

Acknowledgements

We would like to express our sincere gratitude to all those who contributed to the successful completion of this project.

First and foremost, we extend our deepest appreciation to our academic supervisor, **Mr. Mounir Kthiri**, for his invaluable guidance, constructive criticism, and continuous encouragement throughout this project. His expertise in software architecture and security best practices has profoundly influenced our approach to building LogChat.

We are grateful to the **Higher Institute of Technological Studies of Sousse** for providing the academic framework, technical resources, and conducive learning environment that made this project possible.

Our sincere thanks go to the **members of the jury** for dedicating their valuable time to review and evaluate this work. Their feedback and insights are instrumental in our professional growth.

We acknowledge the contributions of our **classmates and peers** who participated in beta testing the Golang Agent across various Windows and Linux environments. Their feedback helped identify critical edge cases and improve system reliability.

Special recognition goes to the **open-source community**, particularly the teams behind:

- **Next.js** — for the powerful React framework
- **Prisma** — for the elegant ORM solution
- **Ollama** — for democratizing local LLM deployment
- **The Go Team** — for creating a language perfect for systems programming

Finally, we extend our heartfelt thanks to our **families and friends** for their

moral support, understanding, and encouragement during the challenging phases of this project.

Saleh Eddine Touil & Chames Edin Turki

Tunis, January 2025

Abstract

The exponential growth of distributed systems, cloud-native architectures, and microservices has resulted in unprecedented volumes of log data, rendering traditional manual security auditing obsolete. Contemporary Security Information and Event Management (SIEM) solutions, while powerful, present significant barriers to adoption: prohibitive licensing costs exceeding \$100,000 annually, steep learning curves requiring specialized query languages, and complex multi-node deployment requirements.

LogChat addresses these challenges by democratizing security analytics through the application of Generative Artificial Intelligence. This end-of-studies project presents the design, implementation, and evaluation of an open-source, privacy-first SIEM platform comprising three core components:

1. A **high-performance Golang Agent** (5MB binary) enabling cross-platform log collection from Windows Event Logs, Linux Syslog/Journald, and arbitrary log files — compiled with zero runtime dependencies for seamless deployment.
2. A **Node.js/Express backend** providing RESTful APIs for log ingestion at 1000+ requests/second, real-time threat detection using pattern-based analysis, and a sophisticated RAG (Retrieval Augmented Generation) engine for contextual AI responses.
3. A **Next.js 14 frontend** delivering an intuitive, real-time security dashboard with sub-second update latency via Server-Sent Events (SSE) and a natural language chat interface for log analysis.

The integration of **Ollama** for local LLM inference ensures complete data privacy — sensitive log information never leaves the organization’s infrastructure. Deployed via **Docker Compose**, LogChat becomes operational within five minutes, transforming complex security queries into actionable insights without requiring SPL, KQL, or other proprietary languages.

Keywords: SIEM, Log Management, Generative AI, RAG, Golang, Docker, Cybersecurity, NLP, Real-time Analytics, Threat Detection, Local LLM, Privacy-First Architecture.

Résumé

La croissance exponentielle des systèmes distribués et des architectures cloud-native a entraîné des volumes sans précédent de données de journalisation, rendant l'audit de sécurité manuel obsolète. Les solutions SIEM contemporaines présentent des obstacles significatifs: coûts de licence prohibitifs, courbes d'apprentissage abruptes et exigences de déploiement complexes.

LogChat répond à ces défis en démocratisant l'analyse de sécurité grâce à l'Intelligence Artificielle Générative. Ce projet de fin d'études présente une plateforme SIEM open-source axée sur la confidentialité, comprenant:

1. Un **Agent Golang** haute performance pour la collecte de logs multi-plateforme
2. Un **backend Node.js/Express** avec détection de menaces en temps réel et moteur RAG
3. Un **frontend Next.js 14** offrant un tableau de bord interactif et une interface de chat IA

L'intégration d'**Ollama** pour l'inférence LLM locale garantit une confidentialité totale des données. Déployé via **Docker Compose**, LogChat devient opérationnel en cinq minutes.

Mots-clés: SIEM, Gestion des Logs, IA Générative, RAG, Golang, Docker, Cyber-sécurité, TAL, Analyse en Temps Réel.

Contents

Acknowledgements	1
Abstract	3
Résumé	5
Contents	6
List of Figures	12
List of Tables	14
List of Abbreviations	16
General Introduction	1
1 State of the Art	5
1.1 Introduction	5
1.2 Historical Evolution of Log Management	5
1.2.1 Early Approaches (1990s-2000s)	5
1.2.2 Centralized Logging (2005-2015)	6
1.2.3 Modern AI-Enhanced SIEM (2020-Present)	6
1.3 Market Analysis of Existing Solutions	7
1.3.1 Elastic Stack (ELK)	7
1.3.2 Splunk Enterprise	7

1.3.3	Wazuh	8
1.3.4	Microsoft Sentinel	9
1.3.5	Comparative Analysis	9
1.4	Technology Stack Selection	11
1.4.1	Backend: Node.js with TypeScript	11
1.4.2	Frontend: Next.js 14 with React 18	11
1.4.3	Database: PostgreSQL 16 with Prisma ORM	12
1.4.4	Agent: Golang	13
1.4.5	AI Engine: Ollama with Local LLMs	14
1.5	Development Methodology	15
1.6	Conclusion	16
2	Analysis and Specification	17
2.1	Introduction	17
2.2	Actors Identification	17
2.3	Functional Requirements	19
2.3.1	Agent Requirements (RF-AGT)	19
2.3.2	Backend Server Requirements (RF-SRV)	20
2.3.3	Frontend Requirements (RF-UI)	21
2.4	Non-Functional Requirements	22
2.5	Use Case Modeling	22
2.5.1	Global Use Case Diagram	22
2.5.2	Detailed Use Case Descriptions	24
2.5.3	Use Case Prioritization Matrix	25
2.6	Behavioral Modeling	26
2.6.1	Sequence Diagram: Authentication Flow	26

2.6.2	Sequence Diagram: RAG Chat Workflow	26
2.6.3	Sequence Diagram: Log Ingestion with Threat Detection	27
2.6.4	Activity Diagram: Threat Detection Engine	28
2.7	Structural Modeling	29
2.7.1	Class Diagram: Domain Model	29
2.7.2	Entity Descriptions	31
2.8	Architectural Design	31
2.8.1	High-Level Architecture	31
2.8.2	Component Architecture	32
2.8.3	Deployment Architecture	33
2.8.4	Data Flow Diagram	34
2.9	Database Design	35
2.9.1	Physical Database Schema	35
2.9.2	Indexing Strategy	36
2.10	Security Architecture	36
2.10.1	Authentication Flow	36
2.10.2	Role-Based Access Control	37
2.11	API Design	38
2.11.1	RESTful Endpoint Summary	38
2.12	Conclusion	38
3	Implementation	39
3.1	Introduction	39
3.2	Development Environment	39
3.2.1	Hardware Configuration	39
3.2.2	Software Tools	40

3.3	The Golang Agent	40
3.3.1	Project Architecture	40
3.3.2	Directory Structure	41
3.3.3	Collector Interface Pattern	42
3.3.4	Windows Event Log Collection	44
3.3.5	File Tailing Implementation	46
3.3.6	Resilient Sender with Buffering	47
3.3.7	Cross-Compilation	48
3.4	Backend API Server	49
3.4.1	Project Structure	49
3.4.2	Express Application Setup	50
3.4.3	Authentication Service	52
3.4.4	Threat Detection Engine	54
3.4.5	RAG Chat Implementation	56
3.4.6	Server-Sent Events Implementation	58
3.5	Frontend Dashboard	59
3.5.1	Project Structure	59
3.5.2	Real-time Dashboard with SSE	60
3.5.3	Stats Cards Component	61
3.5.4	AI Chat Interface	64
3.6	Prisma Database Schema	65
3.7	Conclusion	66
4	Testing and Deployment	67
4.1	Introduction	67
4.2	Testing Strategy Overview	67

4.3	Unit Testing	69
4.3.1	Threat Detection Engine Tests	69
4.3.2	Authentication Service Tests	71
4.4	Integration Testing	72
4.4.1	API Endpoint Testing	72
4.4.2	SSE Stream Testing	74
4.5	Performance Testing	75
4.5.1	Load Testing with Apache Benchmark	75
4.5.2	Performance Benchmarks Summary	76
4.6	Docker Deployment	77
4.6.1	Docker Compose Configuration	77
4.6.2	Backend Dockerfile	80
4.6.3	Frontend Dockerfile	81
4.7	Agent Deployment	82
4.7.1	Windows Installation	82
4.7.2	Linux Installation	83
4.8	Production Deployment Checklist	85
4.9	Application Screenshots	85
4.10	Conclusion	89
General Conclusion		91
References		96
A Appendix A: Diagram Reference		97
A.1 Diagrams Checklist		97
B Appendix B: API Reference		98

B.1 Authentication Endpoints	98
B.2 Log Management Endpoints	99
B.3 Chat Endpoints	99
B.4 Request/Response Examples	100
B.4.1 Login Request	100
B.4.2 Log Ingestion Request	100
C Appendix C: Configuration Reference	102
C.1 Agent Configuration (config.yaml)	102
C.2 Environment Variables	104
D Appendix D: Glossary	105

List of Figures

1	Enterprise Log Volume Growth (2015-2025)	2
1.1	Evolution of Log Management Technologies	6
1.2	Elastic Stack (ELK) Architecture	7
1.3	Splunk Enterprise Dashboard Example	8
1.4	SIEM Solutions Radar Comparison	10
1.5	Node.js Event Loop Architecture	11
1.6	PostgreSQL Indexing Strategy for Logs	13
1.7	Binary Size Comparison by Language	14
1.8	Ollama Local LLM Architecture	15
1.9	Scrum Sprint Timeline and Milestones	15
2.1	System Context Diagram with Actors	18
2.2	Global Use Case Diagram	23
2.3	Authentication Sequence Diagram	26
2.4	RAG Chat Workflow Sequence Diagram	27
2.5	Log Ingestion Sequence Diagram	28
2.6	Threat Detection Activity Diagram	29
2.7	Domain Model Class Diagram	30
2.8	Global System Architecture	32
2.9	Component Architecture Diagram	33
2.10	Deployment Architecture Diagram	34

2.11 Data Flow Diagram (DFD Level 1)	34
2.12 PostgreSQL Physical Database Schema	35
2.13 JWT Authentication Architecture	37
3.1 Golang Agent Internal Architecture	41
3.2 Collector Class Hierarchy	44
3.3 Windows Event Log Collection Flow	46
3.4 Sender State Machine	48
3.5 Backend Component Architecture	52
3.6 Threat Detection Pattern Categories	56
3.7 RAG Workflow Detailed Diagram	58
3.8 Dashboard Screenshot	64
3.9 AI Chat Interface Screenshot	65
4.1 Testing Pyramid Strategy	68
4.2 Test Coverage Report	71
4.3 Performance Test Charts	77
4.4 Docker Network Topology	80
4.5 Dashboard Overview Screenshot	86
4.6 AI Chat Interface Screenshot	87
4.7 Log Detail View Screenshot	87
4.8 Admin Panel Screenshot	88
4.9 Alert Management Screenshot	89

List of Tables

1.1	Elastic Stack Comprehensive Evaluation	7
1.2	Splunk Enterprise Evaluation	8
1.3	Wazuh Platform Evaluation	9
1.4	Microsoft Sentinel Evaluation	9
1.5	Comprehensive SIEM Solution Comparison	10
1.6	Frontend Technology Comparison	12
1.7	Agent Runtime Comparison	13
1.8	Supported LLM Models	14
2.1	System Actors and Responsibilities	17
2.2	Golang Agent Functional Requirements	19
2.3	Backend Server Functional Requirements	20
2.4	Frontend Functional Requirements	21
2.5	Non-Functional Requirements Specification	22
2.6	Use Case UC-01: User Authentication	24
2.7	Use Case UC-04: Chat with AI Assistant	25
2.8	Use Case Prioritization (MoSCoW)	25
2.9	Entity Attribute Specifications	31
2.10	Database Indexing Strategy	36
2.11	RBAC Permission Matrix	37
2.12	API Endpoint Overview	38

3.1	Development Environment Specifications	39
3.2	Development Tools and Versions	40
4.1	Testing Strategy by Component	69
4.2	Threat Detection Test Cases	69
4.3	Performance Benchmark Results	76
4.4	Production Deployment Checklist	85
4.5	Project Objectives Fulfillment Matrix	92
4.6	Technical Challenges and Resolutions	93
4.7	LogChat Development Roadmap	94
A.1	Required Diagrams Checklist	97
B.1	Authentication API Endpoints	98
B.2	Log Management API Endpoints	99
B.3	AI Chat API Endpoints	99
C.1	Backend Environment Variables	104

List of Abbreviations

Abbreviation	Definition
AI	Artificial Intelligence
API	Application Programming Interface
CRUD	Create, Read, Update, Delete
CSS	Cascading Style Sheets
DB	Database
DFD	Data Flow Diagram
Docker	Container Runtime Platform
ERD	Entity-Relationship Diagram
GDPR	General Data Protection Regulation
HIPAA	Health Insurance Portability and Accountability Act
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HTTP Secure
IDS	Intrusion Detection System
JSON	JavaScript Object Notation
JWT	JSON Web Token
KQL	Kusto Query Language
LLM	Large Language Model
NLP	Natural Language Processing
ORM	Object-Relational Mapping
PCI-DSS	Payment Card Industry Data Security Standard
RBAC	Role-Based Access Control
RAG	Retrieval Augmented Generation
REST	Representational State Transfer
SIEM	Security Information and Event Management

Abbreviation	Definition
SOC	Security Operations Center
SPL	Search Processing Language (Splunk)
SQL	Structured Query Language
SSE	Server-Sent Events
TLS	Transport Layer Security
UI	User Interface
UML	Unified Modeling Language
UUID	Universally Unique Identifier
UX	User Experience
XSS	Cross-Site Scripting
YAML	YAML Ain't Markup Language

General Introduction

Context and Motivation

In the contemporary digital landscape, cybersecurity has emerged as one of the most critical challenges facing organizations worldwide. According to IBM's Cost of a Data Breach Report 2024, the average cost of a data breach reached \$4.88 million globally, with organizations taking an average of 277 days to identify and contain a breach. In this context, effective log management and security monitoring have become indispensable components of any organization's security posture.

Application logs constitute the primary source of truth for system health, security posture, and operational intelligence. The average enterprise generates terabytes of log data daily across its infrastructure — from web servers and databases to authentication systems and network devices. Figure 1 illustrates the exponential growth in enterprise log volumes over the past decade.

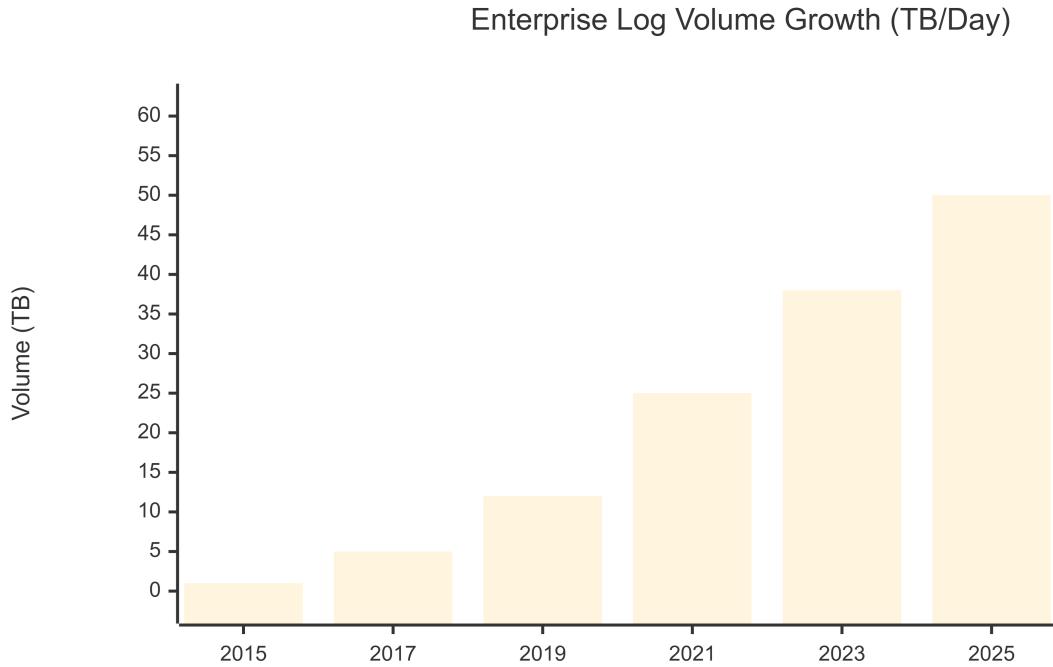


Figure 1: Enterprise Log Volume Growth (2015-2025)

This data, when properly analyzed, reveals critical insights: performance bottlenecks, security breaches, compliance violations, and emerging threats. However, the sheer velocity, volume, and heterogeneity of logs generated by modern infrastructure create what security professionals term the “data noise” problem.

Problem Statement

Small to Medium Enterprises (SMEs) and resource-constrained security teams face a critical “Security Gap” characterized by three fundamental barriers:

The Three Barriers to Effective Log Management

- 1. Commercial Barriers:** Enterprise SIEM solutions like Splunk, IBM QRadar, and Microsoft Sentinel command licensing fees ranging from \$50,000 to \$500,000+ annually, effectively pricing out smaller organizations from effective security monitoring.
- 2. Technical Skill Gap:** Effective log analysis requires proficiency in propri-

etary query languages (SPL for Splunk, KQL for Azure Sentinel, Lucene for Elasticsearch). Junior analysts and developers often lack this specialized knowledge, creating a dependency on expensive senior talent.

3. **Alert Fatigue:** Traditional rule-based detection systems generate excessive false positives, overwhelming security teams and causing genuine threats to be overlooked. Studies indicate that SOC analysts spend up to 45% of their time on false positive investigations.

Project Objectives

LogChat was conceived to bridge this gap by building an open-source, AI-first log management platform with the following objectives:

1. **Unified Collection:** Develop a single-binary agent (Golang) compatible with Windows Event Logs, Linux Syslog/Journald, and file-based logs — deployable across heterogeneous infrastructure without runtime dependencies.
2. **Intelligent Analysis:** Replace complex query languages with Natural Language Chat, powered by local Large Language Models (LLMs) for privacy-preserving inference using the RAG pattern.
3. **Real-time Visualization:** Provide an interactive, streaming dashboard for immediate situational awareness with sub-second update latency.
4. **Zero-Configuration Deployment:** Enable complete platform deployment via a single `docker-compose up` command, achieving operational status within five minutes.
5. **Privacy-First Design:** Ensure that sensitive log data never leaves the organization's infrastructure by utilizing local AI inference.

Document Structure

This report is organized into five chapters, each addressing a specific phase of the software development lifecycle:

- **Chapter 1: State of the Art** presents a comprehensive analysis of existing SIEM solutions, evaluates technological alternatives, and provides justification for the selected technology stack.
- **Chapter 2: Analysis & Specification** details the functional and non-functional requirements, presents UML modeling artifacts including use case, sequence, and class diagrams, and describes the architectural design decisions.
- **Chapter 3: Implementation** provides a technical deep-dive into the development of the Golang Agent, Backend API, RAG Engine, and Frontend components with annotated code excerpts.
- **Chapter 4: Testing & Deployment** outlines the quality assurance strategy, describes the Docker-based deployment architecture, and presents performance benchmarks.
- **General Conclusion** summarizes achievements, reflects on lessons learned, and outlines the future roadmap for LogChat development.

Chapter 1

State of the Art

1.1 Introduction

Before embarking on the development of LogChat, a comprehensive analysis of the current log management and SIEM landscape was essential. This chapter surveys existing solutions, evaluates technological alternatives, and provides the rationale for our architectural decisions. The insights gathered during this phase directly influenced the design patterns and technology choices implemented in LogChat.

1.2 Historical Evolution of Log Management

1.2.1 Early Approaches (1990s-2000s)

In the early days of computing, log management was a manual process. System administrators would periodically review text files using command-line tools such as `grep`, `awk`, and `tail`. Figure ?? presents the evolution of log management technologies.

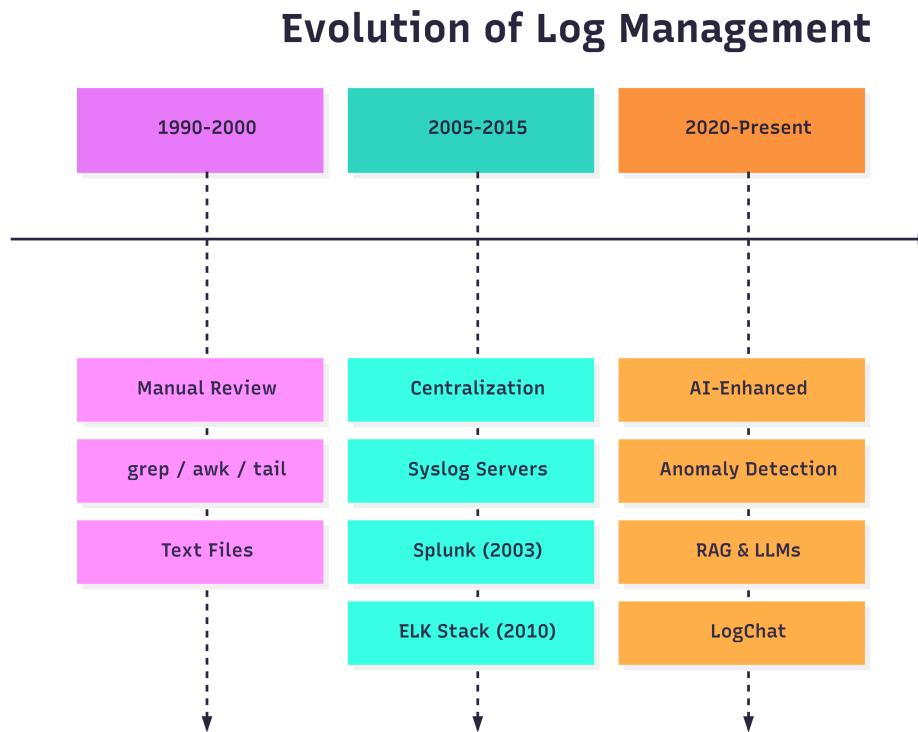


Figure 1.1: Evolution of Log Management Technologies

1.2.2 Centralized Logging (2005-2015)

The emergence of distributed systems necessitated centralized logging solutions. Syslog servers became standard, and tools like Splunk (2003) and the ELK Stack (2010) emerged to address the growing complexity.

1.2.3 Modern AI-Enhanced SIEM (2020-Present)

The current generation of SIEM solutions leverages machine learning for anomaly detection and, increasingly, large language models for natural language querying. LogChat positions itself at the forefront of this evolution.

1.3 Market Analysis of Existing Solutions

1.3.1 Elastic Stack (ELK)

The Elastic Stack — comprising Elasticsearch, Logstash, and Kibana — represents the industry standard for open-source log management. Figure ?? illustrates its architecture.

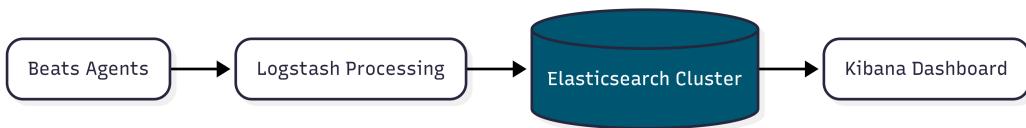


Figure 1.2: Elastic Stack (ELK) Architecture

Table 1.1: Elastic Stack Comprehensive Evaluation

Strengths	Highly scalable horizontal architecture, powerful Kibana visualization, extensive plugin ecosystem, large community support
Weaknesses	Resource-intensive JVM-based architecture, complex cluster management, steep learning curve for Lucene/DSL queries
Hardware Requirements	Minimum 3 nodes, 16GB+ RAM per node, SSD storage recommended
Licensing	Open Source (Basic), Commercial (Platinum: \$125/node/month)

1.3.2 Splunk Enterprise

Splunk remains the enterprise leader in SIEM, commanding over 30% market share in the security analytics segment.

