding account and permissions within Grafana. This interaction allowed the team to demonstrate end-to-end flows, where a user asks the agent for access, receives a least-privilege proposal, and ultimately gains corresponding permissions in the target system after the proposal is submitted.

8.5. Bookstack Knowledge Repository and RAGFlow Integration

Bookstack served as the documentation platform that stores policies, procedures, role descriptions, and system reference material. These documents formed the primary knowledge sources required for retrieval-augmented generation. To enable retrieval operations, the team manually loaded the relevant Bookstack content into RAGFlow, ensuring it was available as a structured knowledge source.


Books > Application Access Guide > Grafana Access

Grafana Access

Overview

Grafana is an open-source analytics and interactive visualization platform used across Sentinel Corporation for monitoring and observability. It provides dashboards and graphs to visualize metrics, logs, and traces from various data sources, enabling teams to monitor system performance, troubleshoot issues, and gain operational insights in real-time.

Purpose

This document provides information on accessing Grafana instances across Sentinel Corporation. Access is managed through role-based access control (RBAC) in the Midpoint identity management system to ensure proper segregation of duties and compliance with security policies.

Available Roles

The following Grafana roles are available, organized by environment:

Production Environment

- **Equities Grafana Prd Reader** (eqt-grafana-prd-reader): Provides read-only access to production Grafana dashboards and metrics for monitoring and analysis purposes.
- **FX Grafana Prd Admin** (fx-grafana-prd-admin): Provides administrative access to production Grafana instances for the FX domain, including dashboard management, data source configuration, and user administration.

Staging Environment

- **Equities Grafana Stg Reader** (eqt-grafana-stg-reader): Provides read-only access to staging Grafana dashboards for testing and validation activities.
- **Equities Grafana Stg Writer** (eqt-grafana-stg-writer): Provides write access to staging Grafana, allowing users to create, modify, and manage dashboards and data sources in the staging environment.

Access Requirements

To obtain access to Grafana roles, users must meet the following criteria:

- Have a valid business justification for accessing Grafana monitoring data.
- Require appropriate level of access (reader, writer, or admin) based on job responsibilities.
- Obtain approval from their direct manager and the relevant Technology Lead.

Fig 8.5.1: Bookstack "Page" representing an internal policy document

To ensure that the documentation made available in Bookstack remained consistent and reproducible across different deployments of the simulated environment, an automated setup script was implemented to handle the initialization and configuration of the Bookstack service. Bookstack's bulk import functionality requires the system to operate temporarily under its standard internal authentication mode rather than under Keycloak OIDC. The script therefore orchestrates a controlled bootstrap sequence in which Bookstack is first started in standard mode, the required documentation is imported, and the application logs are monitored to verify that the import has completed successfully. After the import, the script assigns appropriate default roles to the imported users and then restores Bookstack's OIDC configuration before restarting the service. This automated workflow ensures that the Bookstack instance available to the frontend always contains a clean and complete copy of the documentation without requiring manual intervention, supporting a stable and repeatable simulated environment for user-facing interactions.

RAGFlow provides document chunking, embedding generation, vector storage, metadata indexing, and retrieval APIs. These features made it particularly suitable for the Sentinel project, where the ability to trace retrieved documents, evaluate search accuracy, and tune retrieval behavior is essential.

In the final configuration, the agent accesses RAGFlow whenever it needs contextual policy information. RAGFlow returns the most relevant documentation fragments along with structured metadata that allows the agent to cite or reference them. This capability enhances the agent's trustworthiness and ensures that responses remain grounded in authoritative sources.

8.6. Document Retrieval

8.6.1. RAGFlow

The Sentinel Simulated Environment required a robust retrieval augmented generation system because organizational policies are highly specific, frequently updated, and contain nuanced details that cannot be captured in a general-purpose language model's training data. Additionally, the agent must provide accurate, citable responses referencing authoritative sources, handle time sensitive information such as maintenance windows and access freezes that change dynamically, and navigate complex relationships between policies, roles, and entitlements.

