``

Application Request Orchestrator

Supervised by

Dr. Md. Fazla Elahe
Assistant Professor & Associate Head
Department of Software Engineering
Daffodil International University

Submitted By

Walid Al Hasan
ID: 212-35-730
Department of Software Engineering
Daffodil International University

APPROVAL

This Project titled on **"Application Request Orchestrator"**, submitted by **Walid Al Hasan (ID: 212-35-730)** to the Department of Software Engineering, Daffodil International University has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Bachelor of Science in Software Engineering and approval as to its style and contents.

BOARD OF EXAMINERS

Fazla Elahe

Chairman

Dr. Md. Fazla Elahe
Assistant Professor & Associate Head
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

Marzia

Internal Examiner 1

Dr. Marzia Ahmed
Assistant Professor
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

13.09.2025

Internal Examiner 2

Dr. Shabnom Mustary
Assistant Professor
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

13.09.25

External Examiner

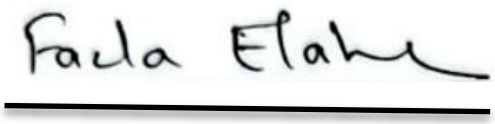
Mohammod Abul Kashem
Professor
Department of Computer Science and Engineering
Dhaka University of Engineering & Technology, Gazipur.

DECLARATION

I hereby declare that; this project has been done by me under the supervision of **Dr. Md. Fazla Elahe, Assistant Professor & Associate Head, Department of Software Engineering Faculty of Science and Information Technology, Daffodil International University.** I also declare that neither this project nor any part of this project has been submitted elsewhere for the award of any degree or diploma.

Walid Al Hasan
ID: 212-35-730
Department of Software Engineering
Daffodil International University

Certified by



Dr. Md. Fazla Elahe
Assistant Professor & Associate Head
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

ACKNOWLEDGEMENT

All praise is due to the **Almighty Allah**, whose endless mercy and blessings have guided us to successfully complete our final year internship. Without His divine help, this achievement would not have been possible.

I am deeply thankful to **My Parents**, whose constant encouragement and sacrifices have been the cornerstone of all my accomplishments. Their support has been a source of strength throughout my journey.

My heartfelt appreciation goes to **Dr. Md. Fazla Elahe, Assistant Professor & Associate Head of Software Engineering at Daffodil International University**. His insightful mentorship, inspiring leadership, and clear direction played a vital role during both the internship period and the report-writing phase. His valuable input helped shape the core understanding reflected in this report.

I would also like to extend my sincere thanks to **Md Tanvir Hasan Joha, MD, Backdoor Private Limited**, for giving the opportunity of doing Internship at his reputed Company and his constant motivation and unwavering belief in my potential throughout this experience.

Special recognition is due to my internship supervisors, **Tahsina Sadia Meem (Forensic Analyst)** and **Shuvo Sarkar (SOC Analyst)** and **GM of Backdoor Mohammad Rashed Mahbub** whose guidance, patience, and constructive feedback were immensely helpful throughout my internship journey.

ABSTRACT

This project presents ABC Proxy Security Gateway, a scalable, high-performance reverse proxy designed as a pluggable web application firewall (WAF) built with Golang. It intercepts and processes HTTP requests through a modular plugin architecture, enabling dynamic security policy enforcement such as SQL injection detection, XSS filtering, and rate limiting — all without coupling to backend application code. The gateway includes a React + Inertia.js dashboard for real-time observability and granular control over plugin lifecycle and rule configuration.

Leveraging PostgreSQL for persistent configuration storage and designed with containerization (Docker) and orchestration (Kubernetes) in mind, the system supports modern cloud-native deployment and scaling strategies. Structured logging and metrics provide full observability into request flows and security actions. The modular design reduces development overhead and increases flexibility compared to traditional monolithic WAF solutions.

Planned enhancements include AI-driven anomaly detection for zero-day threat identification, multi-language plugin support, and TLS termination to inspect encrypted payloads, aiming to meet enterprise-grade security demands and ease operational complexity.

CHAPTER 1: INTRODUCTION

1.1 Project Overview

I developed the **ABC Proxy** project as a modular, plugin-based Web Application Firewall (WAF) and reverse proxy system to protect backend applications from common web-based attacks while giving developers a plug-and-play security framework. I wanted to solve the problem I often saw during my work: traditional WAFs were tightly coupled with backend implementations, forcing developers to bake security logic into their own code. With ABC Proxy, developers can secure their apps without rewriting backend logic.

The system sits between the client and backend application, intercepting HTTP requests and responses, applying security rules, and forwarding legitimate traffic. I built it with a plugin-based architecture so that security modules can be added, removed, or reconfigured at runtime.

1.2 Project Purpose

1.2.1 Background

While interning at **Backdoor Private Limited**, I performed web vulnerability assessments using tools like SQLMap. I discovered that I could bypass existing rate-limiting measures using multiple VPNs. That finding triggered my curiosity about Web Application Firewalls (WAFs). My research showed that most WAFs were tightly integrated with backend logic, requiring developers to constantly adjust or implement their own security solutions.

I wanted to change that by building a **standalone, modular, easily integrable WAF** that functions as a reverse proxy.

1.2.2 Benefits and Beneficiaries

Benefits:

- Centralized security management for multiple backend services.
- Real-time detection and prevention of common attacks like SQL Injection, XSS, and brute force.
- Runtime plugin configuration without service restarts.
- Full observability with metrics, tracing, and logging.