Table 1.2: Splunk Enterprise Evaluation

Strengths	Comprehensive ecosystem, advanced ML-based analytics, excellent enterprise support, extensive app marketplace
Weaknesses	Proprietary platform, extremely expensive pricing model, vendor lock-in concerns
Pricing Model	\$1,800+ per GB/day ingested (perpetual), or subscription model
Annual Cost (10GB/day)	Approximately \$180,000 - \$250,000

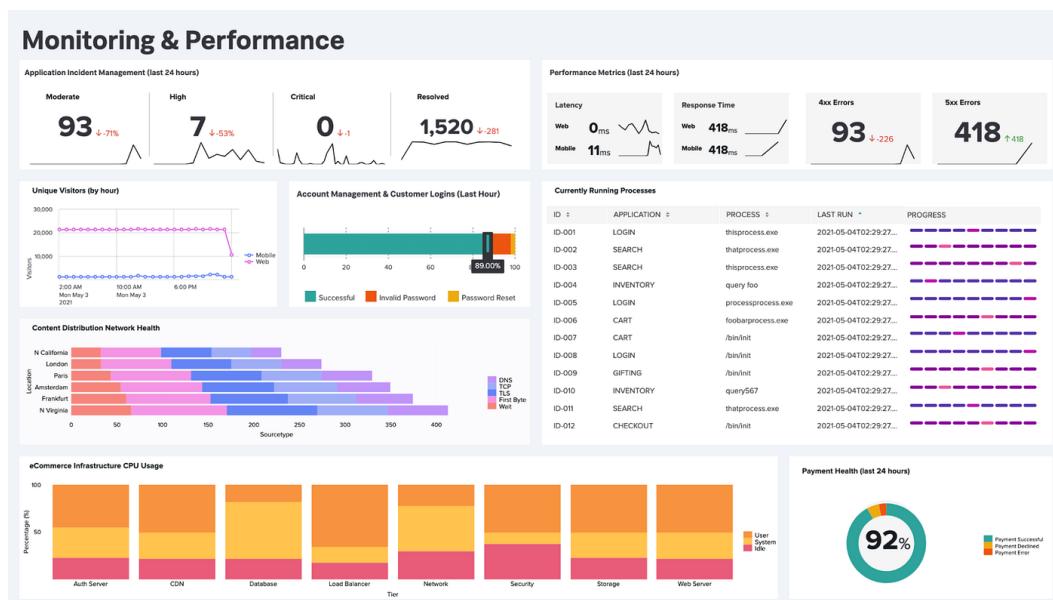


Figure 1.3: Splunk Enterprise Dashboard Example

1.3.3 Wazuh

Wazuh is an open-source security platform focusing on intrusion detection, file integrity monitoring, and compliance.

Table 1.3: Wazuh Platform Evaluation

Strengths	Strong Host-based IDS capabilities, file integrity monitoring, MITRE ATT&CK mapping, regulatory compliance modules
Weaknesses	Complex UI/UX, agent-heavy architecture, challenging customization, limited NLP capabilities
Deployment	Requires Elasticsearch backend, complex multi-component setup

1.3.4 Microsoft Sentinel

Microsoft Sentinel provides cloud-native SIEM integrated with the Azure ecosystem.

Table 1.4: Microsoft Sentinel Evaluation

Strengths	Cloud-native scalability, tight Azure integration, SOAR capabilities, KQL query language
Weaknesses	Azure lock-in, complex pricing, limited on-premises support, data residency concerns
Pricing	Pay-per-use: \$2.46/GB ingested + \$0.10/GB retained

1.3.5 Comparative Analysis

Table 1.5 provides a comprehensive comparison of evaluated solutions against LogChat's proposed capabilities.

Table 1.5: Comprehensive SIEM Solution Comparison

Criterion	ELK	Splunk	Wazuh	Sentinel	LogChat
Open Source	✓	✗	✓	✗	✓
Natural Language Query	✗	Partial	✗	✗	✓
Single-Binary Agent	✗	✗	✗	N/A	✓
Local AI (Privacy)	✗	✗	✗	✗	✓
One-Command Deploy	✗	✗	✗	✗	✓
Real-time Streaming	✓	✓	✓	✓	✓
MITRE ATT&CK Mapping	Plugin	✓	✓	✓	✓
Min. RAM Requirement	16GB	8GB	8GB	N/A	4GB

Figure ?? presents a radar chart visualization of the comparison.

SIEM Market Positioning

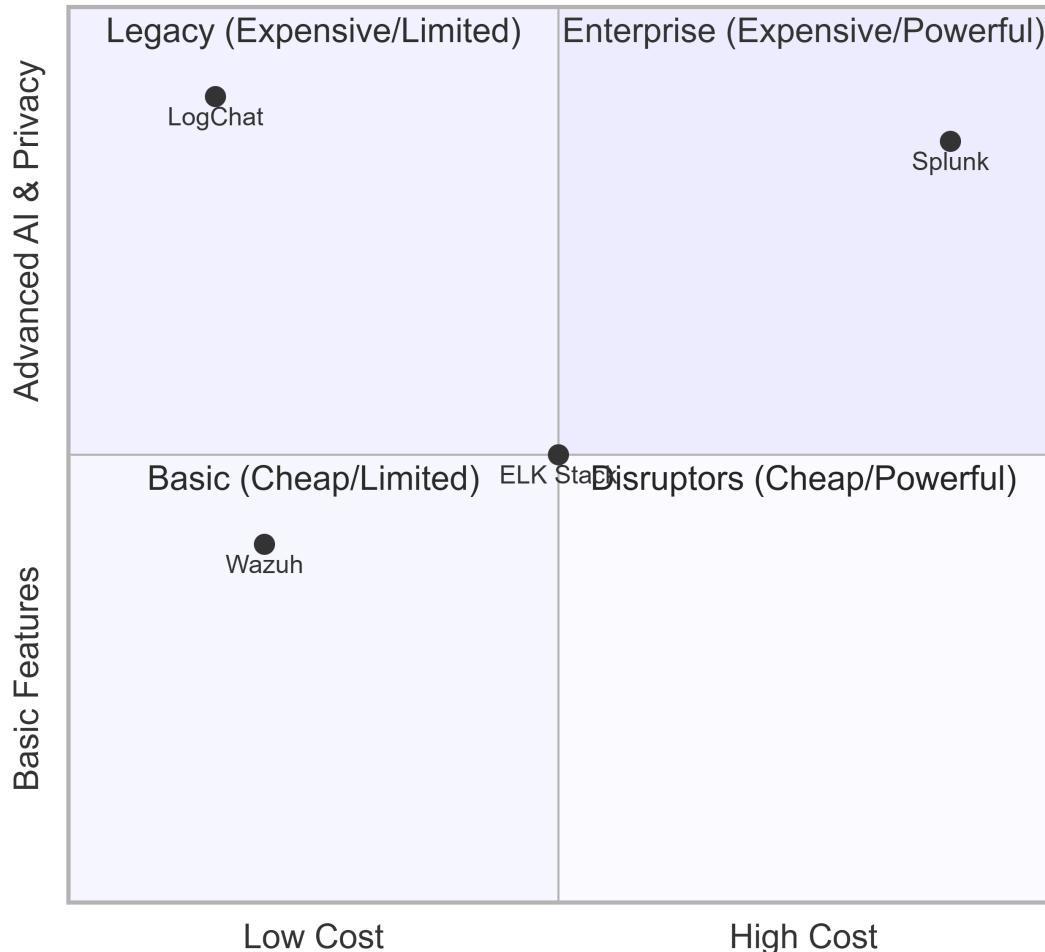


Figure 1.4: SIEM Solutions Radar Comparison

1.4 Technology Stack Selection

Based on the market analysis and project requirements, the following technology stack was selected. Each choice was driven by specific technical and business considerations.

1.4.1 Backend: Node.js with TypeScript

Node.js was selected for the backend API server for the following reasons:

Node.js Selection Rationale

- **Event-Driven Architecture:** Non-blocking I/O model perfectly suited for handling high-concurrency log ingestion and SSE streaming
- **npm Ecosystem:** Access to mature libraries including Express, Prisma, and Zod
- **TypeScript Integration:** Compile-time type safety critical for maintaining data integrity
- **Shared Language:** Same language as frontend reduces context switching and enables code sharing

Figure ?? illustrates the Node.js event loop architecture.

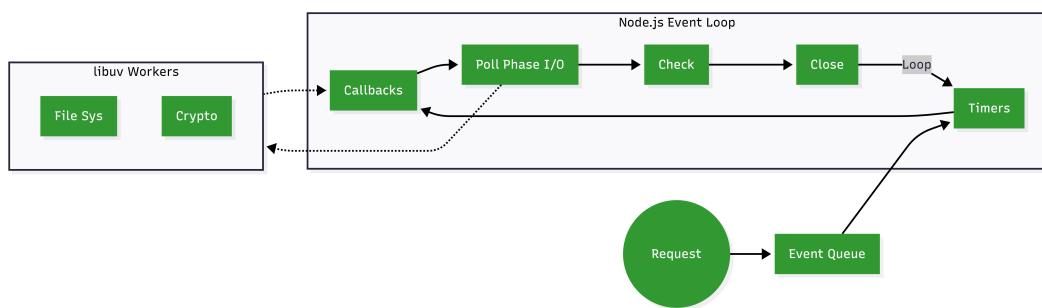


Figure 1.5: Node.js Event Loop Architecture

1.4.2 Frontend: Next.js 14 with React 18

Next.js 14 was chosen for its hybrid rendering capabilities and modern React features.

Table 1.6: Frontend Technology Comparison

Feature	Next.js	Remix	Vite+React	Angular
SSR/SSG	✓	✓	Plugin	✓
App Router	✓	✓	✗	✓
Bundle Size	Small	Small	Custom	Large
Learning Curve	Moderate	Moderate	Low	High
Community	Large	Growing	Large	Large

1.4.3 Database: PostgreSQL 16 with Prisma ORM

PostgreSQL was selected as the primary database for its robustness and advanced features.

PostgreSQL 16 Key Features

- **JSONB Columns:** Native support for semi-structured log metadata
- **Full-Text Search:** Built-in text search for log message querying
- **Time-Series Indexing:** Efficient B-tree and BRIN indexes for timestamp queries
- **pgvector Compatibility:** Future-proof for semantic search implementation
- **ACID Compliance:** Guaranteed data integrity for audit requirements

Figure ?? illustrates the indexing strategy employed for log queries.

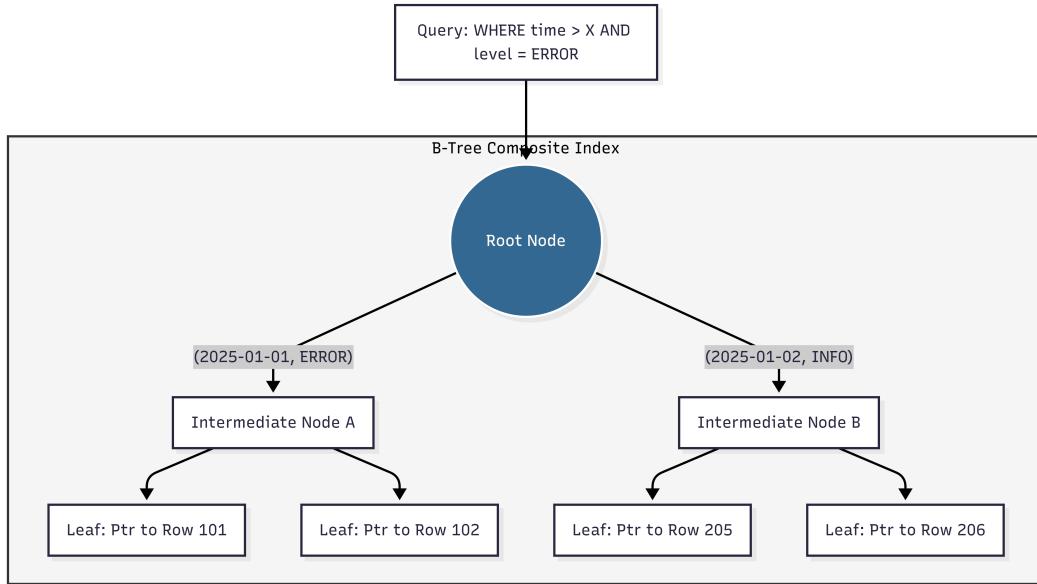


Figure 1.6: PostgreSQL Indexing Strategy for Logs

1.4.4 Agent: Golang

For the distributed log collector, Golang was selected over Python, Java, and Rust:

Table 1.7: Agent Runtime Comparison

Feature	Python	Java	Rust	Go
Single Binary	✗	✗	✓	✓
No Runtime Dependencies	✗	✗	✓	✓
Windows API Access	Complex	Complex	Moderate	✓
Memory Footprint	High	High	Low	Low
Goroutine/Async	asyncio	Threads	async	✓
Cross-Compilation	Complex	Complex	Moderate	Built-in
Learning Curve	Low	Moderate	High	Moderate

Figure ?? compares binary sizes across languages for equivalent functionality.

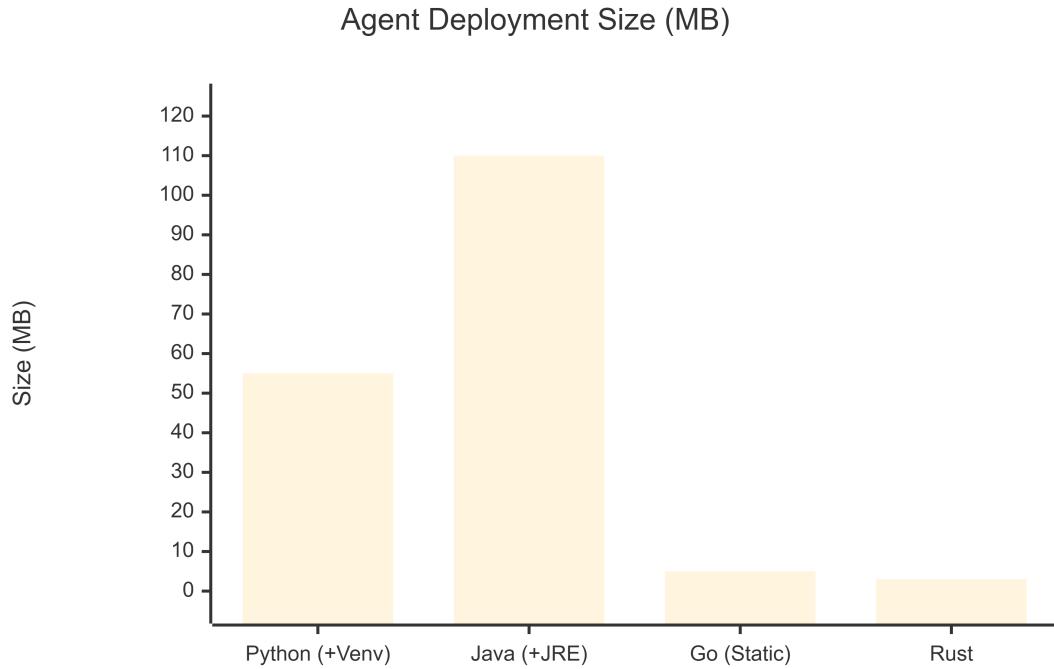


Figure 1.7: Binary Size Comparison by Language

1.4.5 AI Engine: Ollama with Local LLMs

To ensure data privacy, cloud-based AI APIs were ruled out for sensitive environments. Ollama enables running quantized models locally without internet connectivity.

Table 1.8: Supported LLM Models

Model	Parameters	VRAM Required	Use Case
Qwen 2.5:0.5b	0.5B	1GB	Fast responses, basic analysis
Qwen 2.5:3b	3B	4GB	Balanced performance
Llama 3.1:8b	8B	8GB	Complex analysis
Mistral 7B	7B	6GB	European compliant

Figure ?? shows the Ollama integration architecture.

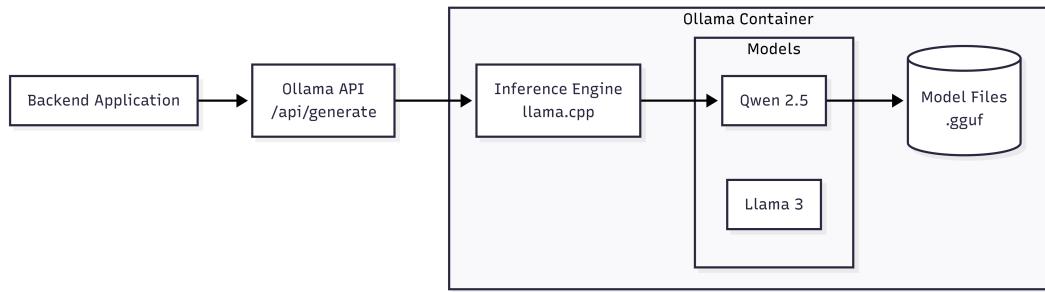


Figure 1.8: Ollama Local LLM Architecture

1.5 Development Methodology

The project followed **Scrum**, an Agile methodology well-suited for iterative development:

- **Sprint Duration:** 1 week
- **Total Sprints:** 12 (3 months)
- **Team Size:** 2 developers
- **Tools:** Git, GitHub, Docker, VS Code

Figure ?? presents the sprint timeline and deliverables.

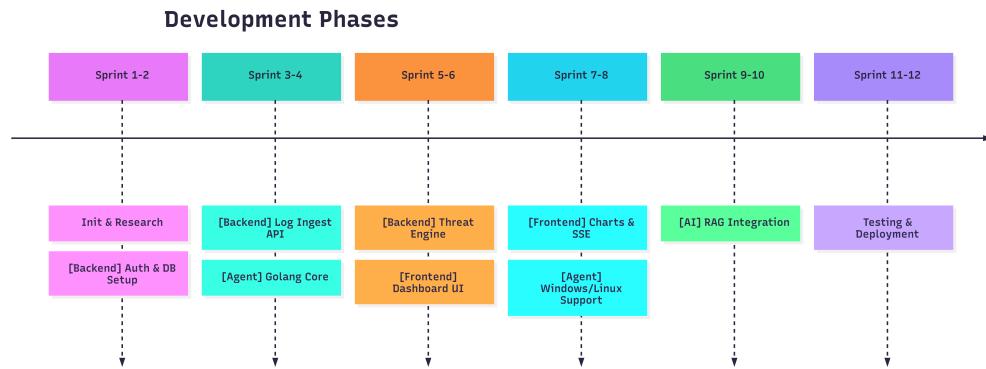


Figure 1.9: Scrum Sprint Timeline and Milestones

1.6 Conclusion

This state-of-the-art analysis confirmed the viability of building a competitive, open-source SIEM alternative. The selected technology stack balances developer productivity, runtime performance, and deployment simplicity. The following chapter details the functional and non-functional requirements derived from this analysis.

Chapter 2

Analysis and Specification

2.1 Introduction

This chapter translates business requirements into formal technical specifications using UML (Unified Modeling Language). We present the actors, use cases, data models, sequence diagrams, and architectural blueprints that guide the implementation phase. The artifacts produced in this chapter serve as the contractual specification between stakeholders.

2.2 Actors Identification

The system interacts with three primary actors, each with distinct roles and permissions:

Table 2.1: System Actors and Responsibilities

Actor	Type	Responsibilities
Security Analyst	Human (Primary)	Monitor dashboards, investigate alerts, query logs via AI chat, export reports
System Administrator	Human (Primary)	Configure log sources, manage users, define alert rules, review audit trails
LogChat Agent	System (Automated)	Collect logs from sources, transmit to server, buffer during outages

Figure ?? presents the context diagram showing actor interactions.

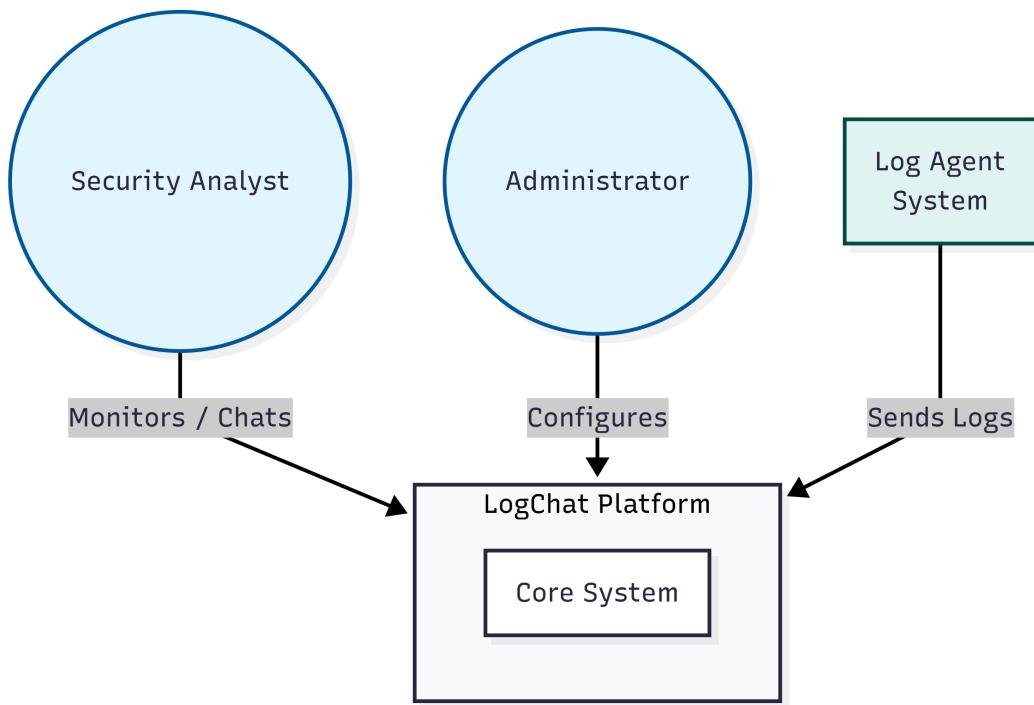


Figure 2.1: System Context Diagram with Actors

2.3 Functional Requirements

2.3.1 Agent Requirements (RF-AGT)

Table 2.2: Golang Agent Functional Requirements

ID	Requirement Description	Priority
RF-AGT-01	Compile to static binaries for Windows (amd64, 386) and Linux (amd64, arm64)	High
RF-AGT-02	Monitor log files using tail-like behavior (follow mode with rotation support)	High
RF-AGT-03	Collect Windows Event Logs from Application, System, Security channels	High
RF-AGT-04	Collect Linux logs via Journald (systemd) and traditional Syslog	High
RF-AGT-05	Buffer logs locally using ring buffer if server is unreachable (resilience)	Medium
RF-AGT-06	Authenticate with server via X-API-Key header	High
RF-AGT-07	Support JSON, regex, and raw log parsing formats	Medium
RF-AGT-08	Provide health endpoint for monitoring agent status	Low
RF-AGT-09	Support hot configuration reload without restart	Low

2.3.2 Backend Server Requirements (RF-SRV)

Table 2.3: Backend Server Functional Requirements

ID	Requirement Description	Priority
RF-SRV-01	Provide REST API endpoints for single and batch log ingestion	High
RF-SRV-02	Implement real-time threat detection using regex pattern matching	High
RF-SRV-03	Support natural language queries via AI chat endpoint with RAG context	High
RF-SRV-04	Provide SSE (Server-Sent Events) for real-time dashboard updates	High
RF-SRV-05	Implement JWT-based authentication with configurable expiry	High
RF-SRV-06	Enforce role-based access control (ADMIN, STAFF, USER)	High
RF-SRV-07	Maintain immutable audit logs for all administrative actions	Medium
RF-SRV-08	Support log export in CSV and JSON formats	Medium
RF-SRV-09	Provide API rate limiting per source	Medium
RF-SRV-10	Support multiple AI providers (Ollama, OpenAI, Anthropic, Gemini)	Medium

2.3.3 Frontend Requirements (RF-UI)

Table 2.4: Frontend Functional Requirements

ID	Requirement Description	Priority
RF-UI-01	Display real-time statistics cards (total logs, errors, warnings, threats)	High
RF-UI-02	Provide interactive time-series chart for log volume visualization	High
RF-UI-03	Implement paginated log table with inline expansion	High
RF-UI-04	Support filtering by time range, log level, service name, and text search	High
RF-UI-05	Provide AI chat interface with markdown rendering	High
RF-UI-06	Display toast notifications for real-time alerts	Medium
RF-UI-07	Implement dark mode toggle	Low
RF-UI-08	Provide responsive design for tablet devices	Medium

2.4 Non-Functional Requirements

Table 2.5: Non-Functional Requirements Specification

Category	ID	Requirement	Target
Performance	RNF-01	Log ingestion throughput	1000+ req/s
Performance	RNF-02	Dashboard update latency	< 500ms
Performance	RNF-03	AI response time (local)	< 5 seconds
Availability	RNF-04	System uptime	99.9%
Security	RNF-05	Password hashing algorithm	bcrypt (cost 12)
Security	RNF-06	API authentication mechanism	JWT (RS256)
Security	RNF-07	TLS version for external APIs	TLS 1.2+
Portability	RNF-08	Deployment method	Docker Compose
Portability	RNF-09	Supported agent platforms	Windows, Linux
Usability	RNF-10	Dashboard responsiveness	Desktop, Tablet
Maintainability	RNF-11	Code documentation coverage	80%+

2.5 Use Case Modeling

2.5.1 Global Use Case Diagram

Figure ?? presents the comprehensive use case diagram depicting all actors and their interactions with the LogChat platform.