After evaluating several RAG solutions, the team selected RAGFlow as the foundation for Sentinel's knowledge retrieval system.

- RagPi was considered but lacked comprehensive features such as hybrid search capabilities and reranker integration.
- Custom RAG implementations using LangChain were explored but deemed too resource-intensive given the project's time constraints, as building a production-ready system would have required significant development effort for document chunking, embedding pipelines, vector database integration, and evaluation frameworks.
- LlamaIndex would have required substantial configuration work to achieve the same functionality that RAGFlow provides out of the box.

Thus, RAGFlow emerged as the optimal choice because it provides a complete, integrated platform that includes document parsing and chunking, embedding generation, vector storage, hybrid retrieval capabilities, reranker integration, and built-in evaluation tools, significantly reducing development time while providing production-ready features.

The system operates through a workflow where documents are parsed, chunked into semantically meaningful segments, embedded using selected models, and stored in Elasticsearch. During retrieval, when the agent submits a query through the Sentinel frontend via Bastion, RAGFlow performs hybrid search combining weighted keyword similarity (BM25) with weighted vector similarity, merges and deduplicates candidates, and optionally forwards them to a reranker service for refined ranking. The reranked results with source metadata are returned to the agent, which incorporates them into its context to generate accurate, grounded responses with inline citations and source references.

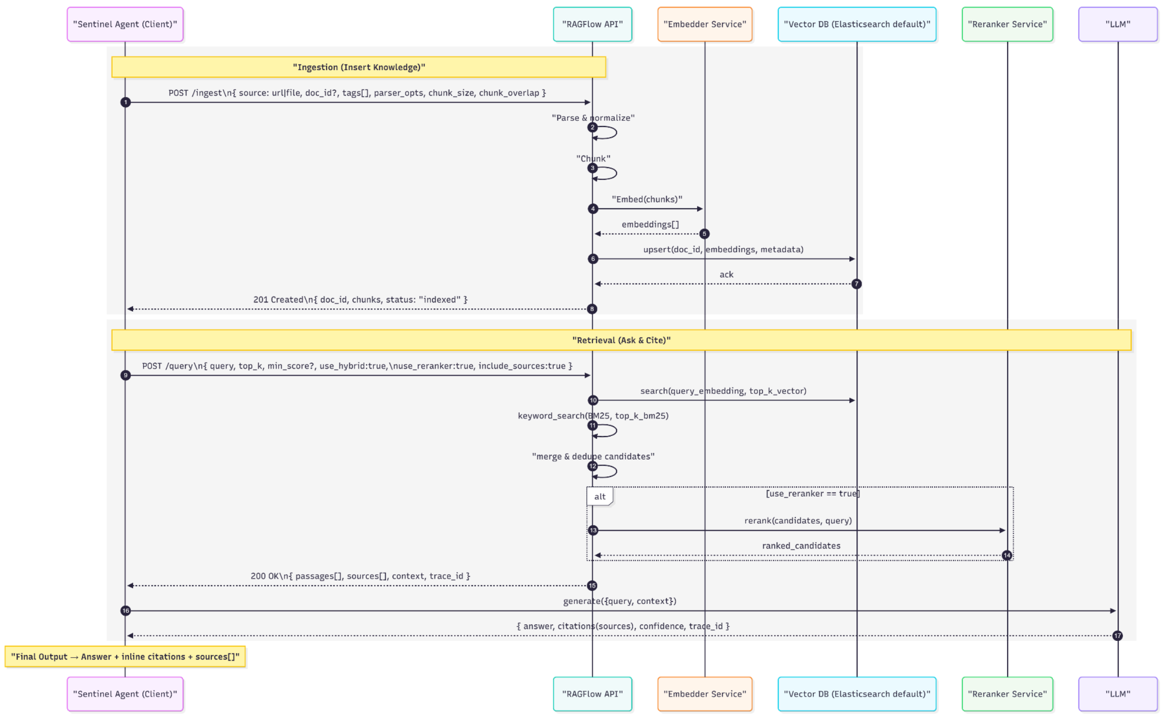
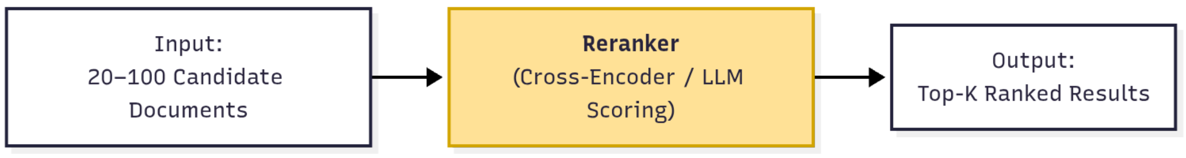

"Sentinel Agent (Client)"
"RAGFlow API"
"Embedder Service"
"Vector DB (Elasticsearch default)"
"Reranker Service"
"LLM"
"Ingestion (Insert Knowledge)"
POST /ingest\n{ source: url|file, doc_id?, tags[], parser_opts, chunk_size, chunk_overlap }
"Parse & normalize"
"Chunk"
"Embed(chunks)"
embeddings[]
upsert(doc_id, embeddings, metadata)
ack
201 Created\n{ doc_id, chunks, status: "indexed" }
"Retrieval (Ask & Cite)"
POST /query\n{ query, top_k, min_score?, use_hybrid:true,\nuse_reranker:true, include_sources:true }
search(query_embedding, top_k_vector)
keyword_search(BM25, top_k_bm25)
"merge & dedupe candidates"
alt
[use_reranker == true]
rerank(candidates, query)
ranked_candidates
200 OK\n{ passages[], sources[], context, trace_id }
generate({query, context})
{ answer, citations(sources), confidence, trace_id }
"Final Output → Answer + inline citations + sources[]"

Fig 8.6.1.1: RAGFlow user flow diagram

8.6.2. Reranker

Rerankers represent a critical component in modern information retrieval systems, operating as the second stage in a two-stage retrieval pipeline that balances efficiency with accuracy. While first-stage retrieval methods such as keyword search and vector similarity are fast and scalable, they have inherent limitations in understanding complex semantic relationships, contextual nuances, and fine-grained relevance distinctions.



Rerankers address these limitations by applying more sophisticated models to a smaller, manageable set of candidates (typically 20-100 documents) retrieved by the first stage. This two-stage approach is essential because running expensive, high-accuracy models on an entire corpus would be prohibitively costly and slow, while using cheap models across the entire corpus would sacrifice accuracy.

Rerankers can be categorized into several types: feature-based rerankers using handcrafted features and traditional ML models, bi-encoder rerankers that encode queries and documents separately, cross-encoder rerankers that process query-document pairs together enabling rich interaction modeling, and LLM-powered rerankers that leverage large language models for advanced understanding. Each type offers different tradeoffs in accuracy, latency, cost, and deployment complexity, making the choice dependent on specific application requirements, budget constraints, and performance targets.

To evaluate reranker performance and determine the optimal choice for Sentinel's production system, the team conducted two comprehensive experiments. Experiment 1 established baseline performance characteristics using a controlled benchmark with 100 curated sentences and 10 carefully designed queries with ground truth relevance labels, testing four reranker providers: Cohere's rerank-english-v3.0, Voyage AI's rerank-2, Contextual AI's ctxl-rerank-v2-instruct-multilingual, and Jina AI's jina-reranker-v2-base-multilingual. Experiment 2 evaluated reranker performance in a production-like RAG system using real-world documentation chunks from Sentinel's RAGFlow instance and 19 challenging edge case queries designed to test time-sensitivity, context-dependency, negative queries, distinction queries, exception queries, role-based queries, and multi-hop reasoning scenarios. The experimental setup involved ingesting Sentinel and edge case datasets into RAGFlow, which chunked documents into 512-token segments, embedded them using text-embedding-3-small, and stored them in the knowledge base. During retrieval, RAGFlow performed first-stage hybrid retrieval combining weighted keyword search and weighted vector search, merging results to produce top-k=10 candidate chunks. These candidates were then reranked locally using each of the four reranker services, producing final ranked results for evaluation.