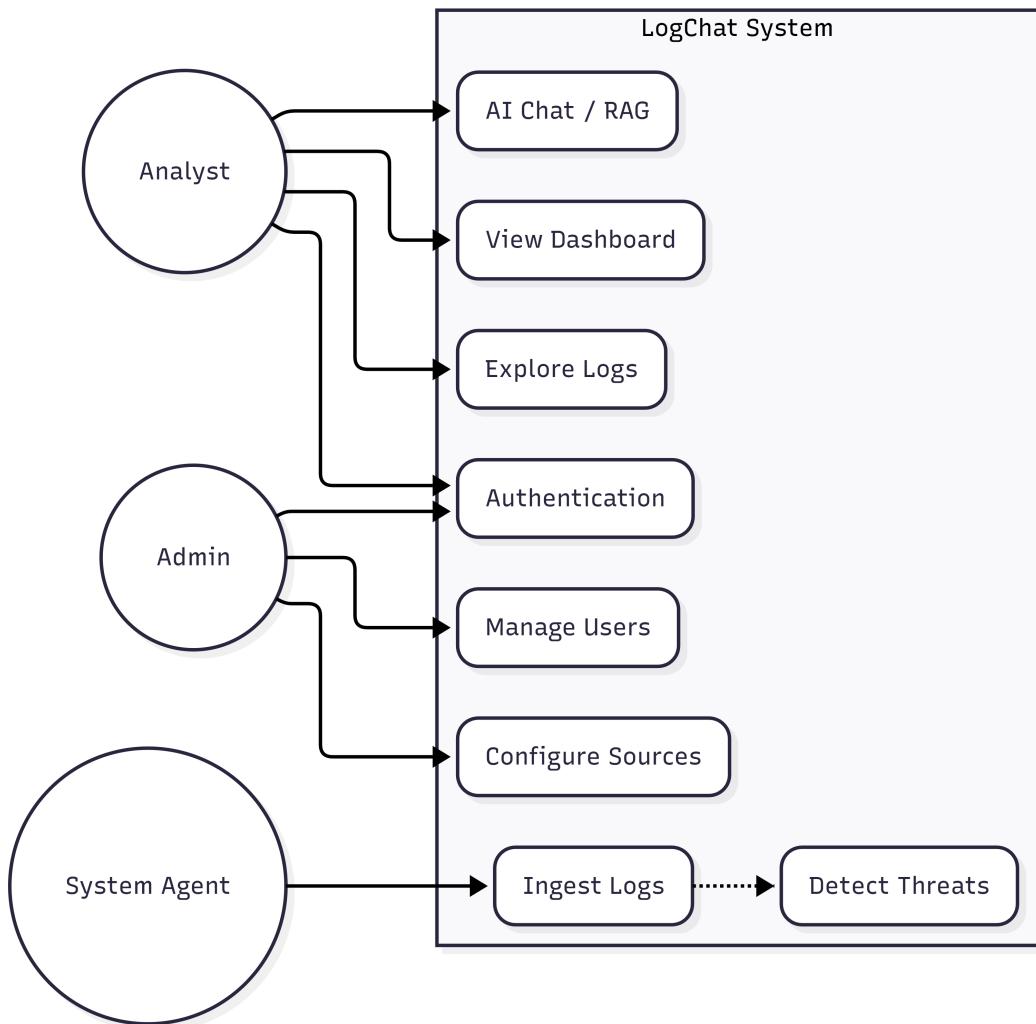


Figure 2.2: Global Use Case Diagram

2.5.2 Detailed Use Case Descriptions

UC-01: User Authentication

Table 2.6: Use Case UC-01: User Authentication

Use Case ID	UC-01
Name	User Authentication
Actor	Security Analyst, Administrator
Precondition	User has valid credentials in the system
Main Flow	<ol style="list-style-type: none"> 1. User navigates to login page 2. User enters email and password 3. System validates credentials 4. System generates JWT token 5. System creates session record 6. User is redirected to dashboard
Alternative Flow	A1. Invalid credentials: Display error, remain on login A2. Account disabled: Display account status message
Postcondition	User is authenticated with valid session

UC-04: Chat with AI Assistant

Table 2.7: Use Case UC-04: Chat with AI Assistant

Use Case ID	UC-04
Name	Chat with AI Assistant
Actor	Security Analyst
Precondition	User is authenticated; AI service is available
Main Flow	<ol style="list-style-type: none"> 1. User navigates to Chat interface 2. User enters natural language query 3. System retrieves relevant logs (RAG) 4. System constructs context-enriched prompt 5. System sends prompt to LLM 6. LLM generates analysis response 7. System displays formatted response
Alternative Flow	A1. AI unavailable: Display offline message with suggestions
Postcondition	Chat message and response are persisted

2.5.3 Use Case Prioritization Matrix

Table 2.8: Use Case Prioritization (MoSCoW)

Use Case	Category	Sprint
UC-01: Authentication	Must Have	1-2
UC-02: View Dashboard	Must Have	3-4
UC-03: Explore Logs	Must Have	3-4
UC-04: AI Chat	Must Have	5-6
UC-10: Ingest Logs	Must Have	2-3
UC-11: Threat Detection	Should Have	4-5
UC-06: Manage Users	Should Have	7-8
UC-05: Export Reports	Could Have	9

2.6 Behavioral Modeling

2.6.1 Sequence Diagram: Authentication Flow

Figure ?? details the complete authentication sequence from user input to dashboard redirect.

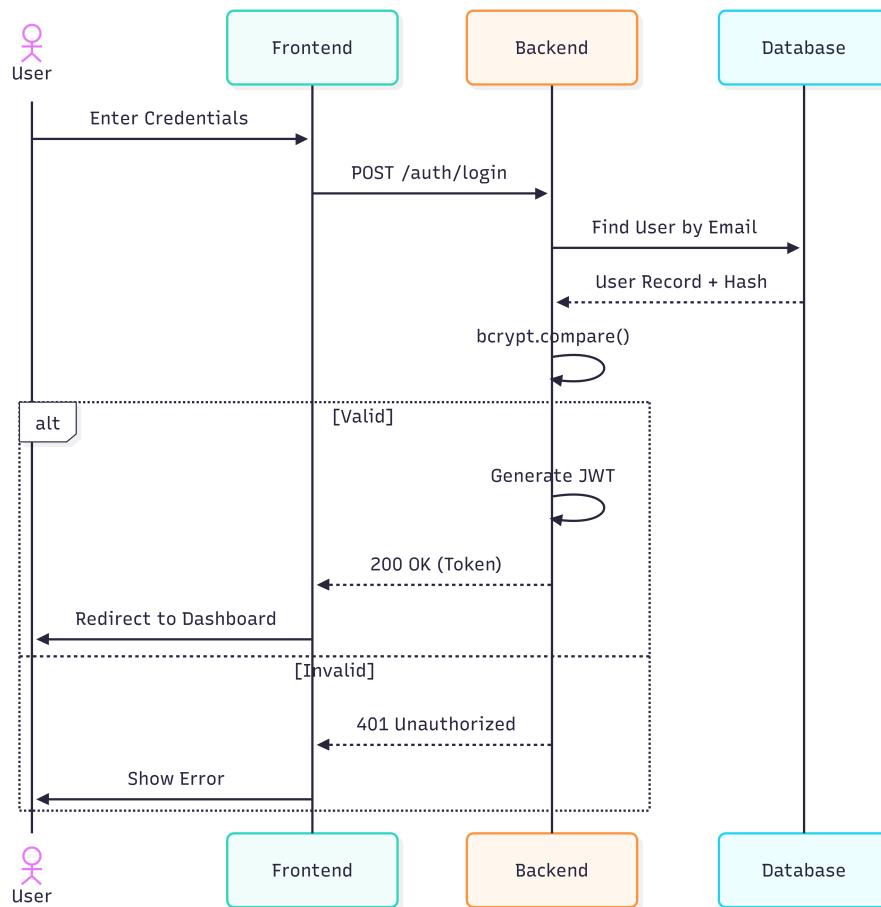


Figure 2.3: Authentication Sequence Diagram

2.6.2 Sequence Diagram: RAG Chat Workflow

Figure ?? illustrates the RAG (Retrieval Augmented Generation) process, which is central to LogChat's AI capabilities.

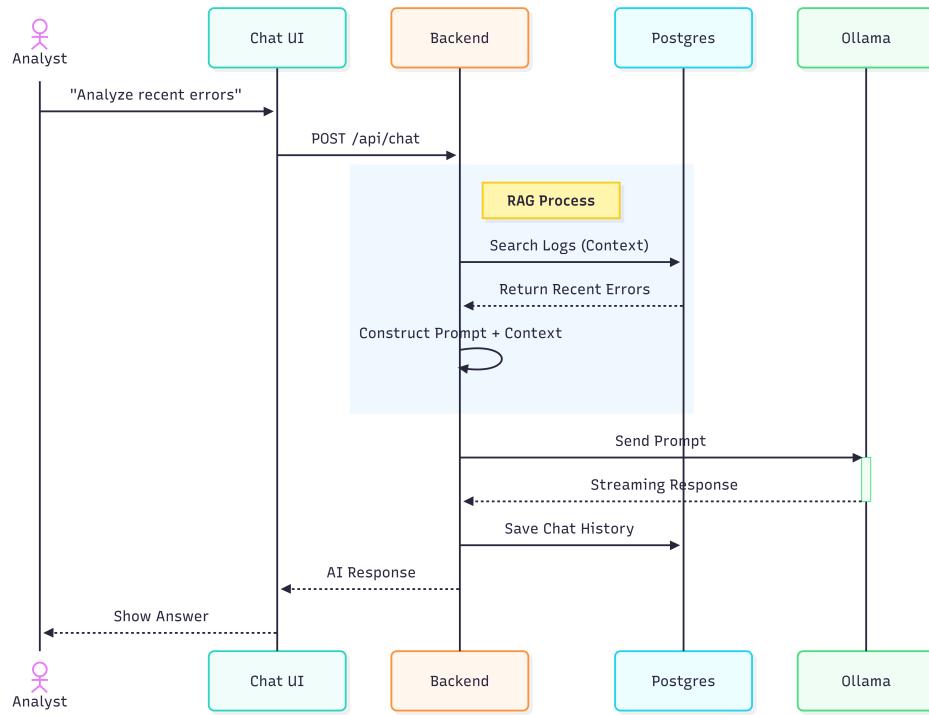


Figure 2.4: RAG Chat Workflow Sequence Diagram

2.6.3 Sequence Diagram: Log Ingestion with Threat Detection

Figure ?? presents the complete flow from agent transmission to threat detection and dashboard notification.

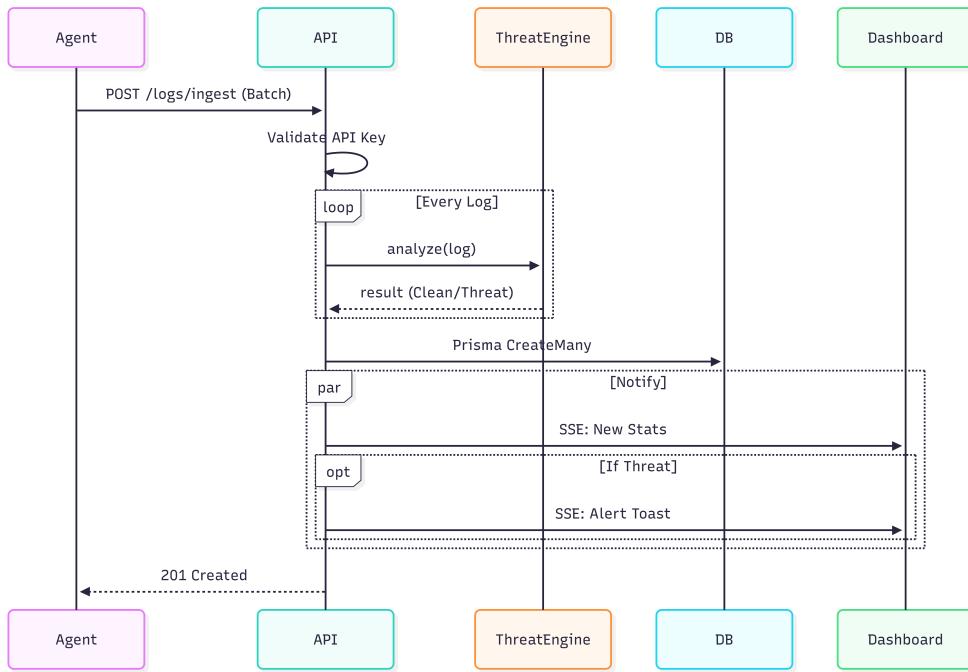


Figure 2.5: Log Ingestion Sequence Diagram

2.6.4 Activity Diagram: Threat Detection Engine

Figure ?? models the decision flow within the threat detection engine.

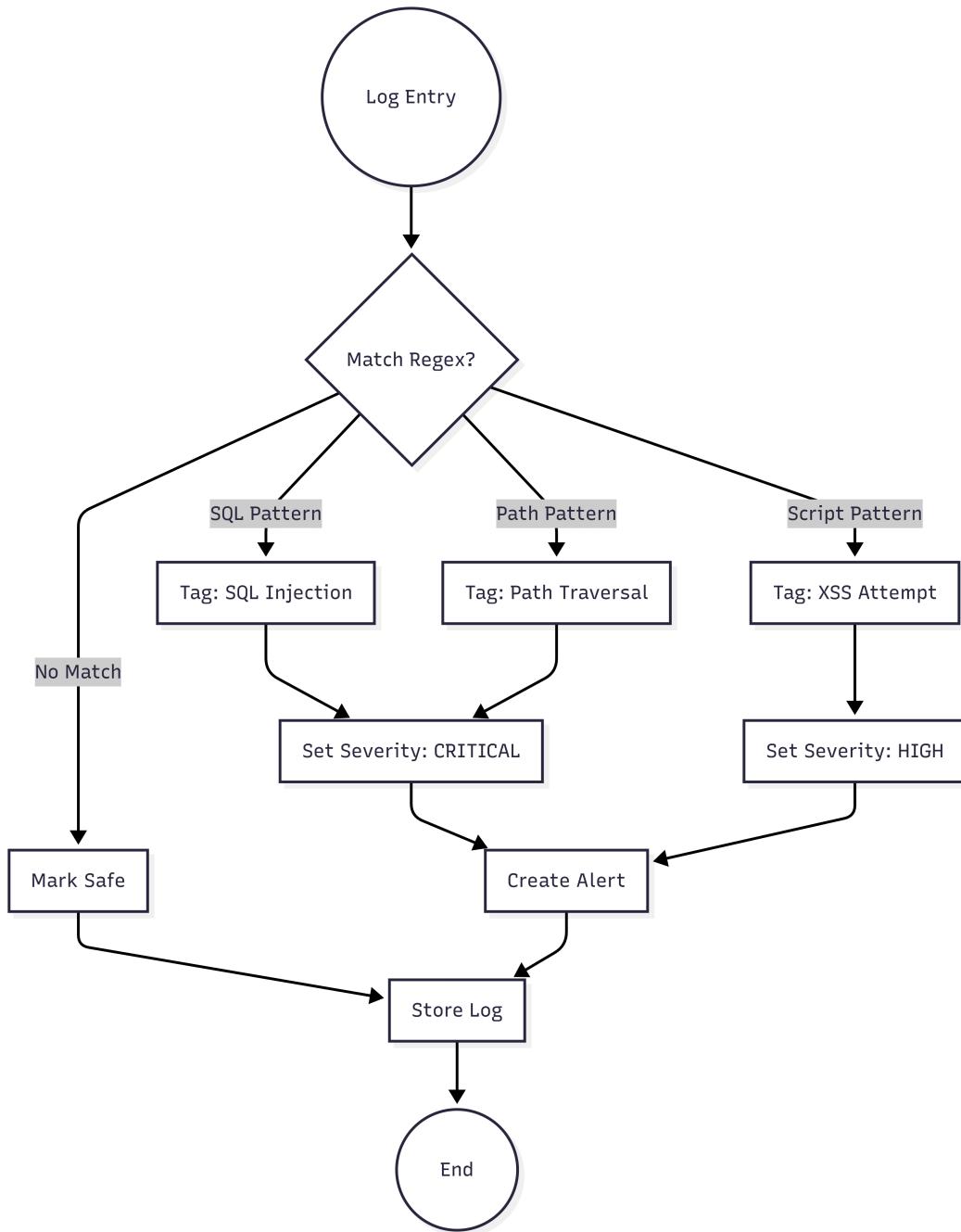


Figure 2.6: Threat Detection Activity Diagram

2.7 Structural Modeling

2.7.1 Class Diagram: Domain Model

Figure ?? presents the domain model with all entities, attributes, and relationships.

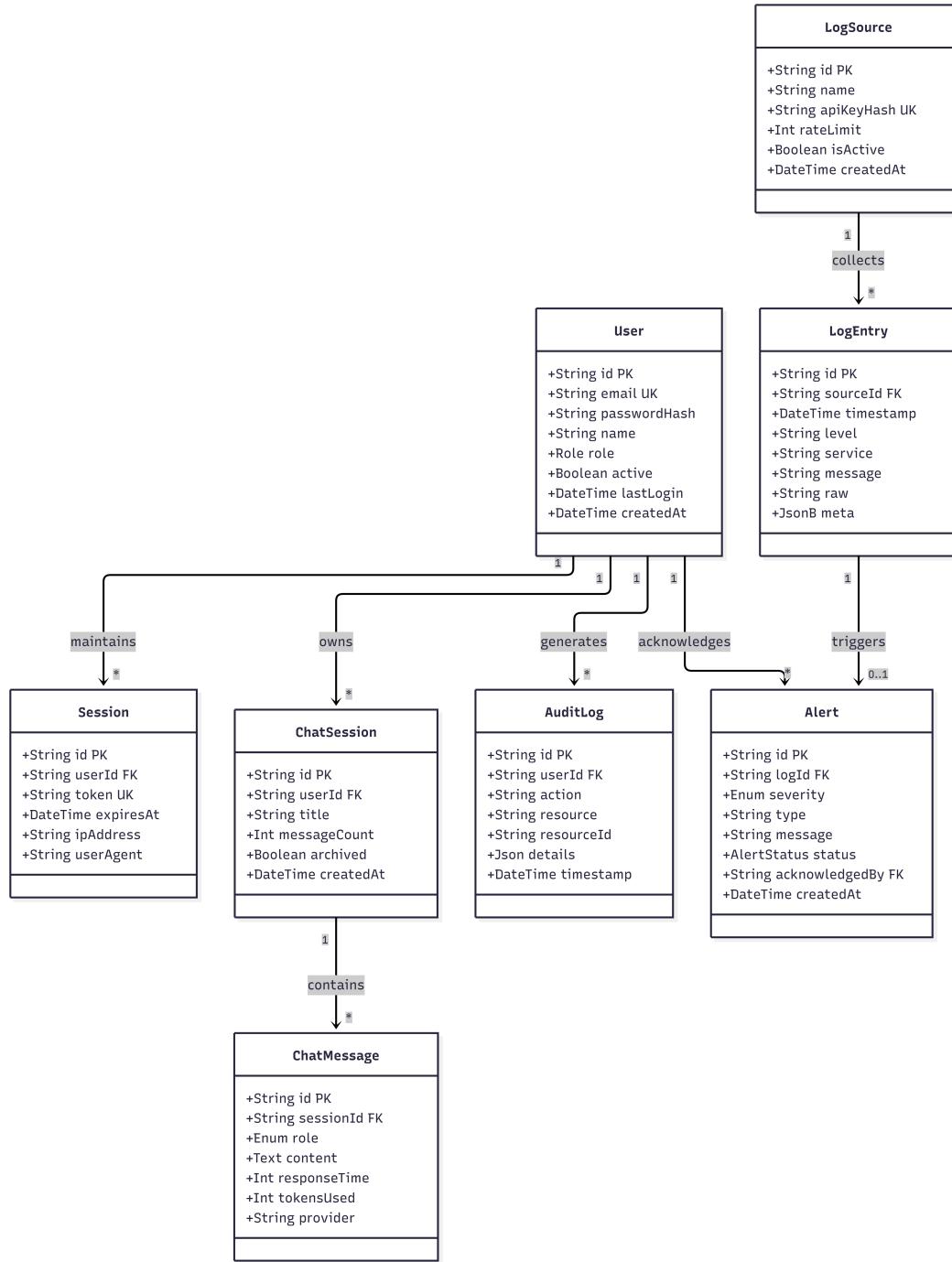


Figure 2.7: Domain Model Class Diagram

2.7.2 Entity Descriptions

Table 2.9: Entity Attribute Specifications

Entity	Key Attributes
User	id (CUID), email (unique), passwordHash, name, role (enum), active, lastLogin
Session	id, token (unique), userId (FK), expiresAt, ipAddress, userAgent
LogEntry	id, timestamp, level (enum), service, message, raw, meta (JSONB)
ChatSession	id, userId (FK), title, messageCount, archived, createdAt
ChatMessage	id, sessionId (FK), role (enum), content, responseTime, tokensUsed
Alert	id, logId (FK), severity (enum), type, message, status, acknowledgedBy
LogSource	id, name, apiKey (unique), apiKeyHash, rateLimit, isActive
AuditLog	id, userId (FK), action, resource, resourceId, details (JSON), timestamp

2.8 Architectural Design

2.8.1 High-Level Architecture

Figure ?? presents the global system architecture showing all components and their interactions.