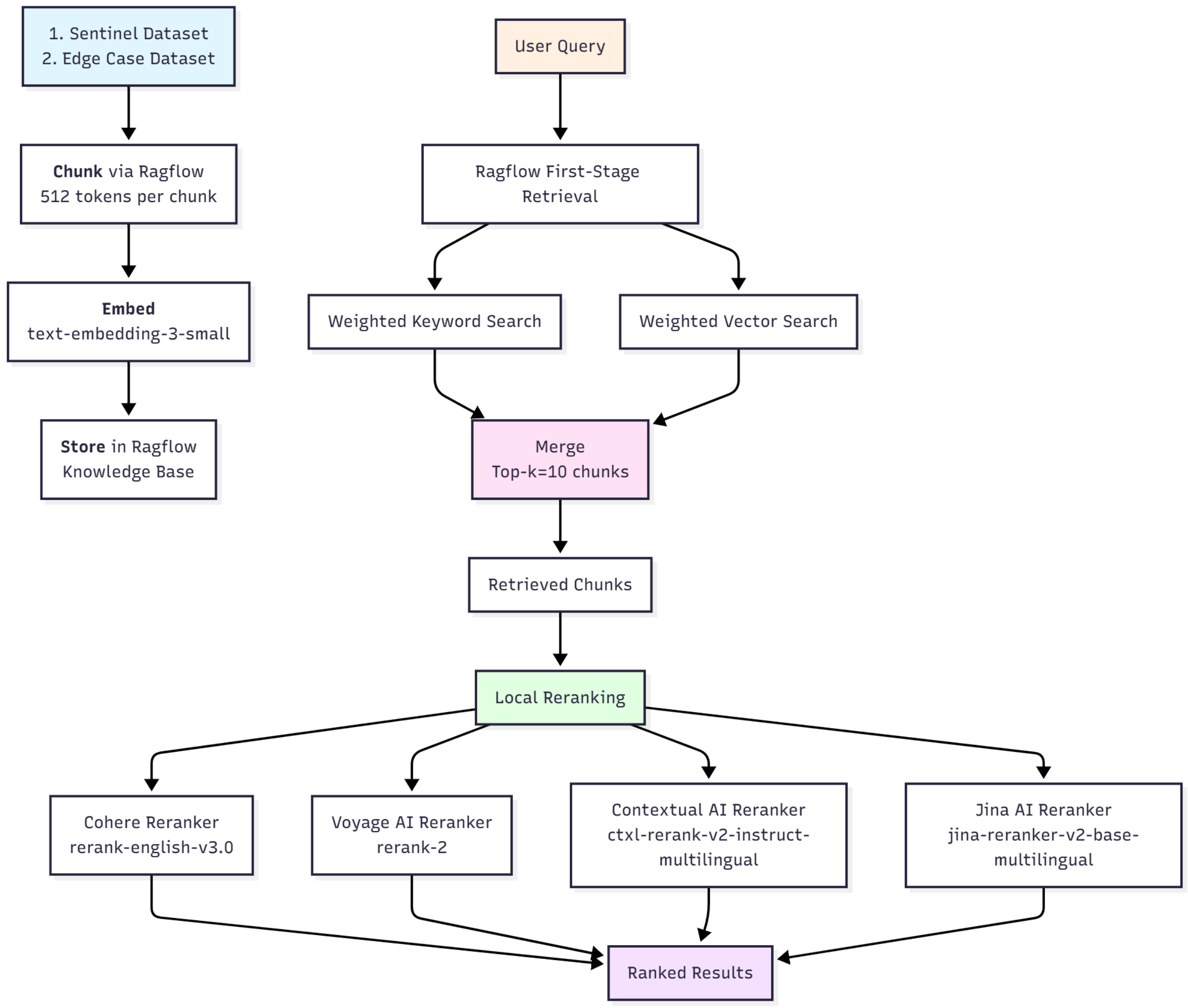


Fig 8.6.2.1: Experiment Flow Diagram

The experiments revealed significant insights about reranker performance. Experiment 1 results showed Voyage AI achieving perfect MRR (1.000) and tying with Contextual AI for highest NDCG@10 (0.729), while Cohere demonstrated best latency (411ms mean) with strong quality.

Experiment 2, which tested on real-world edge case queries, demonstrated that reranking adds substantial value, with Cohere and Voyage AI delivering 12%+ improvements in NDCG@10 over the baseline RAGFlow first-stage retrieval (0.911-0.914 vs 0.811 baseline). Voyage AI emerged as the overall winner in Experiment 2, achieving excellent ranking quality (MRR: 0.944, NDCG@10: 0.914), fastest latency (673ms mean), lowest cost ($0.50/1K docs), and 94.74% success rate. Cohere demonstrated the highest MRR (0.972) and excellent NDCG@10 (0.911) with good latency (877ms), making it an excellent alternative for applications prioritizing absolute best ranking quality. Contextual AI showed strong ranking quality (MRR: 0.872, NDCG@10: 0.847) but with highest latency (3,190ms), suitable for batch processing. Jina AI offered competitive performance (MRR: 0.794, NDCG@10: 0.783) with moderate latency (2,024ms) and good cost efficiency ($0.60/1K docs). The experiments also revealed that different rerankers capture different aspects of relevance, as evidenced by low to moderate inter-reranker agreement scores (Kendall's tau ranging from 0.173 to 0.480 in Experiment 2), suggesting that rerankers use different ranking strategies and that ensemble approaches could potentially outperform individual rerankers.

Metric	**Cohere**	**Voyage AI**	**Contextual AI**	**Jina AI**	**Baseline (RAGFlow)**
MRR	**0.972**	0.944	0.872	0.794	0.880
NDCG@5	0.875	**0.895**	0.801	0.746	0.747
NDCG@10	0.911	**0.914**	0.847	0.783	0.811
Precision@5	0.367	**0.389**	0.356	0.344	0.322
Recall@10	**0.968**	0.963	0.963	0.921	0.940
Mean Latency (ms)	877	**673**	3,190	2,024	-
Cost/1K Docs ($)	1.00	**0.50**	0.80	0.60	-
Success Rate	94.74%	94.74%	94.74%	94.74%	-

Fig 8.6.2.2: Reranker Performance Comparison Table (Experiment 2)

Deciding which reranker to use requires careful consideration of multiple factors. Performance metrics such as MRR, NDCG@10, precision, and recall directly impact retrieval quality, with higher scores indicating better ability to surface relevant documents at top positions.

User experience is closely tied to latency, as slower response times degrade the interactive experience, making Voyage AI's 673ms mean latency and Cohere's 877ms particularly attractive for real-time applications, while Contextual AI's 3,190ms latency may be acceptable only for batch processing scenarios.

Cost considerations become critical at scale, with Voyage AI offering the lowest cost at $0.50 per 1000 documents reranked, followed by Jina AI at $0.60, Contextual AI at $0.80, and Cohere at $1.00, representing a 2x cost difference between the cheapest and most expensive options.

When performance metrics are similar, inter-reranker agreement measured through Kendall's tau becomes particularly important, as low agreement scores (ranging from 0.173 to 0.480 in our experiments) indicate that different rerankers capture different aspects of relevance and use distinct ranking strategies. This low agreement means it is extra important to select the right reranker that fits your specific dataset and query patterns, as the choice can significantly impact results even when aggregate metrics appear similar.