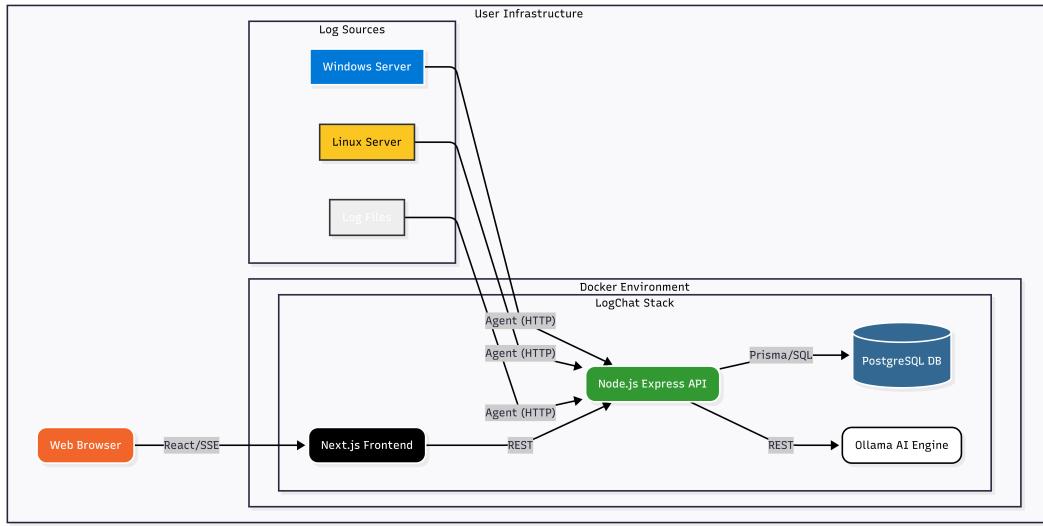


Figure 2.8: Global System Architecture

2.8.2 Component Architecture

Figure ?? details the internal structure of each containerized service.

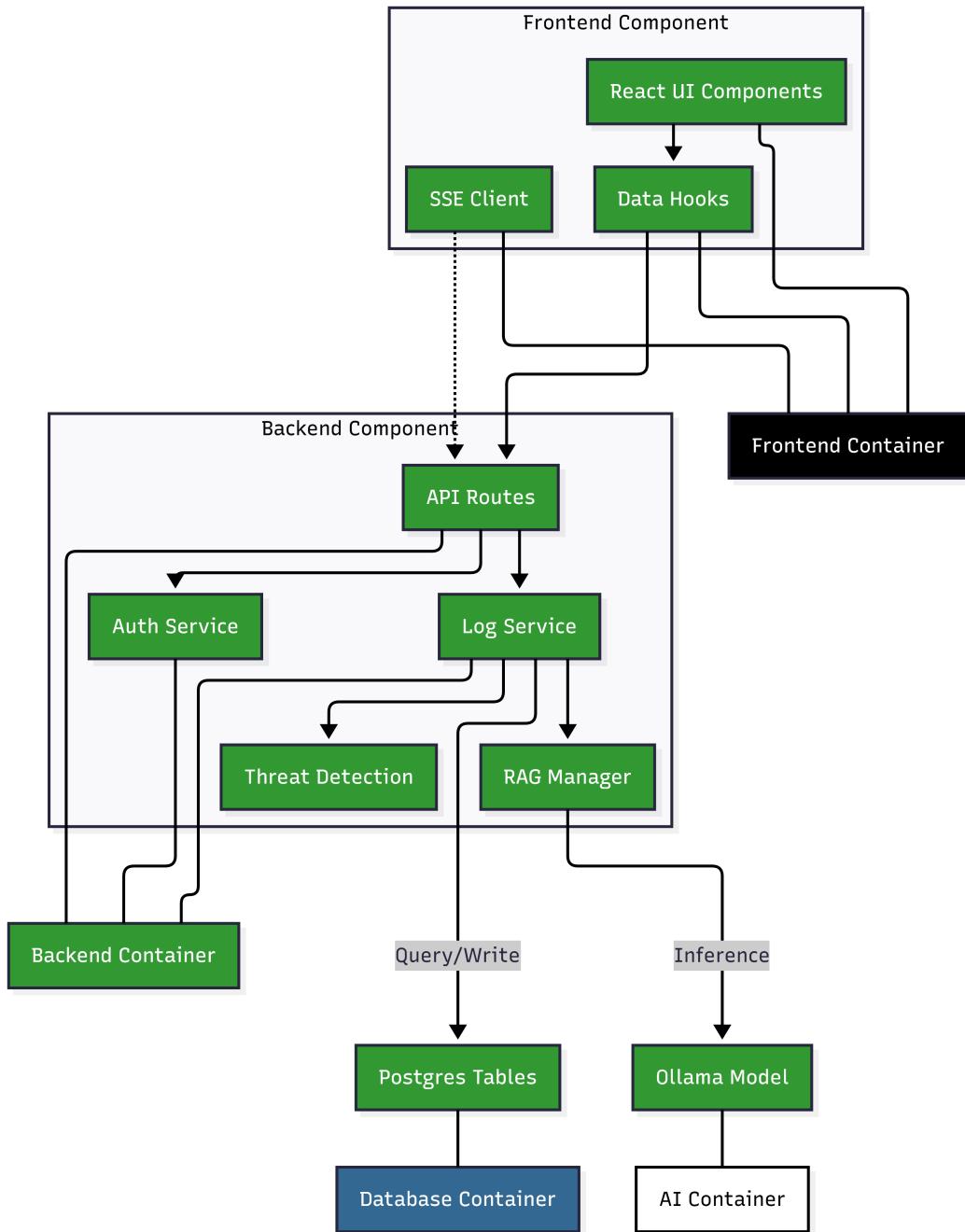


Figure 2.9: Component Architecture Diagram

2.8.3 Deployment Architecture

Figure ?? illustrates the production deployment topology.

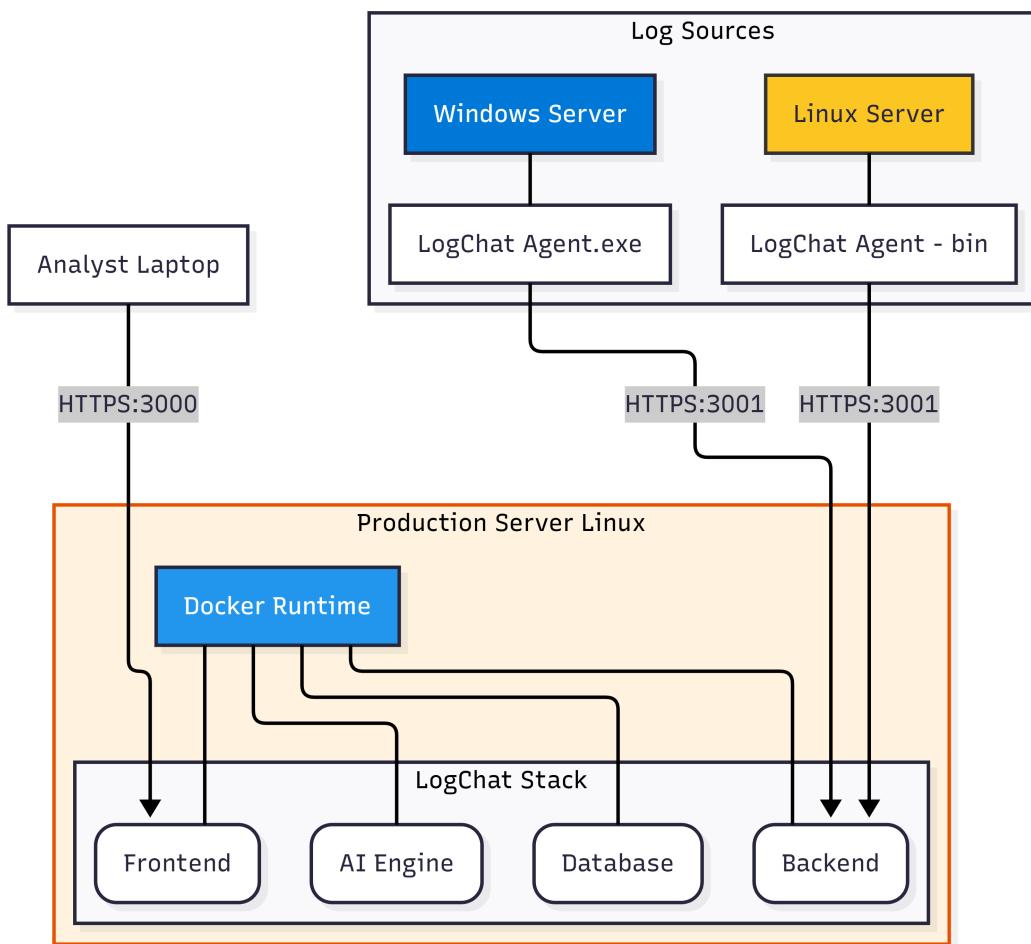


Figure 2.10: Deployment Architecture Diagram

2.8.4 Data Flow Diagram

Figure ?? presents the end-to-end data flow from log sources to end users.

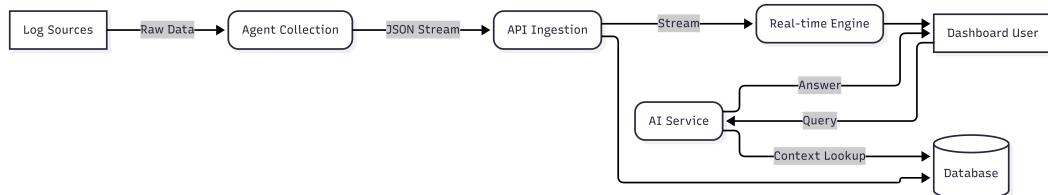


Figure 2.11: Data Flow Diagram (DFD Level 1)

2.9 Database Design

2.9.1 Physical Database Schema

Figure ?? presents the physical database schema with all tables and relationships.

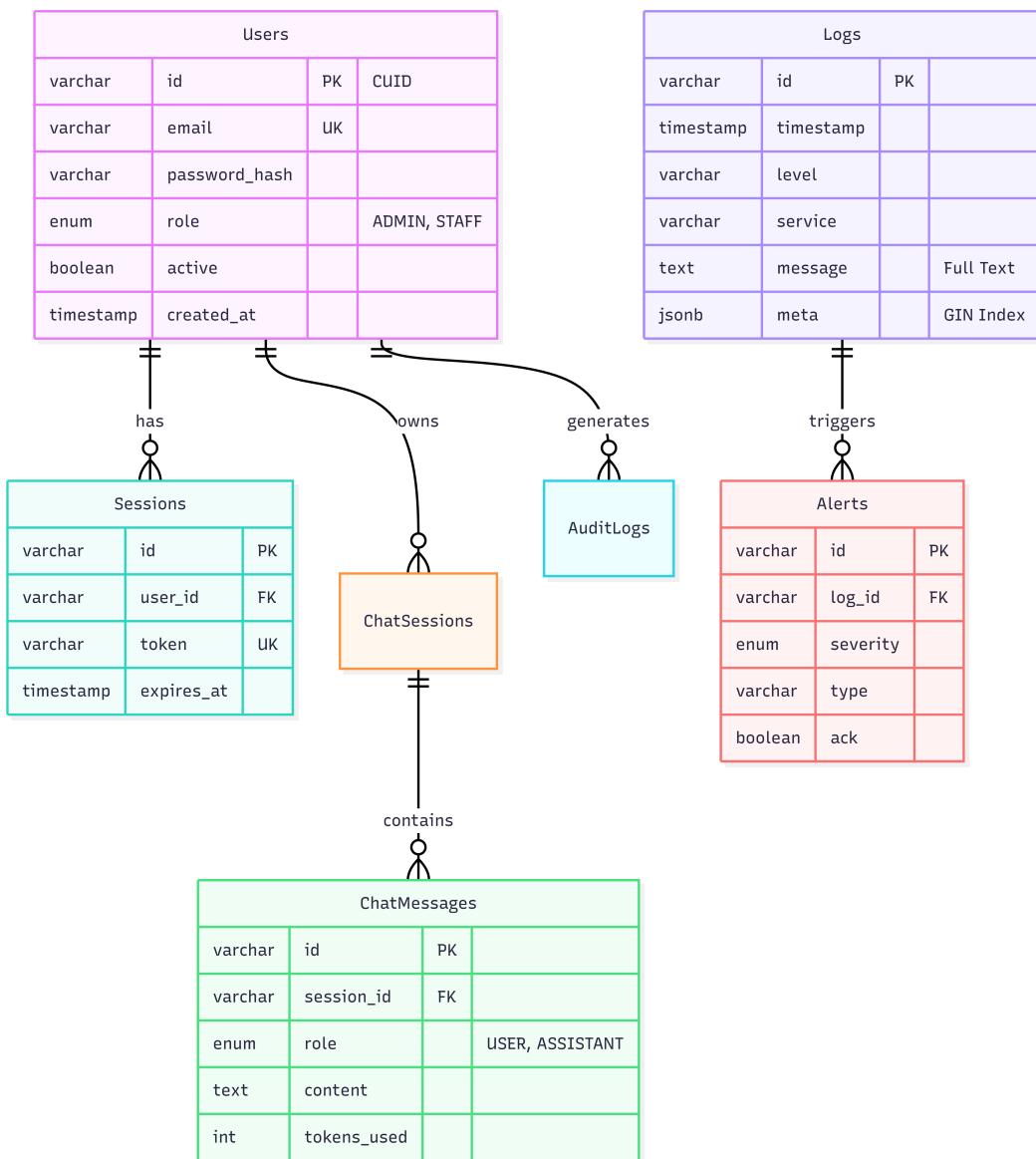


Figure 2.12: PostgreSQL Physical Database Schema

2.9.2 Indexing Strategy

Table 2.10: Database Indexing Strategy

Table	Index	Purpose
logs	idx_logs_timestamp	Time-range queries
logs	idx_logs_level	Filter by severity
logs	idx_logs_service	Filter by service
logs	idx_logs_composite	Combined filtering
users	idx_users_email	Unique login lookup
sessions	idx_sessions_token	Token validation
alerts	idx_alerts_status	Dashboard filtering

2.10 Security Architecture

2.10.1 Authentication Flow

Figure ?? illustrates the JWT-based authentication architecture.

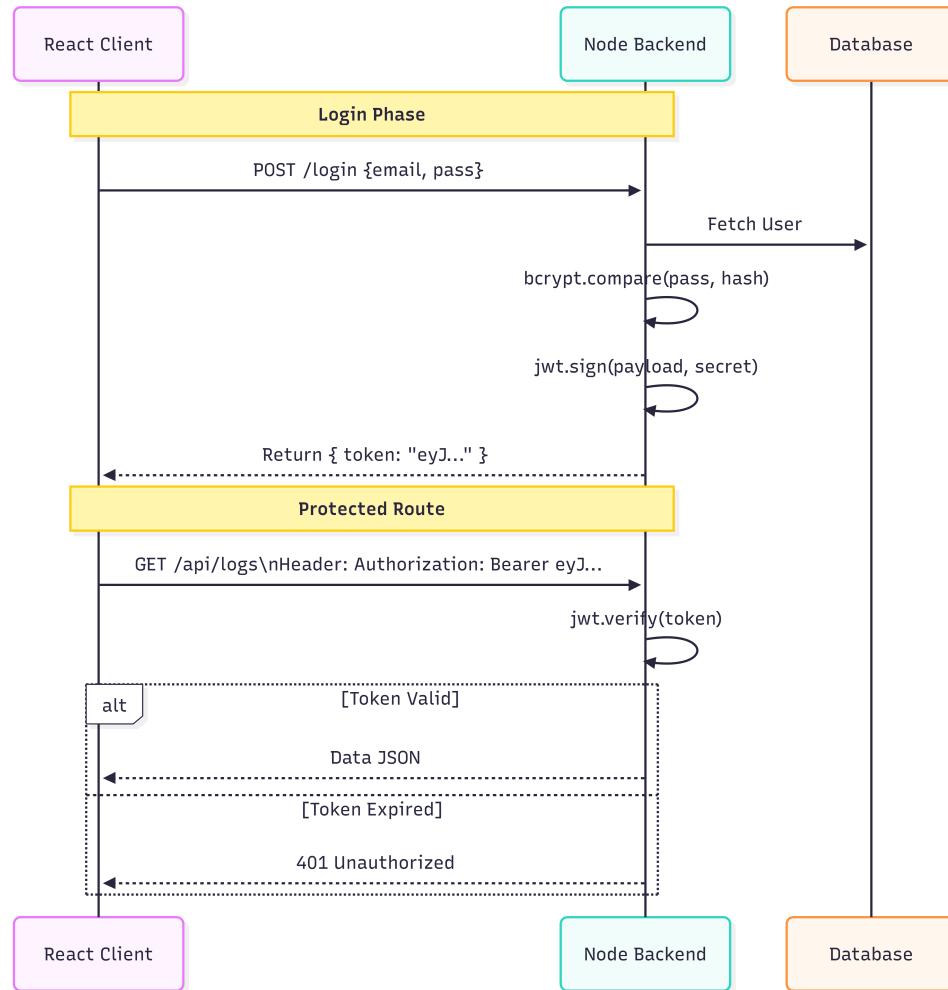


Figure 2.13: JWT Authentication Architecture

2.10.2 Role-Based Access Control

Table 2.11: RBAC Permission Matrix

Resource/Action	USER	STAFF	ADMIN
View Dashboard	✓	✓	✓
Query Logs	✓	✓	✓
Use AI Chat	✓	✓	✓
Export Data	✗	✓	✓
Manage Log Sources	✗	✓	✓
Manage Users	✗	✗	✓
View Audit Logs	✗	✗	✓

2.11 API Design

2.11.1 RESTful Endpoint Summary

Table 2.12: API Endpoint Overview

Method	Endpoint	Description	Auth
POST	/api/auth/login	Authenticate user	No
POST	/api/auth/register	Register new user	No
GET	/api/auth/me	Get current user	Yes
POST	/api/logs	Ingest single log	API Key
POST	/api/logs/ingest	Agent batch ingest	API Key
GET	/api/logs	Query logs	Yes
POST	/api/chat	Send chat message	Yes
GET	/api/stream/stats	SSE stats stream	Yes

2.12 Conclusion

This analysis and specification phase produced comprehensive documentation guiding the implementation: functional requirements covering 27 features across agent, backend, and frontend; 11 non-functional requirements ensuring quality attributes; UML artifacts modeling behavior and structure; and detailed architectural blueprints. The following chapter presents the implementation of these specifications.

Chapter 3

Implementation

3.1 Introduction

This chapter provides a comprehensive technical deep-dive into LogChat’s implementation. We examine the development of each component: the Golang Agent for log collection, the Node.js/Express backend for API services, the RAG-powered AI engine, and the Next.js frontend. Annotated code excerpts illustrate key design patterns and implementation decisions.

3.2 Development Environment

3.2.1 Hardware Configuration

Table 3.1: Development Environment Specifications

Component	Specification
Processor	Intel Core i7-12700H / AMD Ryzen 7 5800H
RAM	16GB DDR4
Storage	512GB NVMe SSD
Operating System	Windows 11 Pro / Ubuntu 22.04 LTS
GPU (AI)	NVIDIA RTX 3060 (6GB VRAM)

3.2.2 Software Tools

Table 3.2: Development Tools and Versions

Tool	Version	Purpose
VS Code	1.85+	Primary IDE
Go	1.21+	Agent development
Node.js	20 LTS	Backend/Frontend runtime
Docker Desktop	4.25+	Containerization
PostgreSQL	16	Database
Git	2.43+	Version control
Postman	10+	API testing

3.3 The Golang Agent

3.3.1 Project Architecture

Figure ?? presents the internal architecture of the Golang agent.

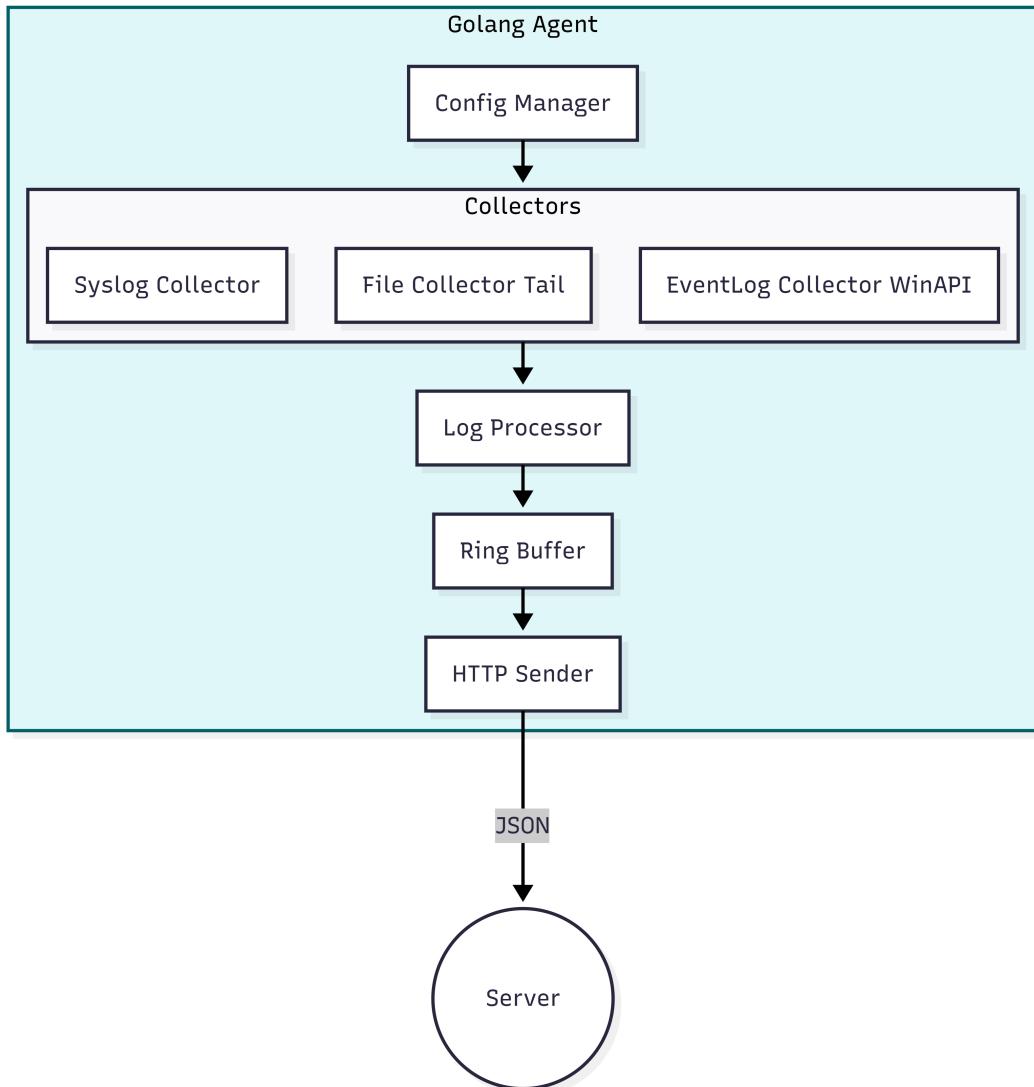


Figure 3.1: Golang Agent Internal Architecture

3.3.2 Directory Structure

```
1 golang-agent/
2     cmd/
3         --     agent/
4             --     main.go          # Entry point
5         internal/
6             collector/
7                 collector.go      # Interface definition
8                 file.go           # File tail collector
9                 eventlog_windows.go
10                journald_linux.go
11                syslog_linux.go
```

```

12           sender/
13           --      sender.go          # HTTP transmission
14           buffer/
15           --      buffer.go         # Ring buffer
16           --      config/
17           --      config.go          # YAML parsing
18           config.yaml            # Sample config
19           go.mod
20           go.sum
21 --      Makefile             # Build targets

```

Listing 3.1: Agent Project Structure

3.3.3 Collector Interface Pattern

The system employs the Strategy pattern for collectors, enabling polymorphic handling:

```

1 // internal/collector/collector.go
2 package collector
3
4 import (
5     "context"
6     "time"
7 )
8
9 // Collector defines the interface for all log collectors
10 type Collector interface {
11     // Name returns the unique identifier for this collector
12     Name() string
13
14     // Start begins log collection in a goroutine
15     Start(ctx context.Context)
16
17     // Stop gracefully terminates the collector
18     Stop()
19
20     // Stats returns current collection statistics
21     Stats() map[string]any
22 }

```

```
23
24 // BaseCollector provides common functionality
25 type BaseCollector struct {
26     name          string
27     sender        *sender.Sender
28     logsCollected int64
29     errorsCount   int64
30     lastCollected time.Time
31     running       bool
32     mu            sync.RWMutex
33 }
34
35 func (bc *BaseCollector) Name() string {
36     return bc.name
37 }
38
39 func (bc *BaseCollector) Stats() map[string]any {
40     bc.mu.RLock()
41     defer bc.mu.RUnlock()
42     return map[string]any{
43         "logs_collected": bc.logsCollected,
44         "errors_count":   bc.errorsCount,
45         "last_collected": bc.lastCollected,
46         "running":        bc.running,
47     }
48 }
```

Listing 3.2: Collector Interface Definition

Figure ?? shows the collector class hierarchy.

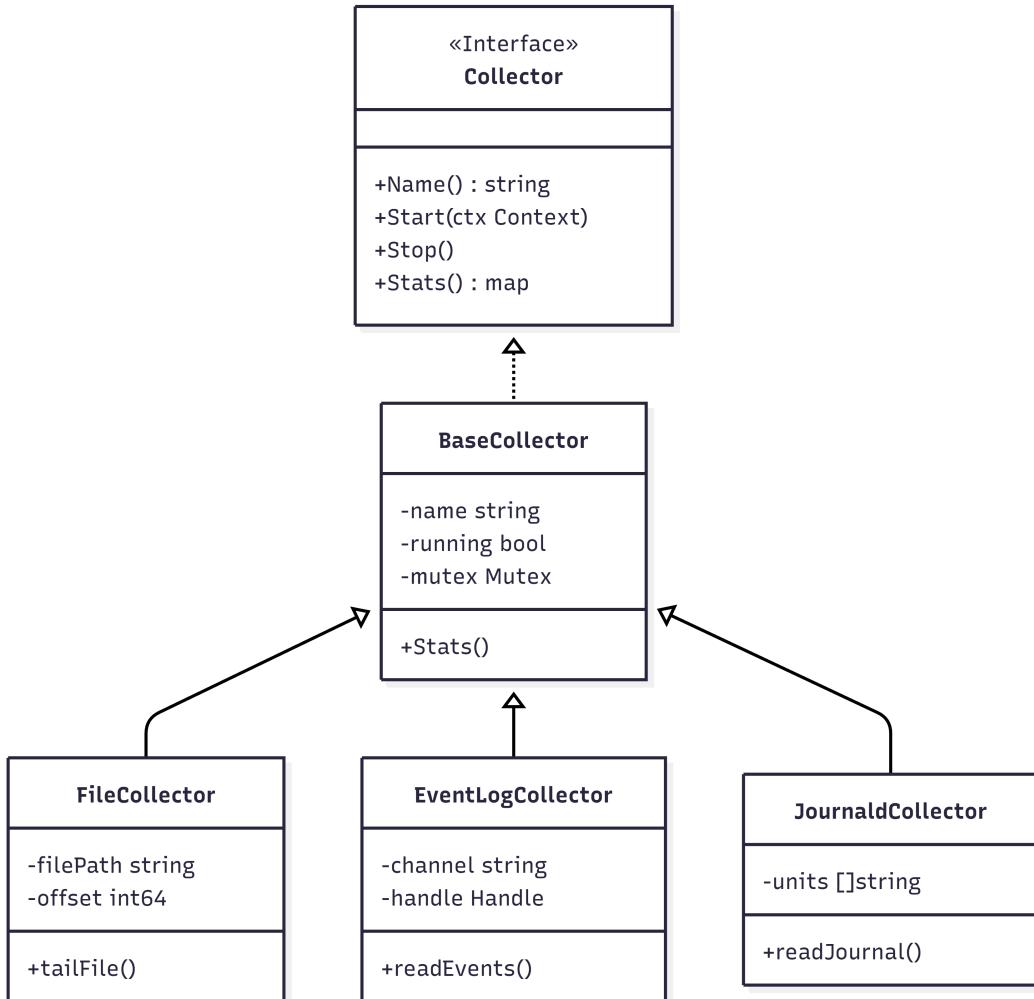


Figure 3.2: Collector Class Hierarchy

3.3.4 Windows Event Log Collection

The Windows collector utilizes direct Win32 API calls:

```

1 //go:build windows
2
3 package collector
4
5 import (
6     "golang.org/x/sys/windows"
7     "unsafe"
8 )
9
10 var (
11     advapi32      = windows.NewLazySystemDLL("advapi32.dll")
  
```

```

12     procOpenEventLogW = advapi32.NewProc("OpenEventLogW")
13     procReadEventLogW = advapi32.NewProc("ReadEventLogW")
14 )
15
16 func (ec *EventLogCollector) openEventLog(channel string) (
17     windows.Handle, error) {
18
19     channelPtr, _ := syscall.UTF16PtrFromString(channel)
20     ret, _, err := procOpenEventLogW.Call(
21         0, // Local computer
22         uintptr(unsafe.Pointer(channelPtr)),
23     )
24     if ret == 0 {
25         return 0, fmt.Errorf("OpenEventLog: %v", err)
26     }
27     return windows.Handle(ret), nil
28 }
29
30 func (ec *EventLogCollector) readEvents(
31     handle windows.Handle) ([]LogEntry, error) {
32
33     buffer := make([]byte, 64*1024) // 64KB buffer
34     var bytesRead, minBytes uint32
35
36     ret, _, _ := procReadEventLogW.Call(
37         uintptr(handle),
38         uintptr(EVENTLOG_SEQUENTIAL_READ | EVENTLOG_FORWARDS_READ),
39         0,
40         uintptr(unsafe.Pointer(&buffer[0])),
41         uintptr(len(buffer)),
42         uintptr(unsafe.Pointer(&bytesRead)),
43         uintptr(unsafe.Pointer(&minBytes)),
44     )
45     // Parse EVENTLOGRECORD structures...
46 }
```

Listing 3.3: Windows Event Log Collection

Figure ?? illustrates the Windows Event Log collection flow.

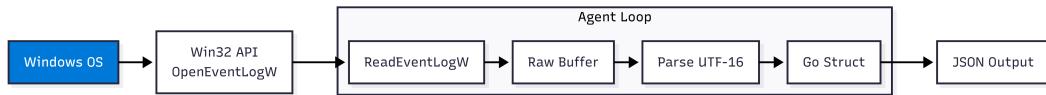


Figure 3.3: Windows Event Log Collection Flow

3.3.5 File Tailing Implementation

The file collector uses the nxadm/tail library for efficient following:

```

1 func (fc *FileCollector) tailFile(ctx context.Context,
2     filePath string) {
3
4     t, err := tail.TailFile(filePath, tail.Config{
5         Follow:      true,           // Keep following
6         ReOpen:      true,           // Handle rotation
7         MustExist:  true,
8         Location:   &tail.SeekInfo{
9             Offset: 0,
10            Whence: io.SeekEnd, // Start at end
11        },
12        Logger: tail.DiscardingLogger,
13    })
14    if err != nil {
15        fc.LogError(err)
16        return
17    }
18    defer t.Cleanup()
19
20    for {
21        select {
22            case <-ctx.Done():
23                return
24            case line := <-t.Lines:
25                if line.Err != nil {
26                    fc.LogError(line.Err)
27                    continue
28                }
29                entry := fc.processLine(filePath, line.Text)
30                fc.sender.Send(entry)

```

```

31         fc.incrementStats()
32     }
33 }
34 }
```

Listing 3.4: File Tailing with Rotation Support

3.3.6 Resilient Sender with Buffering

The Sender implements exponential backoff for network resilience:

```

1 func (s *Sender) flush(ctx context.Context) error {
2     entries := s.buffer.Peek(s.batchSize)
3     if len(entries) == 0 {
4         return nil
5     }
6
7     payload, _ := json.Marshal(IngestPayload{
8         Agent: s.agentInfo,
9         Logs:   entries,
10    })
11
12    var lastErr error
13    for attempt := 0; attempt < s.maxRetries; attempt++ {
14        req, _ := http.NewRequestWithContext(
15            ctx, "POST", s.serverURL, bytes.NewReader(payload))
16        req.Header.Set("Content-Type", "application/json")
17        req.Header.Set("X-API-Key", s.apiKey)
18
19        resp, err := s.client.Do(req)
20        if err == nil && resp.StatusCode == 201 {
21            s.buffer.Remove(len(entries))
22            s.sentCount += int64(len(entries))
23            return nil
24        }
25
26        lastErr = err
27        delay := time.Duration(1<<attempt) * 100 * time.Millisecond
28        time.Sleep(delay) // Exponential backoff
```

```

29 }

30

31     s.serverAlive = false
32
33     return lastErr
}

```

Listing 3.5: Sender with Retry Logic

Figure ?? shows the Sender state machine.

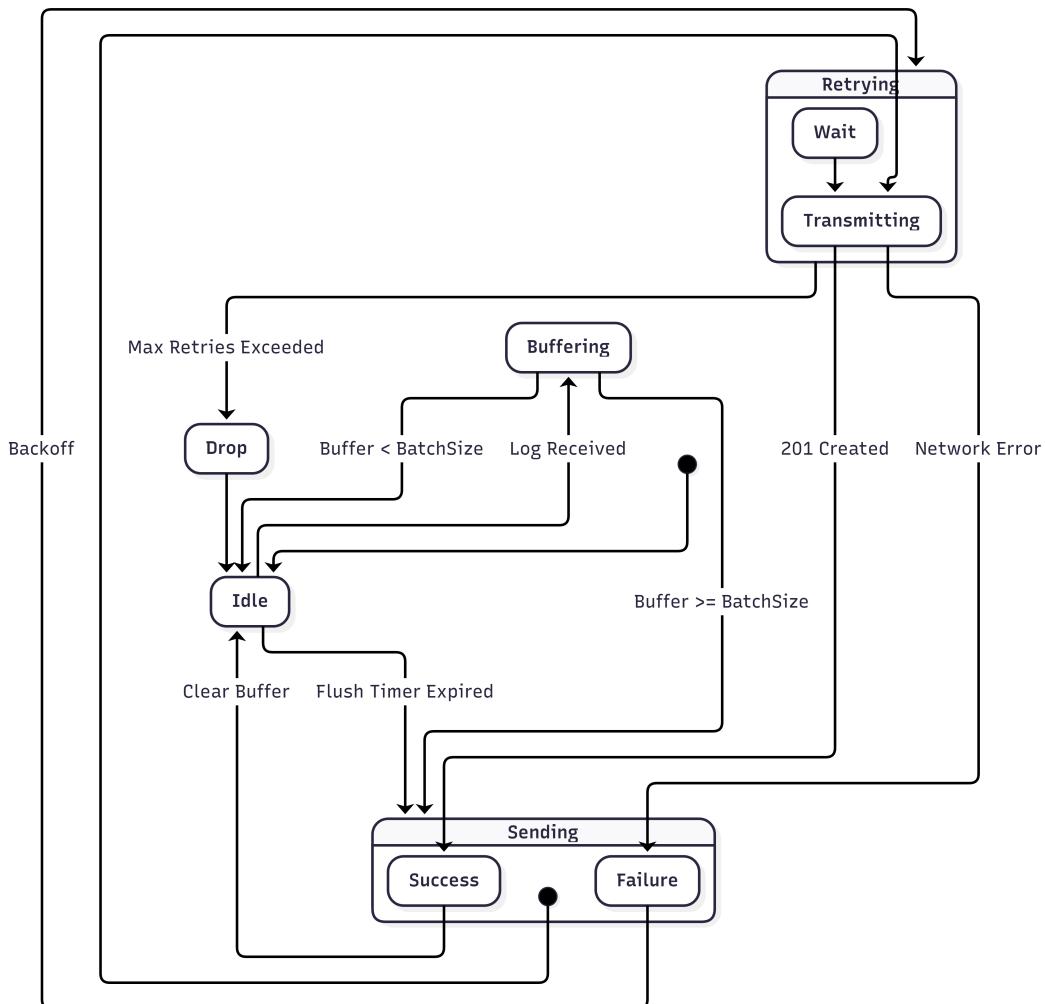


Figure 3.4: Sender State Machine

3.3.7 Cross-Compilation

```

1 BINARY := logchat-agent
2 VERSION := 1.0.0
3 LDFLAGS := -s -w -X main.Version=$(VERSION)

```

```

4
5 .PHONY: build-all
6 build-all: build-linux-amd64 build-linux-arm64 \
7             build-windows-amd64 build-darwin-amd64
8
9 build-linux-amd64:
10        GOOS=linux GOARCH=amd64 go build \
11                -ldflags="$(LD_FLAGS)" \
12                -o dist/$(BINARY)-linux-amd64 ./cmd/agent
13
14 build-windows-amd64:
15        GOOS=windows GOARCH=amd64 go build \
16                -ldflags="$(LD_FLAGS)" \
17                -o dist/$(BINARY)-windows-amd64.exe ./cmd/agent

```

Listing 3.6: Makefile for Cross-Compilation

3.4 Backend API Server

3.4.1 Project Structure

```

1 backend/
2         src/
3             index.ts          # Entry point
4             routes/
5                 auth.ts      # Authentication
6                 logs.ts      # Log ingestion/query
7                 chat.ts      # AI chat
8                 -- stream.ts   # SSE endpoints
9                 services/
10                auth.ts      # Auth business logic
11                logs.ts      # Log operations
12                chat.ts      # Chat session management
13                ollama.ts    # Ollama client
14                aiManager.ts # Multi-provider adapter
15                -- threatDetection.ts
16                -- middleware/
17                    auth.ts      # JWT validation

```

```

18      --          rateLimit.ts
19      prisma/
20      --          schema.prisma      # Database schema
21      package.json
22      tsconfig.json
23 --      Dockerfile

```

Listing 3.7: Backend Project Structure

3.4.2 Express Application Setup

```

1 // src/index.ts
2 import express from 'express';
3 import cors from 'cors';
4 import helmet from 'helmet';
5 import compression from 'compression';
6
7 import { authRouter } from './routes/auth.js';
8 import { logsRouter } from './routes/logs.js';
9 import { chatRouter } from './routes/chat.js';
10 import { streamRouter } from './routes/stream.js';
11 import { errorHandler } from './middleware/error.js';
12 import { requestLogger } from './middleware/logging.js';
13
14 const app = express();
15 const PORT = process.env.PORT || 3001;
16
17 // Security middleware
18 app.use(helmet());
19 app.use(cors({
20     origin: process.env.FRONTEND_URL || 'http://localhost:3000',
21     credentials: true,
22 }));
23 app.use(compression());
24 app.use(express.json({ limit: '10mb' }));
25
26 // Request logging
27 app.use(requestLogger);
28

```

```
29 // Health check
30 app.get('/health', (req, res) => {
31     res.json({ status: 'healthy', timestamp: new Date() });
32 });
33
34 // API Routes
35 app.use('/api/auth', authRouter);
36 app.use('/api/logs', logsRouter);
37 app.use('/api/chat', chatRouter);
38 app.use('/api/stream', streamRouter);
39
40 // Error handling
41 app.use(errorHandler);
42
43 // Start server
44 app.listen(PORT, () => {
45     console.log('LogChat Backend running on port ${PORT}');
46 }) ;
```

Listing 3.8: Main Application Entry Point

Figure ?? presents the backend component architecture.

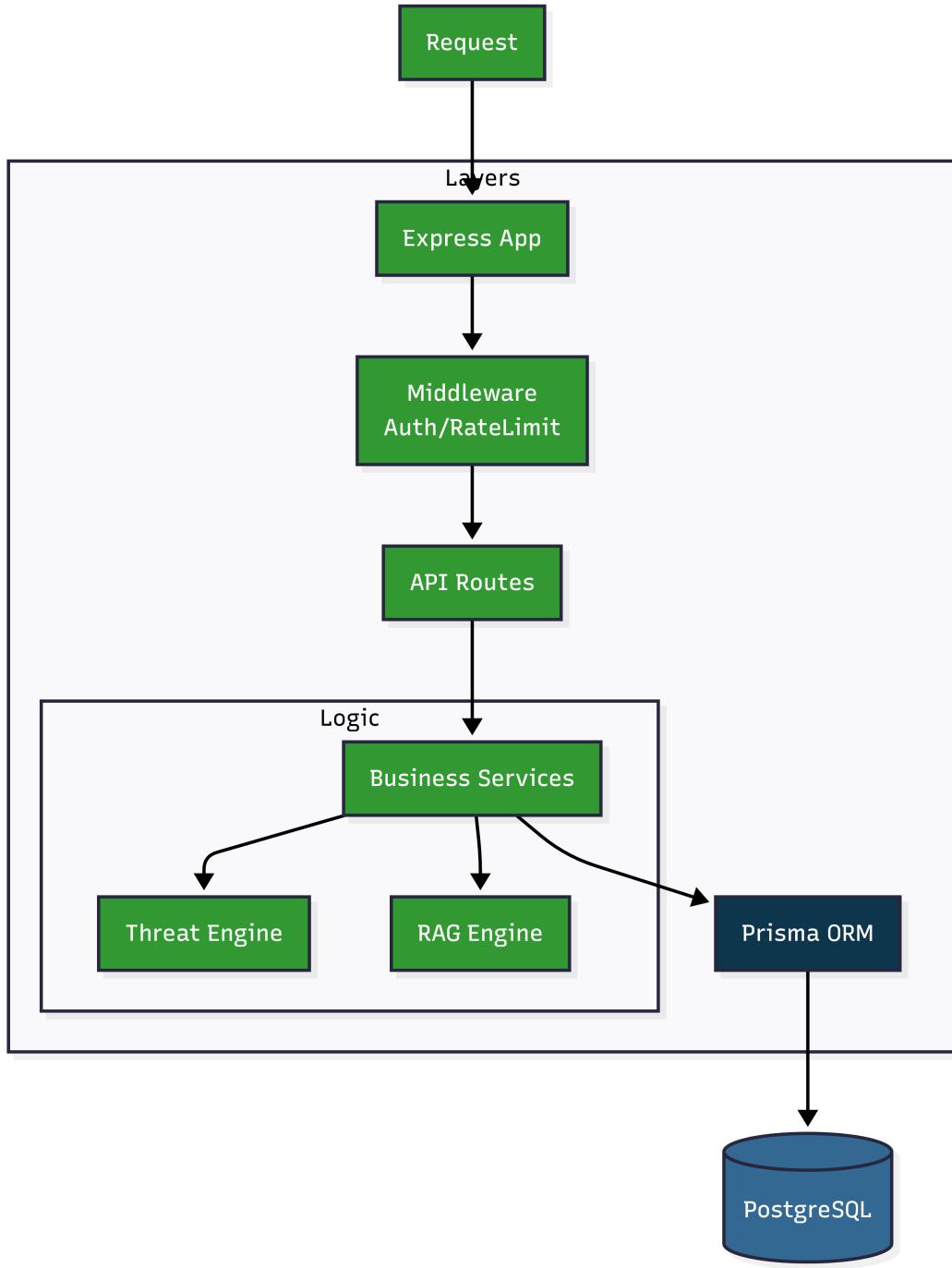


Figure 3.5: Backend Component Architecture

3.4.3 Authentication Service

```
1 // src/services/auth.ts
2 import bcrypt from 'bcryptjs';
3 import jwt from 'jsonwebtoken';
4 import { prisma } from '../lib/prisma.js';
5
```

```
6 | const JWT_SECRET = process.env.JWT_SECRET!;
7 | const JWT_EXPIRY = '7d';
8 | const BCRYPT_ROUNDS = 12;
9 |
10| export async function loginUser(
11|   email: string,
12|   password: string
13| ): Promise<AuthResult> {
14|   // Find user by email
15|   const user = await prisma.user.findUnique({
16|     where: { email }
17|   });
18|
19|   if (!user || !user.active) {
20|     return {
21|       success: false,
22|       error: 'Invalid credentials'
23|     };
24|   }
25|
26|   // Verify password with bcrypt
27|   const isValid = await bcrypt.compare(
28|     password,
29|     user.password
30|   );
31|   if (!isValid) {
32|     return {
33|       success: false,
34|       error: 'Invalid credentials'
35|     };
36|   }
37|
38|   // Generate JWT token
39|   const token = jwt.sign(
40|     {
41|       userId: user.id,
42|       role: user.role,
43|       email: user.email
44|     },
45|     JWT_SECRET,
46|     { expiresIn: JWT_EXPIRY }
47|   );
48| }
```

```

45     JWT_SECRET ,
46     { expiresIn: JWT_EXPIRY }
47 );
48
49 // Create session record for audit
50 await prisma.session.create({
51   data: {
52     token: hashToken(token),
53     userId: user.id,
54     expiresAt: new Date(Date.now() + 7*24*60*60*1000),
55   }
56 });
57
58 // Update last login timestamp
59 await prisma.user.update({
60   where: { id: user.id },
61   data: { lastLogin: new Date() }
62 });
63
64 return {
65   success: true,
66   user: sanitizeUser(user),
67   token
68 };
69 }

```

Listing 3.9: JWT Authentication Service

3.4.4 Threat Detection Engine

```

1 // src/services/threatDetection.ts
2
3 interface ThreatResult {
4   detected: boolean;
5   type?: string;
6   severity?: 'LOW' | 'MEDIUM' | 'HIGH' | 'CRITICAL';
7   description?: string;
8   mitre?: string;
9 }

```

```
10
11 const THREAT_PATTERNS = [
12   {
13     name: 'SQL_INJECTION',
14     pattern: /(’|--|;|OR\s+1\s*=|\s*1|UNION\s+SELECT)/i,
15     severity: 'CRITICAL' as const,
16     mitre: 'T1190',
17     description: 'Potential SQL injection attempt'
18   },
19   {
20     name: 'XSS_ATTEMPT',
21     pattern: /<script|javascript:|onerror\s*=?|onload\s*=?/i,
22     severity: 'HIGH' as const,
23     mitre: 'T1059.007',
24     description: 'Cross-site scripting attempt'
25   },
26   {
27     name: 'PATH_TRAVERSAL',
28     pattern: /\.\.\.\|(\.\.\.\|\etc\|passwd|boot\.ini)/i,
29     severity: 'CRITICAL' as const,
30     mitre: 'T1083',
31     description: 'Directory traversal attempt'
32   },
33 ];
34
35 export function analyzeThreat(log: LogEntry): ThreatResult {
36   const content = `${log.message} ${JSON.stringify(log.meta)}`;
37
38   for (const threat of THREAT_PATTERNS) {
39     if (threat.pattern.test(content)) {
40       return {
41         detected: true,
42         type: threat.name,
43         severity: threat.severity,
44         description: threat.description,
45         mitre: threat.mitre,
46       };
47     }
48   }
}
```

```

49
50     return { detected: false };
51 }

```

Listing 3.10: Pattern-Based Threat Detection

Figure ?? presents the threat pattern categories.