9. Future Work

9.1. Integration with Production IGA and Enterprise Systems

The current prototype operates against a simulated environment built with MidPoint, Keycloak, Grafana and Bookstack. A natural next step is to integrate Sentinel with Morgan Stanley's IGA platform, documentation systems, and ticketing tools. This would involve replacing or augmenting MidPoint with the internal IGA system, connecting to the organization's real role catalog and wiring Sentinel into existing access request workflows. Such integration would validate Sentinel under real organisational constraints, expose the agent to a wider variety of edge cases and provide more accurate feedback on how the tool behaves when interacting with live users and production data.

9.2. Expansion of Knowledge Sources and RAG capabilities

At present, the RAGFlow subsystem operates on a curated set of documentation sourced from Bookstack. In a production environment, the documentation corpus would be significantly larger and more heterogeneous, spanning multiple wikis, internal portals, and policy repositories. Future work could focus on building a more sophisticated ingestion pipeline that supports incremental updates, document versioning, and automatic reindexing when policies change. Additional effort could be invested in fine tuning retrieval parameters and integrating the best performing reranker models identified during the evaluation phase, with particular attention to cost, latency, and domain specific accuracy. This would improve the agent's ability to provide precise, citation backed answers in complex policy scenarios.

9.3. Advanced Retrieval Strategies Beyond Reranking

Beyond rerankers, future work could explore additional retrieval strategies to further strengthen the RAG pipeline. Examples include enriching document chunks with tags and automatically generated keywords to support more precise metadata based filtering, as well as using question generation to improve how the system handles varied user phrasing. Structured approaches such as knowledge graphs and entity relationship extraction could help capture cross document links and support multi hop reasoning. Techniques like RAPTOR (Recursive Abstractive Processing for Tree Organised Retrieval) may also be investigated to build hierarchical representations over longer documents. These methods would complement rerankers and together could lead to a more accurate and resilient retrieval layer for Sentinel.

9.4. Advanced Guardrails and Continuous Risk Monitoring

The current guardrail system concentrates on prompt injection detection, invisible character filtering, toxicity checking, and generic rejection classification. An important area for future enhancement is the development of domain specific guardrails that are aware of the organization's risk profile, regulatory constraints, and internal policies. For example, Sentinel could incorporate rules to detect when a user is requesting roles that conflict with segregation of duties requirements, or when a proposed access path involves high risk entitlements that require additional approvals. These rules could be combined with continuous monitoring and alerting, so that unusual usage patterns or repeated attempts to bypass controls can be flagged to security teams in real time.

9.5. Evaluating with Real Users and Human in the Loop Workflows

While the current evaluation focuses on technical metrics such as retrieval performance, guardrail accuracy, and agent correctness on curated scenarios, future work should include user studies involving non technical staff, access approvers, and IAM engineers. Such studies would assess how effectively Sentinel reduces ticket resolution time, how understandable users find the generated proposals, and how confident approvers feel when relying on agent recommendations. Insights from these studies could inform improvements to the conversational UX, the presentation of proposals, and the level of explanation provided for each recommendation. In parallel, human in the loop workflows could be deepened so that approvers can provide structured feedback that the system can log and later use to refine its behavior.

9.6. Policy Validator Agent

Finally, the original project proposal included a second proof of concept focused on policy validation. With the access remediation agent now in place, future work could revisit this second POC and explore a multi agent architecture in which one agent drafts remediation proposals while another checks them against formal policy constraints. This policy validator agent could, for example, verify that a proposed role does not violate segregation of duties rules, that the requested duration is appropriate, or that the justification aligns with documented use cases. Over time, the two agents could be extended to coordinate on more complex workflows such as access reviews, attestation campaigns, or automated revocation of unused entitlements, further increasing the value of the Sentinel platform within an enterprise IGA programme.