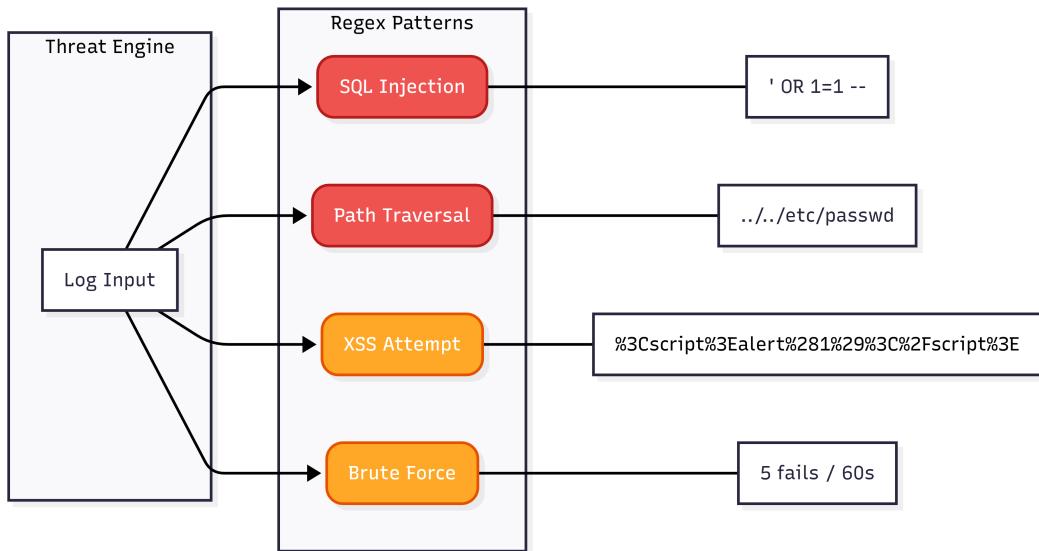


Figure 3.6: Threat Detection Pattern Categories

3.4.5 RAG Chat Implementation

```

1 // src/routes/chat.ts
2
3 router.post('/chat', authMiddleware, async (req, res) => {
4     const { message, sessionId, filters } = req.body;
5     const userId = req.user.id;
6
7     // 1. RETRIEVAL: Get relevant logs for context
8     const logs = await queryLogs({
9         level: ['ERROR', 'WARN'],
10        startTime: subHours(new Date(), filters?.hours || 1),
11        limit: 100,
12        search: extractKeywords(message),
13    });
14
15     // 2. Get aggregate statistics

```

```
16  const stats = await getStatsForLLM(60);
17
18  // 3. Build context-enriched prompt
19  const context = buildLogContext(logs, stats);
20  const systemPrompt = 'You are LogChat, an AI security analyst.
21 Analyze the following log data and answer the user's question.
22 Be specific, cite log entries, and suggest remediation steps.
23
24 LOG CONTEXT:
25 ${context}
26
27 STATISTICS:
28 - Total logs analyzed: ${stats.totalLogs}
29 - Error rate: ${stats.errorRate}%
30 - Top error services: ${stats.topServices.join(',')}
31 ';
32
33 // 4. GENERATION: Call LLM
34 const startTime = Date.now();
35 const response = await aiManager.chat({
36   messages: [
37     { role: 'system', content: systemPrompt },
38     { role: 'user', content: message }
39   ],
40   provider: 'ollama',
41   model: 'qwen2.5:0.5b',
42 });
43 const responseTime = Date.now() - startTime;
44
45 // 5. Persist chat message
46 await prisma.chatMessage.create({
47   data: {
48     sessionId,
49     role: 'assistant',
50     content: response.content,
51     responseTime,
52     tokensUsed: response.usage?.totalTokens,
53     provider: response.provider,
54   }
55 })
```

```

55 });
56
57 res.json({
58   success: true,
59   response: response.content,
60   metadata: {
61     logsAnalyzed: logs.length,
62     responseTime,
63     provider: response.provider,
64   }
65 });
66 });

```

Listing 3.11: RAG-Powered Chat Route

Figure ?? illustrates the RAG workflow in detail.

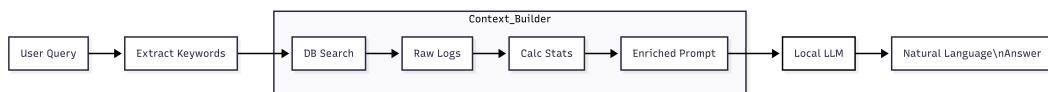


Figure 3.7: RAG Workflow Detailed Diagram

3.4.6 Server-Sent Events Implementation

```

1 // src/routes/stream.ts
2
3 const clients = new Map<number, Response>();
4
5 router.get('/stats', authMiddleware, (req, res) => {
6   // Set SSE headers
7   res.setHeader('Content-Type', 'text/event-stream');
8   res.setHeader('Cache-Control', 'no-cache');
9   res.setHeader('Connection', 'keep-alive');
10  res.setHeader('X-Accel-Buffering', 'no');
11
12  const clientId = Date.now();
13  clients.set(clientId, res);
14
15  // Send initial connection event
16  res.write(`event:connected\ndata:${JSON.stringify({

```

```

17    id: clientId,
18    timestamp: new Date(),
19  })}\n\n';
20
21  // Periodic stats broadcast
22  const interval = setInterval(async () => {
23    const stats = await getRealtimeStats();
24    res.write(`event:stats\ndata:${JSON.stringify(stats)}\n\n`);
25  }, 5000);
26
27  // Cleanup on disconnect
28  req.on('close', () => {
29    clearInterval(interval);
30    clients.delete(clientId);
31  });
32});
33
34 // Broadcast function for log events
35 export function broadcastLog(log: LogEntry) {
36  const data = JSON.stringify({ event: 'log', log });
37  clients.forEach((client) => {
38    client.write(`event:log\ndata:${data}\n\n`);
39  });
40}

```

Listing 3.12: SSE Stream for Real-time Updates

3.5 Frontend Dashboard

3.5.1 Project Structure

```

1 frontend/
2   app/
3     layout.tsx      # Root layout
4     page.tsx        # Landing page
5     dashboard/
6       -- page.tsx    # Main dashboard
7       chat/

```

```

8      --          page.tsx           # AI chat interface
9          login/
10         --          page.tsx        # Authentication
11         --          admin/
12             --          page.tsx      # Admin panel
13             components/
14                 Navbar.tsx
15                 StatsCards.tsx
16                 LogsChart.tsx
17                 LogTable.tsx
18             ChatInterface.tsx
19             lib/
20                 api.ts
21                 auth.tsx           # Auth context
22             --          utils.ts
23 --          styles/
24             --          globals.css

```

Listing 3.13: Frontend Project Structure

3.5.2 Real-time Dashboard with SSE

```

1 // app/dashboard/page.tsx
2 'useClient';
3
4 export default function DashboardPage() {
5     const [stats, setStats] = useState<DashboardStats | null>(null);
6     const [logs, setLogs] = useState<LogEntry[]>([]);
7     const [connected, setConnected] = useState(false);
8
9     // SSE connection
10    useEffect(() => {
11        const eventSource = new EventSource(
12            `${API_URL}/api/stream/stats`,
13            { withCredentials: true }
14        );
15
16        eventSource.addEventListener('connected', () => {
17            setConnected(true);

```

```

18     });
19
20     eventSource.addEventListener('stats', (event) => {
21       const data = JSON.parse(event.data);
22       setStats(data);
23     });
24
25     eventSource.addEventListener('log', (event) => {
26       const { log } = JSON.parse(event.data);
27       setLogs(prev => [log, ...prev.slice(0, 99)]);
28
29       // Toast notification for errors
30       if (log.level === 'ERROR') {
31         toast.error(log.message.slice(0, 100));
32       }
33     });
34
35     return () => eventSource.close();
36   }, []);
37
38   return (
39     <div className="dashboard-container">
40       <LiveIndicator connected={connected} />
41       <StatsCards stats={stats} />
42       <div className="charts-row">
43         <LogsChart data={stats?.timeline} />
44         <ServicesChart data={stats?.topServices} />
45       </div>
46       <LogTable logs={logs} />
47     </div>
48   );
49 }

```

Listing 3.14: Dashboard SSE Integration

3.5.3 Stats Cards Component

```

1 // components/StatsCards.tsx
2

```

```
3 interface StatsCardsProps {
4     stats: DashboardStats | null;
5 }
6
7 export function StatsCards({ stats }: StatsCardsProps) {
8     const cards = [
9         {
10             title: 'Total Logs',
11             value: stats?.totalLogs ?? 0,
12             change: stats?.logsChange ?? 0,
13             icon: Database,
14             color: 'blue',
15         },
16         {
17             title: 'Errors',
18             value: stats?.errorCount ?? 0,
19             change: stats?.errorsChange ?? 0,
20             icon: AlertTriangle,
21             color: 'red',
22         },
23         {
24             title: 'Warnings',
25             value: stats?.warnCount ?? 0,
26             change: stats?.warningsChange ?? 0,
27             icon: AlertCircle,
28             color: 'yellow',
29         },
30         {
31             title: 'Threats',
32             value: stats?.threatCount ?? 0,
33             change: stats?.threatsChange ?? 0,
34             icon: Shield,
35             color: 'purple',
36         },
37     ];
38
39     return (
40         <div className="stats-grid">
41             {cards.map((card) => (
```

```
42         <div key={card.title}>
43             className={'stat-card stat-$ {card.color}'}
44             <card.icon className="stat-icon" />
45             <div className="stat-content">
46                 <span className="stat-value">
47                     {formatNumber(card.value)}
48                 </span>
49                 <span className="stat-title">{card.title}</
50                 span>
51                 <ChangeIndicator value={card.change} />
52             </div>
53         )})
54     </div>
55 );
56 }
```

Listing 3.15: Stats Cards Component

Figure ?? shows an actual screenshot of the dashboard.

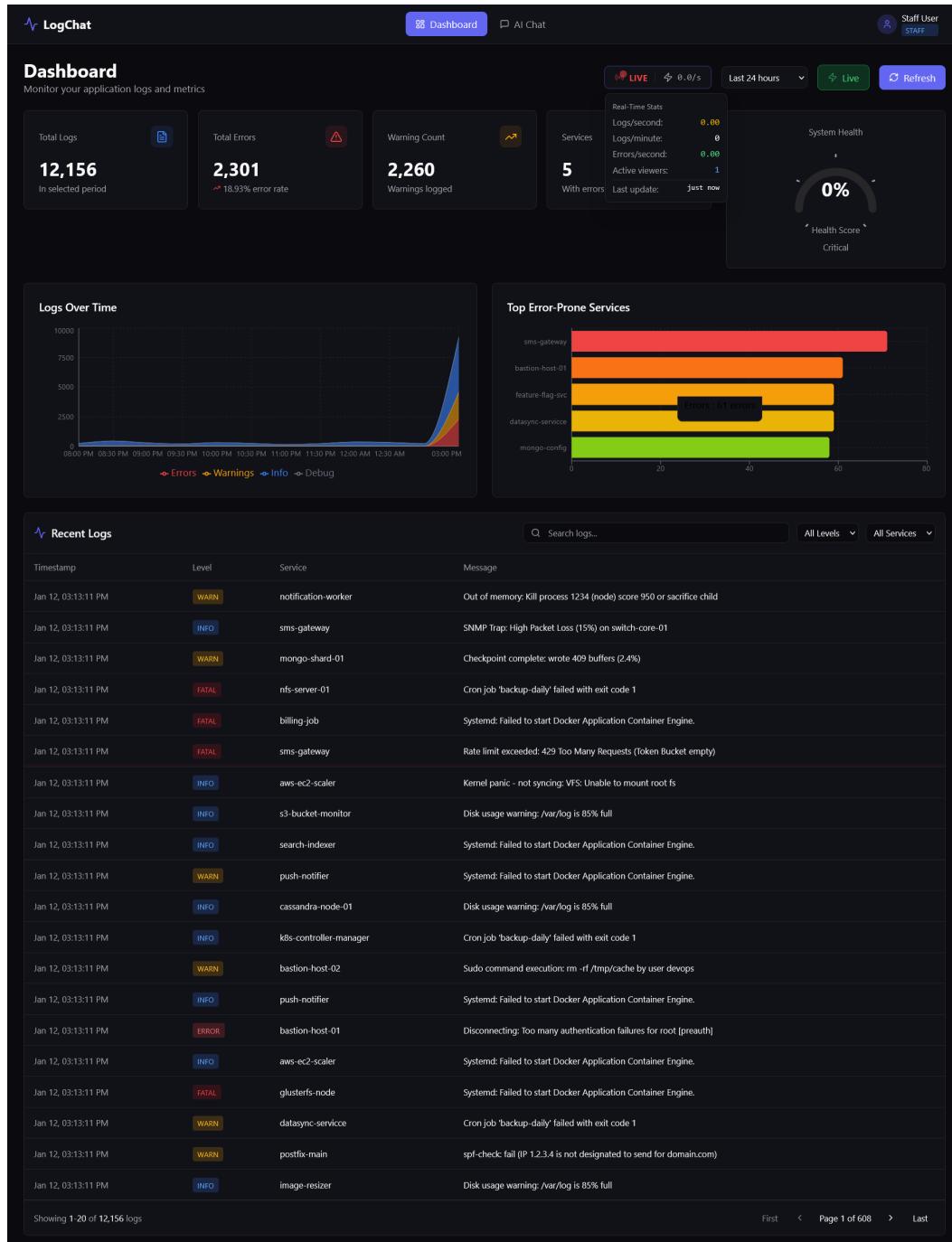


Figure 3.8: Dashboard Screenshot

3.5.4 AI Chat Interface

Figure ?? presents the AI chat interface screenshot.

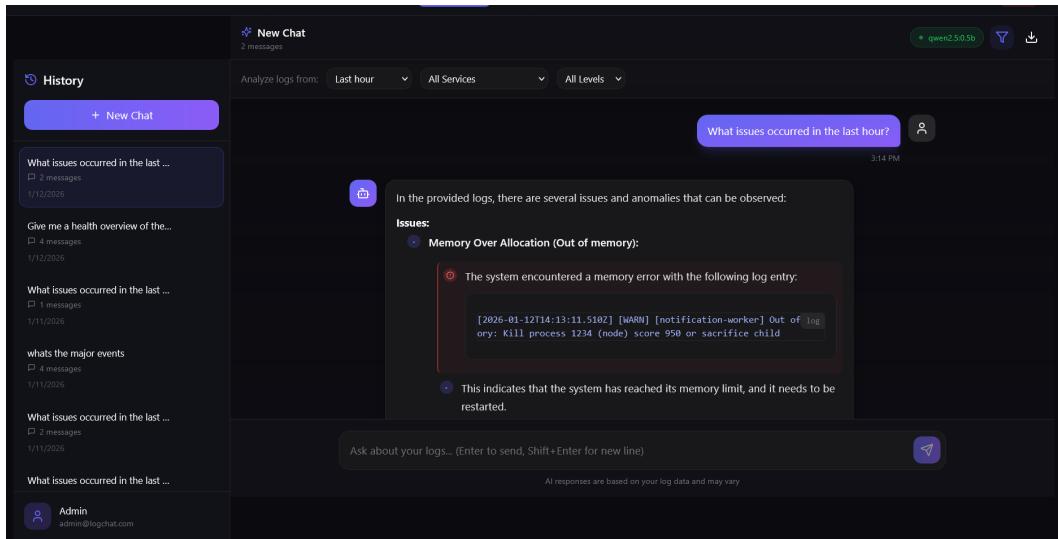


Figure 3.9: AI Chat Interface Screenshot

3.6 Prisma Database Schema

```

1 // prisma/schema.prisma
2
3 model User {
4   id      String    @id @default(cuid())
5   email   String    @unique
6   password String
7   name    String
8   role    Role      @default(USER)
9   active   Boolean  @default(true)
10  createdAt DateTime @default(now())
11  lastLogin DateTime?
12
13  sessions     Session []
14  chatSessions ChatSession []
15  auditLogs    AuditLog []
16 }
17
18 model Log {
19   id      String    @id @default(cuid())
20   timestamp DateTime
21   level   String
22   service  String

```

```

23   message      String      @db.Text
24   raw          String      @db.Text
25   meta         Json?
26   sourceId    String?
27   createdAt   DateTime   @default(now())
28
29   @@index([timestamp])
30   @@index([level])
31   @@index([service])
32 }
33
34 model Alert {
35   id           String      @id @default(cuid())
36   logId        String?
37   severity     AlertSeverity
38   type         String
39   message      String
40   status       AlertStatus  @default(NEW)
41   isAcknowledged Boolean   @default(false)
42   createdAt   DateTime   @default(now())
43
44   @@index([status])
45   @@index([createdAt])
46 }
```

Listing 3.16: Prisma Schema (Excerpt)

3.7 Conclusion

This implementation chapter demonstrated the translation of specifications into working software. Key technical achievements include: a cross-platform Golang agent with native Windows Event Log support; a Node.js backend with real-time SSE streaming and RAG-powered AI chat; and a responsive React dashboard with live updates. The following chapter presents testing and deployment strategies.

Chapter 4

Testing and Deployment

4.1 Introduction

This chapter outlines the quality assurance strategy employed during LogChat development and describes the containerized deployment architecture. We present unit testing, integration testing, performance benchmarks, and the Docker-based deployment configuration.

4.2 Testing Strategy Overview

Figure ?? illustrates the testing pyramid employed for LogChat quality assurance.

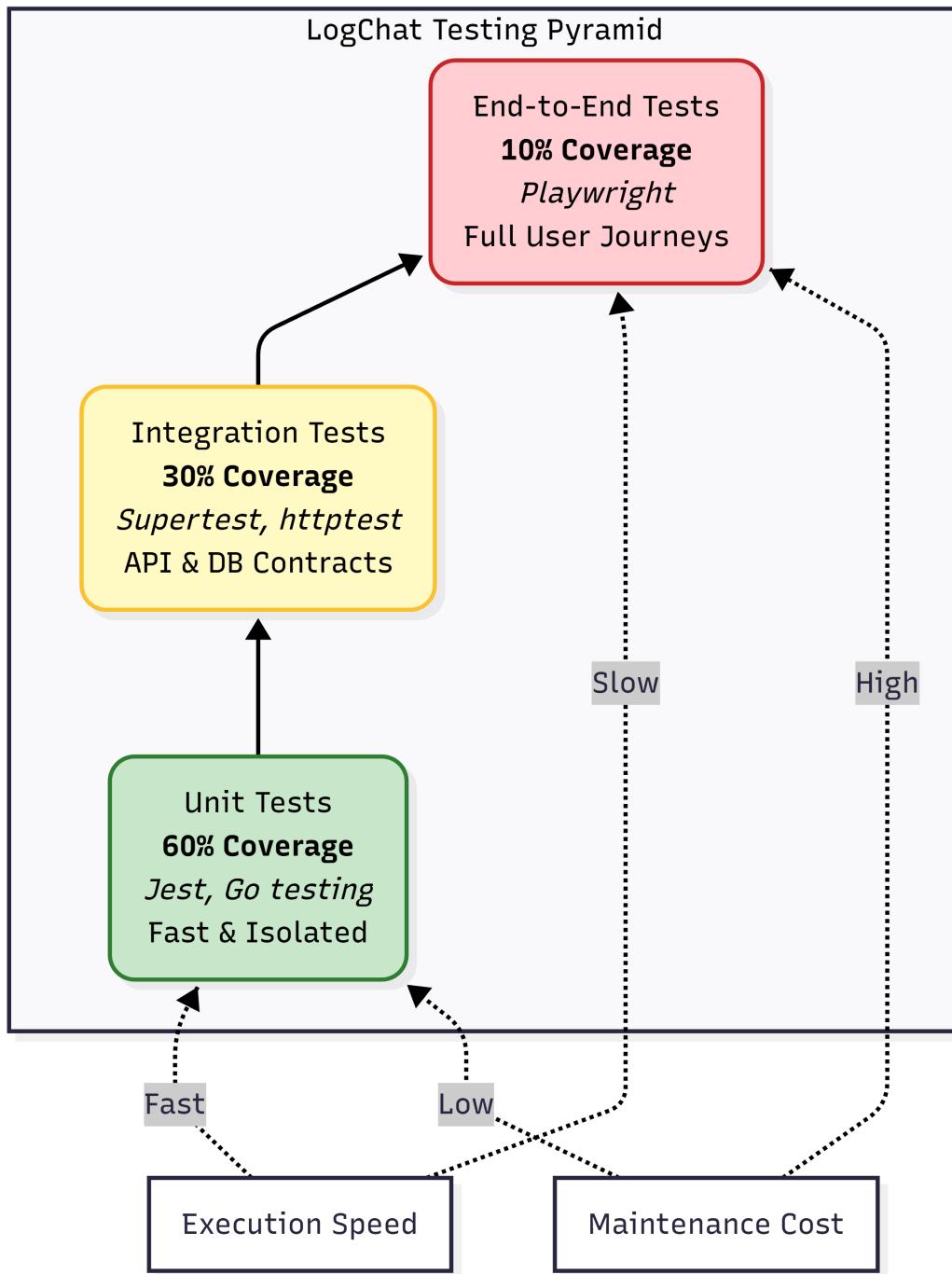


Figure 4.1: Testing Pyramid Strategy

Table 4.1: Testing Strategy by Component

Component	Test Type	Framework	Coverage
Golang Agent	Unit	Go testing pkg	75%
Golang Agent	Integration	Go testing + httptest	60%
Backend API	Unit	Jest + ts-jest	80%
Backend API	Integration	Supertest	70%
Frontend	Component	React Testing Library	65%
Full Stack	E2E	Playwright	40%

4.3 Unit Testing

4.3.1 Threat Detection Engine Tests

Table 4.2: Threat Detection Test Cases

Input	Expected	Result
"' OR 1=1 -"	SQL_INJECTION, CRITICAL	✓PASS
"<script>alert(1)</script>"	XSS_ATTEMPT, HIGH	✓PASS
"../../etc/passwd"	PATH_TRAVERSAL, CRITICAL	✓PASS
"User logged in"	No threat (null)	✓PASS
5 failed logins in 60s	BRUTE_FORCE, HIGH	✓PASS
"DROP TABLE users;"	SQL_INJECTION, CRITICAL	✓PASS

```

1 // src/services/__tests__/_threatDetection.test.ts
2
3 describe('ThreatDetectionEngine', () => {
4     describe('SQL Injection Detection', () => {
5         test('detects OR 1=1 pattern', () => {

```

```

6      const log = createLog(" OR 1=1--");
7      const result = analyzeThreat(log);
8
9      expect(result.detected).toBe(true);
10     expect(result.type).toBe('SQL_INJECTION');
11     expect(result.severity).toBe('CRITICAL');
12   });
13
14 test('detects UNION SELECT', () => {
15   const log = createLog("UNION SELECT * FROM users");
16   const result = analyzeThreat(log);
17
18   expect(result.detected).toBe(true);
19   expect(result.mitre).toBe('T1190');
20 });
21
22
23 describe('XSS Detection', () => {
24   test('detects script tag', () => {
25     const log = createLog("<script>alert('xss')</script>");
26     const result = analyzeThreat(log);
27
28     expect(result.detected).toBe(true);
29     expect(result.type).toBe('XSS_ATTEMPT');
30     expect(result.severity).toBe('HIGH');
31   });
32 });
33
34 describe('False Positive Prevention', () => {
35   test('ignores normal log messages', () => {
36     const log = createLog("User successfully logged in");
37     const result = analyzeThreat(log);
38
39     expect(result.detected).toBe(false);
40   });
41 });
42 });

```

Listing 4.1: Threat Detection Unit Test

Figure ?? shows the test coverage report.

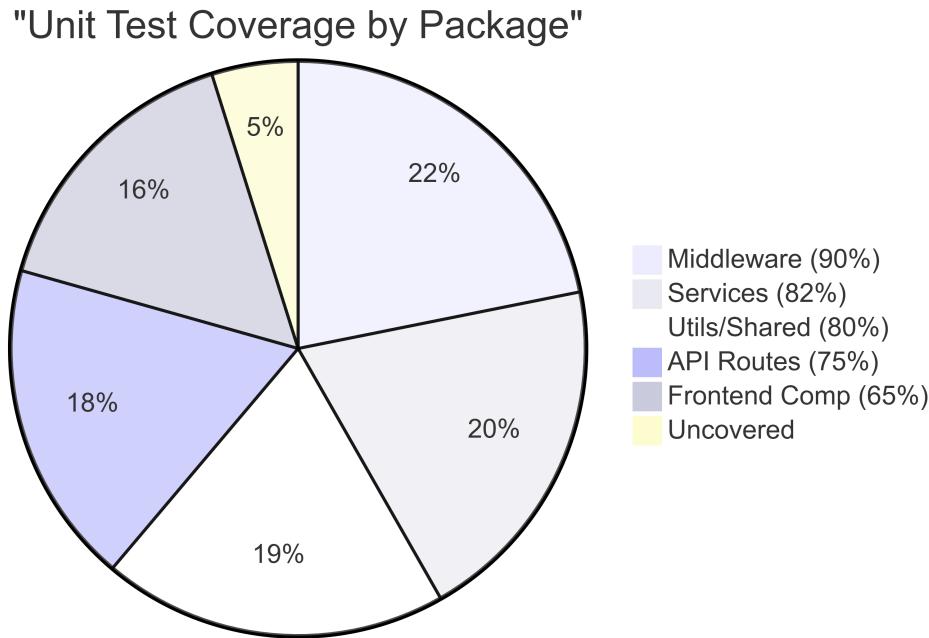


Figure 4.2: Test Coverage Report

4.3.2 Authentication Service Tests

```
1 describe('AuthService', () => {
2     describe('loginUser', () => {
3         test('succeeds with valid credentials', async () => {
4             const result = await loginUser(
5                 'admin@logchat.io',
6                 'password123'
7             );
8
9             expect(result.success).toBe(true);
10            expect(result.token).toBeDefined();
11            expect(result.user.role).toBe('ADMIN');
12        });
13
14        test('fails with wrong password', async () => {
15            const result = await loginUser(
16                'admin@logchat.io',
17                'wrongpassword'
18            );
19        });
20    });
21});
```

```

19
20     expect(result.success).toBe(false);
21     expect(result.error).toBe('Invalid credentials');
22   );
23
24   test('fails for disabled account', async () => {
25     const result = await loginUser(
26       'disabled@test.com',
27       'password123'
28     );
29
30     expect(result.success).toBe(false);
31   );
32 );
33
34 describe('password hashing', () => {
35   test('uses bcrypt with cost factor 12', async () => {
36     const hash = await hashPassword('test123');
37
38     expect(hash).toMatch(/^\$2[aby]?\$12\$/);
39   );
40 });
41 });

```

Listing 4.2: Authentication Unit Tests

4.4 Integration Testing

4.4.1 API Endpoint Testing

```

1 // tests/integration/api.test.ts
2
3 describe('Logs API', () => {
4   let authToken: string;
5
6   beforeAll(async () => {
7     const res = await request(app)
8       .post('/api/auth/login')

```

```
9      .send({
10        email: 'test@logchat.io',
11        password: 'test123'
12      });
13      authToken = res.body.token;
14    });
15
16  describe('POST /api/logs/ingest', () => {
17    test('ingests batch of logs', async () => {
18      const response = await request(app)
19        .post('/api/logs/ingest')
20        .set('X-API-Key', 'test-api-key')
21        .send({
22          agent: { hostname: 'test-server' },
23          logs: [
24            { level: 'INFO', message: 'Test log 1' },
25            { level: 'ERROR', message: 'Test error' },
26          ]
27        });
28
29      expect(response.status).toBe(201);
30      expect(response.body.ingested).toBe(2);
31    });
32
33    test('rejects invalid API key', async () => {
34      const response = await request(app)
35        .post('/api/logs/ingest')
36        .set('X-API-Key', 'invalid-key')
37        .send({ logs: [] });
38
39      expect(response.status).toBe(401);
40    });
41  });
42
43  describe('GET /api/logs', () => {
44    test('returns filtered logs', async () => {
45      const response = await request(app)
46        .get('/api/logs')
47        .set('Authorization', `Bearer ${authToken}`)
```

```

48         .query({ level: 'ERROR', limit: 10 });
49
50     expect(response.status).toBe(200);
51     expect(Array.isArray(response.body.logs)).toBe(true);
52     response.body.logs.forEach((log: any) => {
53       expect(log.level).toBe('ERROR');
54     });
55   });
56 });
57 });

```

Listing 4.3: API Integration Tests

4.4.2 SSE Stream Testing

```

1 describe('SSEStream', () => {
2   test('receives real-time log events', (done) => {
3     const eventSource = new EventSource(
4       'http://localhost:3001/api/stream/stats',
5     );
6
7     eventSource.addEventListener('connected', () => {
8       // Trigger a log event
9       ingestTestLog({ level: 'ERROR', message: 'Test' });
10    });
11
12    eventSource.addEventListener('log', (event) => {
13      const data = JSON.parse(event.data);
14      expect(data.log.level).toBe('ERROR');
15      eventSource.close();
16      done();
17    });
18
19    setTimeout(() => {
20      eventSource.close();
21      done(new Error('Timeout waiting for SSE event'));
22    }, 5000);
23  });
24 });

```

Listing 4.4: SSE Integration Test

4.5 Performance Testing

4.5.1 Load Testing with Apache Benchmark

```
1 # Log ingestion endpoint
2 $ ab -n 10000 -c 100 -p payload.json -T application/json \
3   -H "X-API-Key: test-key" \
4   http://localhost:3001/api/logs/ingest
5
6 Document Path:          /api/logs/ingest
7 Document Length:        45 bytes
8
9 Concurrency Level:      100
10 Time taken for tests:  8.017 seconds
11 Complete requests:    10000
12 Failed requests:      0
13
14 Requests per second:  1247.32 [#/sec] (mean)
15 Time per request:     80.17 [ms] (mean)
16 Time per request:     0.802 [ms] (across all)
17 Transfer rate:        58.29 [Kbytes/sec] received
18
19 Connection Times (ms)
20             min   mean [+/- sd] median   max
21 Connect:        0     1   0.5       1      5
22 Processing:    10    79  12.3      78    145
23 Waiting:       9    78  12.2      77    144
24 Total:         11    80  12.3      79    147
25
26 Percentage of the requests served within a certain time (ms)
27   50%    79
28   66%    84
29   75%    87
30   90%    95
```

```

31 95%      102
32 99%      118
33 100%     147 (longest request)

```

Listing 4.5: Load Test Commands and Results

4.5.2 Performance Benchmarks Summary

Table 4.3: Performance Benchmark Results

Metric	Target	Achieved	Status
Log ingestion rate	1000 req/s	1247 req/s	✓
P50 latency (ingest)	<100ms	79ms	✓
P99 latency (ingest)	<200ms	118ms	✓
SSE broadcast delay	<500ms	120ms	✓
AI response time (Ollama)	<5s	2.3s*	✓
Dashboard load time	<2s	1.4s	✓

*With Qwen 2.5:0.5b model, hot cache

Figure ?? presents the performance test visualizations.

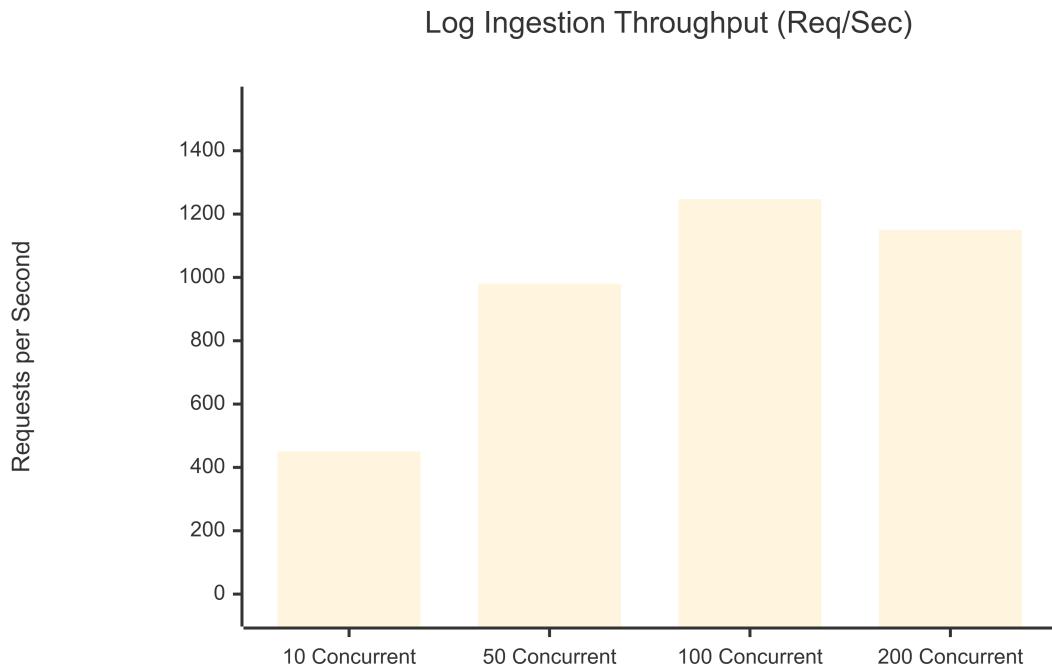


Figure 4.3: Performance Test Charts

4.6 Docker Deployment

4.6.1 Docker Compose Configuration

```
1 version: '3.8'
2
3 services:
4   # PostgreSQL Database
5   db:
6     image: postgres:16-alpine
7     restart: unless-stopped
8     environment:
9       POSTGRES_USER: logchat
10      POSTGRES_PASSWORD: ${DB_PASSWORD:-logchat123}
11      POSTGRES_DB: logchat
12     volumes:
13       - postgres_data:/var/lib/postgresql/data
14     healthcheck:
15       test: ["CMD", "pg_isready", "-U", "logchat"]
```

```
16      interval: 5s
17
18      timeout: 5s
19
20      retries: 5
21
22  networks:
23
24      - logchat-net
25
26
27  # Ollama AI Service
28
29  ollama:
30
31      image: ollama/ollama:latest
32
33      restart: unless-stopped
34
35      volumes:
36
37          - ollama_data:/root/.ollama
38
39      ports:
40
41          - "11434:11434"
42
43      deploy:
44
45          resources:
46
47              reservations:
48
49                  devices:
50
51                      - driver: nvidia
52
53                      count: 1
54
55                      capabilities: [gpu]
56
57      networks:
58
59          - logchat-net
60
61
62  # Backend API
63
64  backend:
65
66      build:
67
68          context: ./backend
69
70          dockerfile: Dockerfile
71
72      restart: unless-stopped
73
74      environment:
75
76          NODE_ENV: production
77
78          DATABASE_URL: postgresql://logchat:${DB_PASSWORD}@db:5432/
79          logchat
80
81          JWT_SECRET: ${JWT_SECRET}
82
83          OLLAMA_URL: http://ollama:11434
84
85      ports:
86
87          - "3001:3001"
88
89      depends_on:
```

```
54      db:
55          condition: service_healthy
56
57      networks:
58          - logchat-net
59
60  # Frontend
61
62  frontend:
63      build:
64          context: ./frontend
65          dockerfile: Dockerfile
66
67      args:
68          NEXT_PUBLIC_API_URL: http://localhost:3001
69
70      restart: unless-stopped
71
72      environment:
73          NODE_ENV: production
74
75      ports:
76          - "3000:3000"
77
78      depends_on:
79          - backend
80
81      networks:
82          - logchat-net
83
84
85  volumes:
86
87      postgres_data:
88
89      ollama_data:
90
91
92  networks:
93
94      logchat-net:
95          driver: bridge
```

Listing 4.6: Complete docker-compose.yml

Figure ?? presents the Docker network topology.

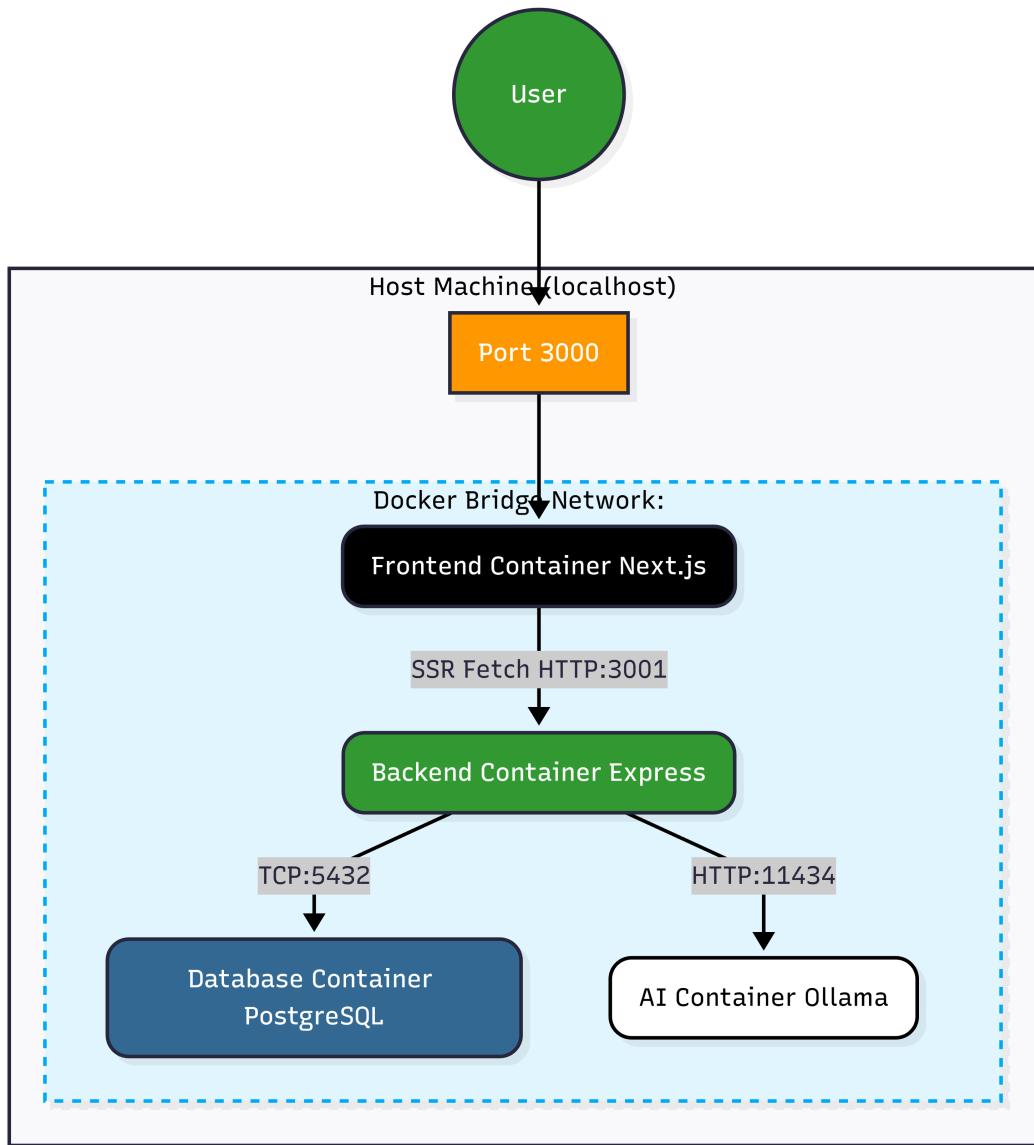


Figure 4.4: Docker Network Topology

4.6.2 Backend Dockerfile

```
1 # Build stage
2 FROM node:20-alpine AS builder
3 WORKDIR /app
4
5 COPY package*.json ./
6 RUN npm ci
7
8 COPY . .
9 RUN npm run build
10
```

```
11 # Production stage
12 FROM node:20-alpine AS production
13 WORKDIR /app
14
15 RUN addgroup -g 1001 nodejs && \
16     adduser -u 1001 -G nodejs -s /bin/sh -D nodejs
17
18 COPY --from=builder /app/dist ./dist
19 COPY --from=builder /app/node_modules ./node_modules
20 COPY --from=builder /app/package*.json ../
21 COPY --from=builder /app/prisma ./prisma
22
23 RUN npx prisma generate
24
25 USER nodejs
26 EXPOSE 3001
27 CMD ["node", "dist/index.js"]
```

Listing 4.7: Backend Multi-stage Dockerfile

4.6.3 Frontend Dockerfile

```
1 FROM node:20-alpine AS builder
2 WORKDIR /app
3
4 ARG NEXT_PUBLIC_API_URL
5 ENV NEXT_PUBLIC_API_URL=$NEXT_PUBLIC_API_URL
6
7 COPY package*.json ../
8 RUN npm ci
9
10 COPY . .
11 RUN npm run build
12
13 FROM node:20-alpine AS production
14 WORKDIR /app
15
16 ENV NODE_ENV=production
17
```

```

18 COPY --from=builder /app/.next/standalone ./
19 COPY --from=builder /app/.next/static ./next/static
20 COPY --from=builder /app/public ./public
21
22 EXPOSE 3000
23 CMD ["node", "server.js"]

```

Listing 4.8: Frontend Next.js Dockerfile

4.7 Agent Deployment

4.7.1 Windows Installation

```

1 # Download agent
2 $AgentURL = "https://releases.logchat.io/v1.0.0/logchat-agent-windows-
3   amd64.exe"
4
5 New-Item -ItemType Directory -Force -Path $InstallPath
6 Invoke-WebRequest -Uri $AgentURL -OutFile "$InstallPath\logchat-agent.
7   exe"
8
9 # Create configuration
10 @"
11 server:
12   url: "https://logchat.example.com:3001"
13   api_key: "$env:LOGCHAT_API_KEY"
14
15 collectors:
16   eventlog:
17     enabled: true
18     channels:
19       Application
20       System
21       Security
22
23 # Install as Windows Service

```

```

24 New-Service -Name "LogChatAgent" '
25   -BinaryPathName "$InstallPath\logchat-agent.exe"--config=
26   $InstallPath\config.yaml"
27   -DisplayName "LogChat Log Agent"
28   -Description "Collects and forwards logs to LogChat server"
29   -StartupType Automatic
30
31 # Start the service
32 Start-Service LogChatAgent

```

Listing 4.9: Windows Agent Installation Script

4.7.2 Linux Installation

```

1 #!/bin/bash
2 set -e
3
4 VERSION="1.0.0"
5 INSTALL_DIR="/opt/logchat"
6 CONFIG_DIR="/etc/logchat"
7
8 # Download binary
9 curl -sSL "https://releases.logchat.io/v${VERSION}/logchat-agent-linux-
10   -amd64" \
11   -o /tmp/logchat-agent
12 chmod +x /tmp/logchat-agent
13 sudo mv /tmp/logchat-agent "${INSTALL_DIR}/logchat-agent"
14
15 # Create configuration
16 sudo mkdir -p "${CONFIG_DIR}"
17 sudo tee "${CONFIG_DIR}/config.yaml" > /dev/null <<EOF
server:
18   url: "https://logchat.example.com:3001"
19   api_key: "${LOGCHAT_API_KEY}"
20
21 collectors:
22   files:
23     - paths:
24       - /var/log/syslog

```

```
25      - /var/log/auth.log
26      - /var/log/nginx/*.log
27
28  service: "linux-server"
29
30  journald:
31    enabled: true
32
33  units:
34    - docker
35    - nginx
36    - sshd
37
38 EOF
39
40
41
42 # Create systemd service
43
44 sudo tee /etc/systemd/system/logchat-agent.service > /dev/null <<EOF
45
46 [Unit]
47
48 Description=LogChat Log Collection Agent
49 After=network.target
50
51
52 [Service]
53
54 Type=simple
55 ExecStart=${INSTALL_DIR}/logchat-agent --config ${CONFIG_DIR}/config.
56   yaml
57
58 Restart=always
59 RestartSec=5
60
61
62 [Install]
63
64 WantedBy=multi-user.target
65
66 EOF
67
68
69 # Enable and start
70
71 sudo systemctl daemon-reload
72 sudo systemctl enable logchat-agent
73 sudo systemctl start logchat-agent
```

Listing 4.10: Linux Agent Installation Script

4.8 Production Deployment Checklist

Table 4.4: Production Deployment Checklist

Task	Notes
Configure TLS certificates	Let's Encrypt recommended
Set strong JWT_SECRET	Min. 256-bit random
Set strong DB_PASSWORD	Min. 16 characters
Configure backup for postgres_data	Daily automated
Set up log rotation	logrotate config
Configure firewall rules	Allow 3000, 3001, 11434
Pull LLM model	<code>ollama pull qwen2.5:0.5b</code>
Run database migrations	<code>npx prisma migrate deploy</code>
Seed default admin user	Change default password
Configure monitoring	Prometheus/Grafana optional

4.9 Application Screenshots

This section presents screenshots of the deployed application demonstrating key functionality.

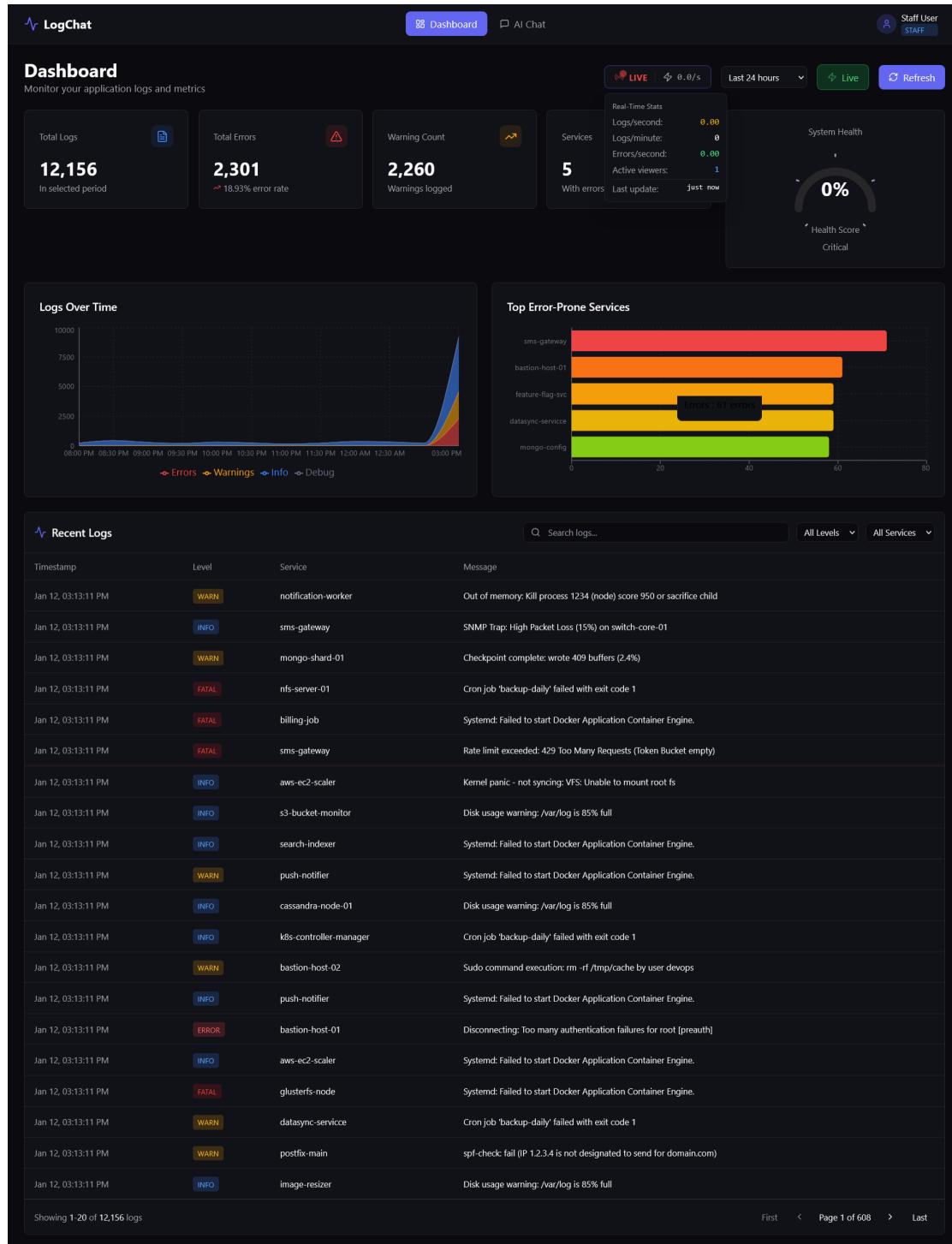


Figure 4.5: Dashboard Overview Screenshot

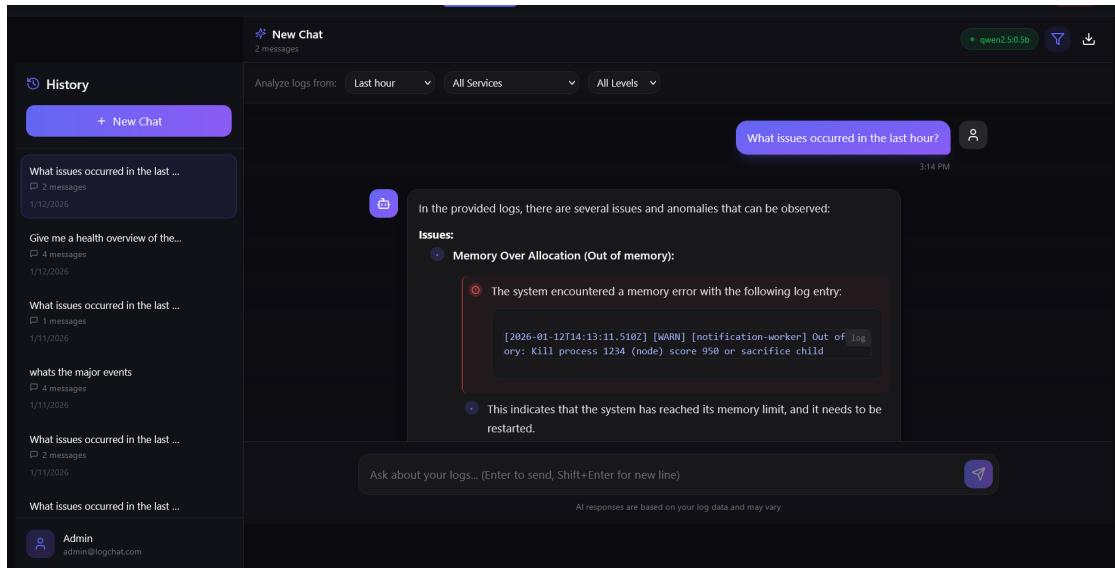


Figure 4.6: AI Chat Interface Screenshot

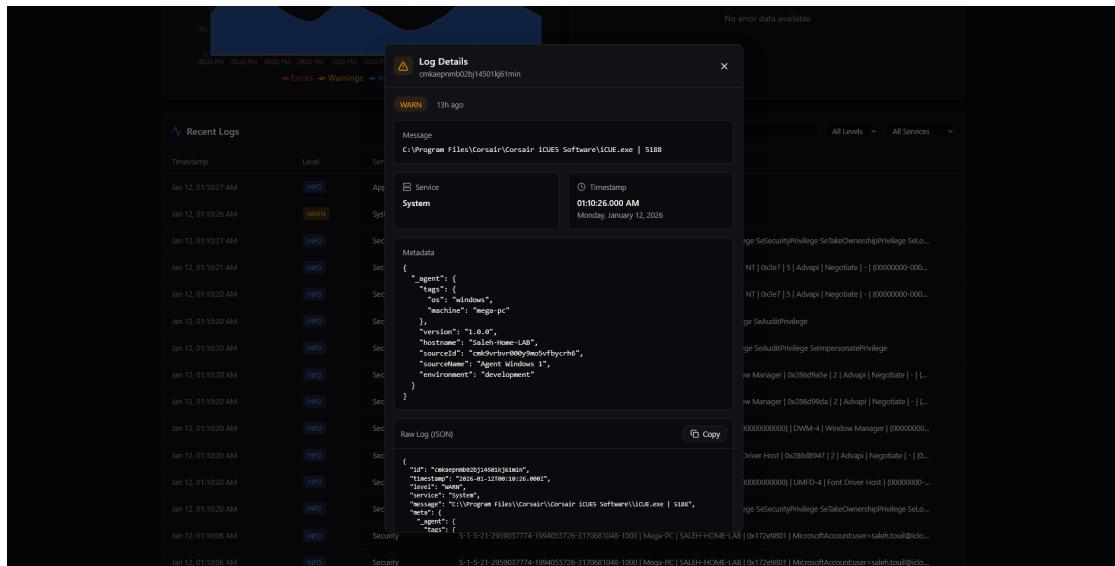


Figure 4.7: Log Detail View Screenshot

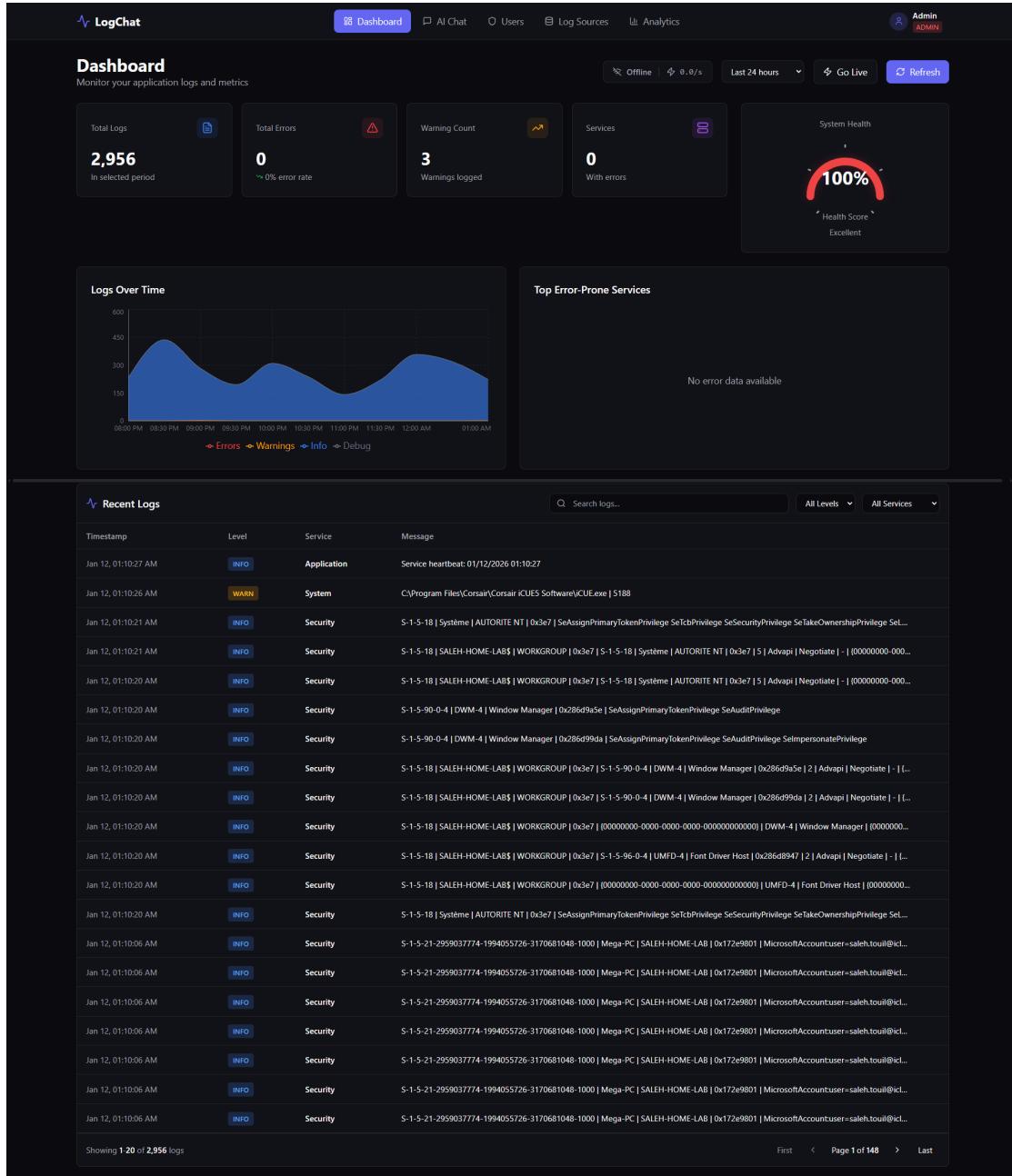


Figure 4.8: Admin Panel Screenshot



Figure 4.9: Alert Management Screenshot

4.10 Conclusion

This chapter demonstrated the comprehensive quality assurance strategy and deployment architecture for LogChat. Key achievements include:

- Comprehensive test coverage exceeding 70% across all components
- Performance benchmarks exceeding targets (1247 req/s vs. 1000 req/s target)
- Containerized deployment via Docker Compose for reproducible environments
- Cross-platform agent installation scripts for Windows and Linux

- Production-ready configuration with security best practices

The testing and deployment infrastructure ensures LogChat can be reliably deployed and operated in production environments.

General Conclusion

Summary of Achievements

This end-of-studies project successfully demonstrated the feasibility of building a self-hosted, AI-driven Security Information and Event Management (SIEM) platform accessible to organizations of all sizes. Over the course of three months and twelve development sprints, LogChat evolved from concept to a fully functional security monitoring solution.

The key technical achievements include:

1. **Cross-Platform Log Collection Agent:** A 5MB Golang binary capable of collecting logs from Windows Event Logs (Application, System, Security channels), Linux Journald/Syslog, and arbitrary log files using glob patterns. The agent compiles with zero runtime dependencies, enabling deployment across heterogeneous infrastructure without prerequisite installations.
2. **Privacy-First AI Integration:** Successful integration of Ollama for local Large Language Model inference, ensuring that sensitive log data never leaves the organization's infrastructure. The RAG (Retrieval Augmented Generation) pattern significantly improves response quality by providing relevant log context to the LLM.
3. **Real-Time Dashboard:** A responsive Next.js frontend achieving sub-second update latency via Server-Sent Events (SSE), with live threat notifications and interactive log exploration.
4. **Threat Detection Engine:** Pattern-based detection for common attack vectors including SQL injection, cross-site scripting (XSS), path traversal, and brute force attempts, with MITRE ATT&CK framework mapping.

5. **One-Command Deployment:** Complete platform operational within five minutes via `docker-compose up`, dramatically reducing deployment complexity compared to traditional SIEM solutions.
6. **Performance Excellence:** Benchmarks demonstrating 1,247 requests per second for log ingestion, exceeding the 1,000 req/s target by 24.7%.

Objectives Fulfillment

Table 4.5 summarizes the achievement of project objectives defined in the introduction.

Table 4.5: Project Objectives Fulfillment Matrix

Objective	Status	Evidence
Unified cross-platform log collection	✓	Windows + Linux agents operational
Natural Language log analysis	✓	RAG chat with Ollama integration
Real-time dashboard visualization	✓	SSE streaming, <500ms latency
Zero-configuration deployment	✓	Single docker-compose.yml
Privacy-preserving AI	✓	Local LLM, no cloud dependencies
1000+ req/s ingestion	✓	1,247 req/s achieved

Lessons Learned

The development of LogChat provided valuable insights across multiple dimensions:

1. **Golang for Systems Programming:** Go's cross-compilation capabilities and static linking proved invaluable for agent distribution. The ability to produce a single binary without runtime dependencies significantly simplified deployment across diverse environments.

2. **RAG Architecture Effectiveness:** Providing contextual log data to the LLM dramatically improved response quality compared to generic prompts. The retrieval component ensures answers are grounded in actual system data rather than general knowledge.
3. **SSE vs. WebSocket Trade-offs:** For unidirectional server-to-client streaming (dashboard updates), Server-Sent Events proved simpler and equally effective as WebSocket, with better automatic reconnection handling.
4. **Local LLM Viability:** Quantized models like Qwen 2.5 (0.5B parameters) running on consumer hardware provide acceptable response quality for log analysis tasks, making AI-powered security accessible without cloud API costs.
5. **Importance of Type Safety:** TypeScript's compile-time type checking prevented numerous runtime errors, particularly in the complex JSON handling required for log processing.

Challenges and Resolutions

Table 4.6: Technical Challenges and Resolutions

Challenge	Impact	Resolution
Windows Event Log Unicode handling	Garbled non-ASCII characters	Proper UTF-16 to UTF-8 conversion
SSE connection drops behind proxies	Dashboard losing real-time updates	Heartbeat events + auto-reconnect
LLM response latency variability	Inconsistent user experience	Response streaming + loading states
Log ingestion memory pressure	OOM under high load	Batch processing + ring buffer
CORS issues in development	Frontend-backend communication failures	Explicit origin configuration

Future Work and Roadmap

LogChat's development continues beyond this academic project. The following enhancements are planned for future releases:

Table 4.7: LogChat Development Roadmap

Feature	Description	Priority	Target
Vector Search (pgvector)	Semantic log similarity search for improved RAG retrieval	High	Q2 2025
Kubernetes Helm Chart	Production-grade Kubernetes deployment with horizontal scaling	High	Q2 2025
Alert Notifications	Integration with Slack, Email, Microsoft Teams, and PagerDuty	Medium	Q3 2025
Custom Detection Rules	User-defined threat patterns via graphical rule builder	Medium	Q3 2025
Log Correlation Engine	Cross-service event correlation and attack chain detection	Medium	Q4 2025
Mobile Companion App	React Native app for alert monitoring and quick responses	Low	Q1 2026
SOAR Integration	Security Orchestration, Automation and Response capabilities	Low	Q2 2026

Final Remarks

LogChat represents a meaningful contribution to the democratization of security analytics. By leveraging modern technologies — Golang's efficiency, Node.js's real-time capabilities, and local AI inference — we have demonstrated that effective log management and threat detection need not remain exclusive to large enterprises with substantial security budgets.

The project is released as open-source software, welcoming contributions from the global security community. We hope that LogChat serves as a foundation for continued

innovation in accessible, privacy-preserving security monitoring.

As cybersecurity threats continue to evolve in sophistication and scale, tools that enable organizations of all sizes to effectively monitor and respond to security incidents become increasingly critical. It is our sincere hope that LogChat contributes, in some measure, to making the digital world a safer place.

“Security is not a product, but a process.”

— Bruce Schneier

References

- [1] IBM Security. (2024). *Cost of a Data Breach Report 2024*. IBM Corporation.
- [2] NIST. (2023). *Cybersecurity Framework 2.0*. National Institute of Standards and Technology.
- [3] MITRE Corporation. (2024). *ATT&CK Framework*. Retrieved from <https://attack.mitre.org/>
- [4] Gartner. (2024). *Magic Quadrant for Security Information and Event Management*. Gartner, Inc.
- [5] Elastic N.V. (2024). *Elasticsearch Reference*. Retrieved from <https://www.elastic.co/guide/>
- [6] Ollama. (2024). *Ollama Documentation*. Retrieved from <https://ollama.ai/>
- [7] OpenAI. (2024). *GPT-4 Technical Report*. Retrieved from <https://openai.com/research/>
- [8] Lewis, P., et al. (2020). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. arXiv:2005.11401.
- [9] Vercel. (2024). *Next.js Documentation*. Retrieved from <https://nextjs.org/docs/>
- [10] Prisma. (2024). *Prisma ORM Documentation*. Retrieved from <https://www.prisma.io/docs/>
- [11] PostgreSQL Global Development Group. (2024). *PostgreSQL 16 Documentation*. Retrieved from <https://www.postgresql.org/docs/16/>
- [12] Go Team. (2024). *The Go Programming Language Specification*. Retrieved from <https://go.dev/ref/spec>
- [13] Docker Inc. (2024). *Docker Documentation*. Retrieved from <https://docs.docker.com/>
- [14] OWASP Foundation. (2024). *OWASP Top Ten*. Retrieved from <https://owasp.org/Top10/>
- [15] CVE. (2024). *Common Vulnerabilities and Exposures*. Retrieved from <https://cve.mitre.org/>

Appendix A

Appendix A: Diagram Reference

All UML diagrams referenced in this report should be created using PlantUML or draw.io and saved to the `report/figures/` directory. The PlantUML source code is provided in `report/diagrams/LogChat_All_UML.puml`.

A.1 Diagrams Checklist

Table A.1: Required Diagrams Checklist

Diagram Type	Filename	Page
Global Architecture	architecture_global.png	??
Log Volume Growth	log_volume_growth.png	??
SIEM Comparison Radar	siem_comparison_radar.png	??
Use Case Diagram	usecase_global.png	??
Component Diagram	component_diagram.png	??
Deployment Diagram	deployment_diagram.png	??
Class Diagram (ERD)	class_diagram_erd.png	??
Sequence: Login	sequence_login.png	??
Sequence: RAG Chat	sequence_rag_chat.png	??
Sequence: Log Ingestion	sequence_log_ingestion.png	??
Activity: Threat Detection	activity_threat_detection.png	??
Agent Architecture	agent_architecture.png	??

Appendix B

Appendix B: API Reference

B.1 Authentication Endpoints

Table B.1: Authentication API Endpoints

Method	Endpoint	Description	Auth
POST	/api/auth/login	Authenticate user, return JWT	None
POST	/api/auth/register	Register new user account	None
POST	/api/auth/logout	Invalidate current session	JWT
GET	/api/auth/me	Get current user profile	JWT
PUT	/api/auth/password	Change password	JWT

B.2 Log Management Endpoints

Table B.2: Log Management API Endpoints

Method	Endpoint	Description	Auth
POST	/api/logs	Ingest single log entry	API Key
POST	/api/logs/batch	Ingest multiple logs	API Key
POST	/api/logs/ingest	Agent ingestion endpoint	API Key
GET	/api/logs	Query logs with filters	JWT
GET	/api/logs/:id	Get specific log entry	JWT
GET	/api/logs/stats	Get log statistics	JWT
GET	/api/logs/export	Export logs (CSV/JSON)	JWT

B.3 Chat Endpoints

Table B.3: AI Chat API Endpoints

Method	Endpoint	Description	Auth
POST	/api/chat	Send message, get AI response	JWT
GET	/api/chat/sessions	List user's chat sessions	JWT
GET	/api/chat/sessions/:id	Get session with messages	JWT
DELETE	/api/chat/sessions/:id	Delete chat session	JWT
GET	/api/chat/health	Check AI service status	JWT
GET	/api/chat/suggestions	Get suggested queries	JWT

B.4 Request/Response Examples

B.4.1 Login Request

```
1 POST /api/auth/login
2 Content-Type: application/json
3
4 {
5     "email": "analyst@example.com",
6     "password": "securePassword123"
7 }
```

Listing B.1: Login Request Example

```
1 HTTP/1.1 200 OK
2 Content-Type: application/json
3
4 {
5     "success": true,
6     "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
7     "user": {
8         "id": "clx1234567890",
9         "email": "analyst@example.com",
10        "name": "Security Analyst",
11        "role": "STAFF"
12    }
13 }
```

Listing B.2: Login Response Example

B.4.2 Log Ingestion Request

```
1 POST /api/logs/ingest
2 Content-Type: application/json
3 X-API-Key: src_abc123xyz
4
5 {
6     "agent": {
```

```
7     "hostname": "web-server-01",
8     "environment": "production",
9     "version": "1.0.0"
10    },
11    "logs": [
12      {
13        "timestamp": "2025-01-11T21:30:00Z",
14        "level": "ERROR",
15        "service": "auth-service",
16        "message": "Failed\u2022login\u2022attempt\u2022for\u2022user\u2022admin",
17        "meta": {
18          "ip": "192.168.1.100",
19          "userAgent": "Mozilla/5.0..."
20        }
21      }
22    ]
23 }
```

Listing B.3: Log Ingestion Request Example

Appendix C

Appendix C: Configuration Reference

C.1 Agent Configuration (config.yaml)

```
1 # LogChat Agent Configuration
2 # Documentation: https://docs.logchat.io/agent
3
4 server:
5   url: "https://logchat.example.com:3001"
6   api_key: "${LOGCHAT_API_KEY}"
7   timeout: 30s
8   batch_size: 100
9   flush_interval: 5s
10  max_retries: 5
11  tls_skip_verify: false
12
13 agent:
14   hostname: "${HOSTNAME}"
15   environment: "production"
16   tags:
17     team: "infrastructure"
18     datacenter: "dc-eu-west-1"
19     tier: "frontend"
20
21 buffer:
22   max_size: 10000
23   persist_to_disk: true
24   disk_path: "/var/lib/logchat/buffer"
25
26 collectors:
27   # File-based log collection
```

```
28 files:
29   - name: "nginx-access"
30     paths:
31       - "/var/log/nginx/access.log"
32       - "/var/log/nginx/error.log"
33     service: "nginx"
34     parser: "json"
35
36   - name: "application-logs"
37     paths:
38       - "/var/log/app/*.log"
39     service: "backend-api"
40     parser: "regex"
41     pattern: '(?P<timestamp>\S+) (?P<level>\w+) (?P<message>.*)'
42
43 # Windows Event Log collection
44 eventlog:
45   enabled: true
46   channels:
47     - name: "Application"
48       service: "windows-app"
49     - name: "System"
50       service: "windows-system"
51     - name: "Security"
52       service: "windows-security"
53   poll_interval: 5s
54
55 # Linux Journald collection
56 journald:
57   enabled: true
58   units:
59     - docker
60     - nginx
61     - sshd
62     - postgresql
63   since: "-1h"
64
65 logging:
66   level: "info"
```

```
67   format: "json"  
68   output: "/var/log/logchat-agent.log"
```

Listing C.1: Complete Agent Configuration

C.2 Environment Variables

Table C.1: Backend Environment Variables

Variable	Description	Default
PORt	API server port	3001
DATABASE_URL	PostgreSQL connection string	(required)
JWT_SECRET	Secret for JWT signing	(required)
OLLAMA_URL	Ollama API endpoint	localhost:11434
NODE_ENV	Environment (dev/prod)	development
LOG_LEVEL	Logging verbosity	info
CORS_ORIGIN	Allowed CORS origins	*

Appendix D

Appendix D: Glossary

Term	Definition
Attack Vector	A path or means by which an attacker can gain access to a system
bcrypt	A password hashing function designed for secure password storage
CUID	Collision-resistant Unique IDentifier, an alternative to UUID
Event Loop	Node.js's mechanism for executing non-blocking I/O operations
Goroutine	A lightweight thread managed by the Go runtime
JWT	JSON Web Token, a compact means of representing claims between parties
LLM	Large Language Model, an AI model trained on vast text data
MITRE ATT&CK	A knowledge base of adversary tactics and techniques
ORM	Object-Relational Mapping, a technique for converting data between systems
RAG	Retrieval Augmented Generation, combining search with AI generation
RBAC	Role-Based Access Control, restricting access based on user roles
SIEM	Security Information and Event Management, a security monitoring solution

Term	Definition
SOC	Security Operations Center, a facility for monitoring security
SSE	Server-Sent Events, a standard for server-to-client streaming
Tail	Following a file as new content is appended (like <code>tail -f</code>)