Beneficiaries:

- **Developers** – Can secure apps without touching core backend logic.
- **Security Teams** – Can create/update rules without backend changes.
- **Organizations** – Protect applications with minimal downtime.
- **End Users** – Enjoy safer, more secure applications.

1.2.3 Goal

My primary goal with ABC Proxy was to create an Enterprise level **modular security gateway** that can:

- Act as a high-performance reverse proxy.
- Dynamically load and run security plugins.
- Update rules without service restarts.
- Provide real-time monitoring and observability.

1.3 Stakeholders

Application Developers – Integrate ABC Proxy into deployment pipelines.

Security Analysts – Maintain rules and plugins.

System Administrators – Deploy, configure, and monitor the system.

End Users – Indirectly benefit from secure apps.

1.4 Proposed System Model

1.4.1 Model Structure

For the development of the project I followed the Waterfall Model.

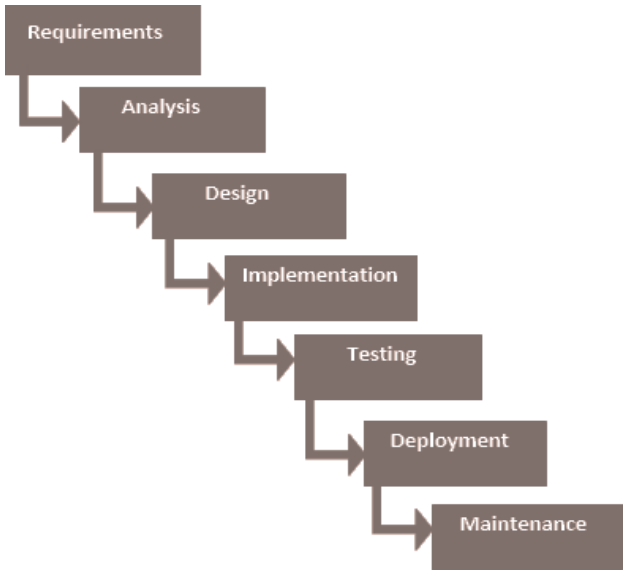


Figure 1: Water Fall Model

And the system uses developed using reverse proxy architecture with a plugin execution chain and organizing the Project in Layered Architecture:

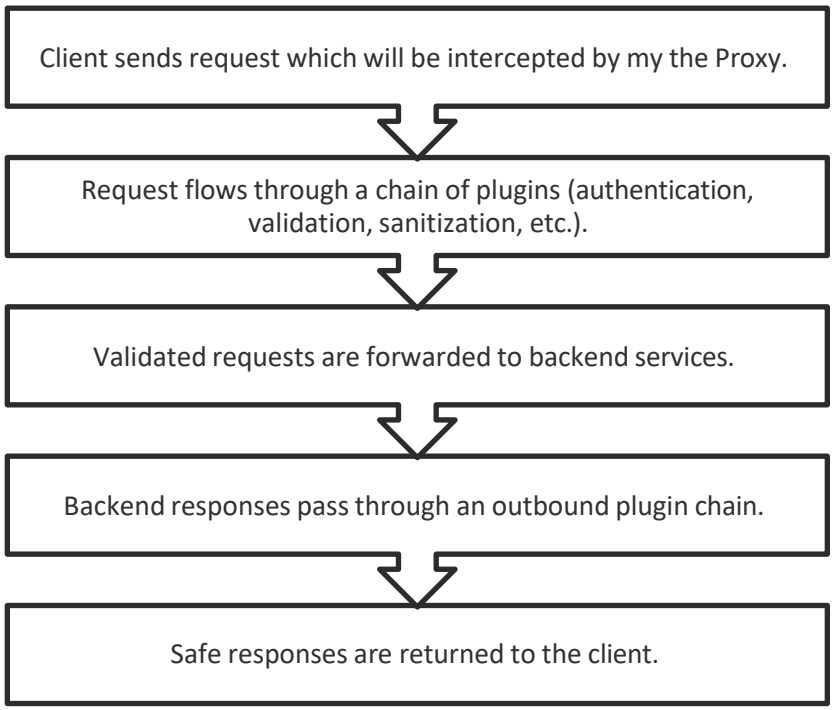


Figure 2: Layered Architecture

1.4.2 How I Used the Model

For the development Model, Waterfall is the best for any personal and one-dev project as it ensures all the necessary stages are followed in a sequence manner. I also followed the same model to create the project.

For the projects architectural project, I used the Reverse proxy Architecture incorporating the Layered Architecture in it. I designed the flow engine (flow.go) to execute registered plugins in a specific order managed by the plugin manager. Each plugin handles its own task (e.g., JWT verification, SQLi detection, rate limiting) and passes the request forward. I made configuration fully dynamic, allowing changes without restarting.

1.5 Project Schedule

1.5.1 Gantt Chart

Development phases:

Phases	July Week-2	July Week-3	July Week-4	Aug Week-1	Aug Week-2	Aug Week-3	Aug Week-4
Requirement Gathering & Research							
System Design & Architecture							
Core Proxy Development							
Plugin Development							
Dashboard Development							
Observability Integration							
Testing & Optimization							
Documentation & Deployment							

Figure 3: Gant Chart

1.5.2 WBS Planning for Development Phase

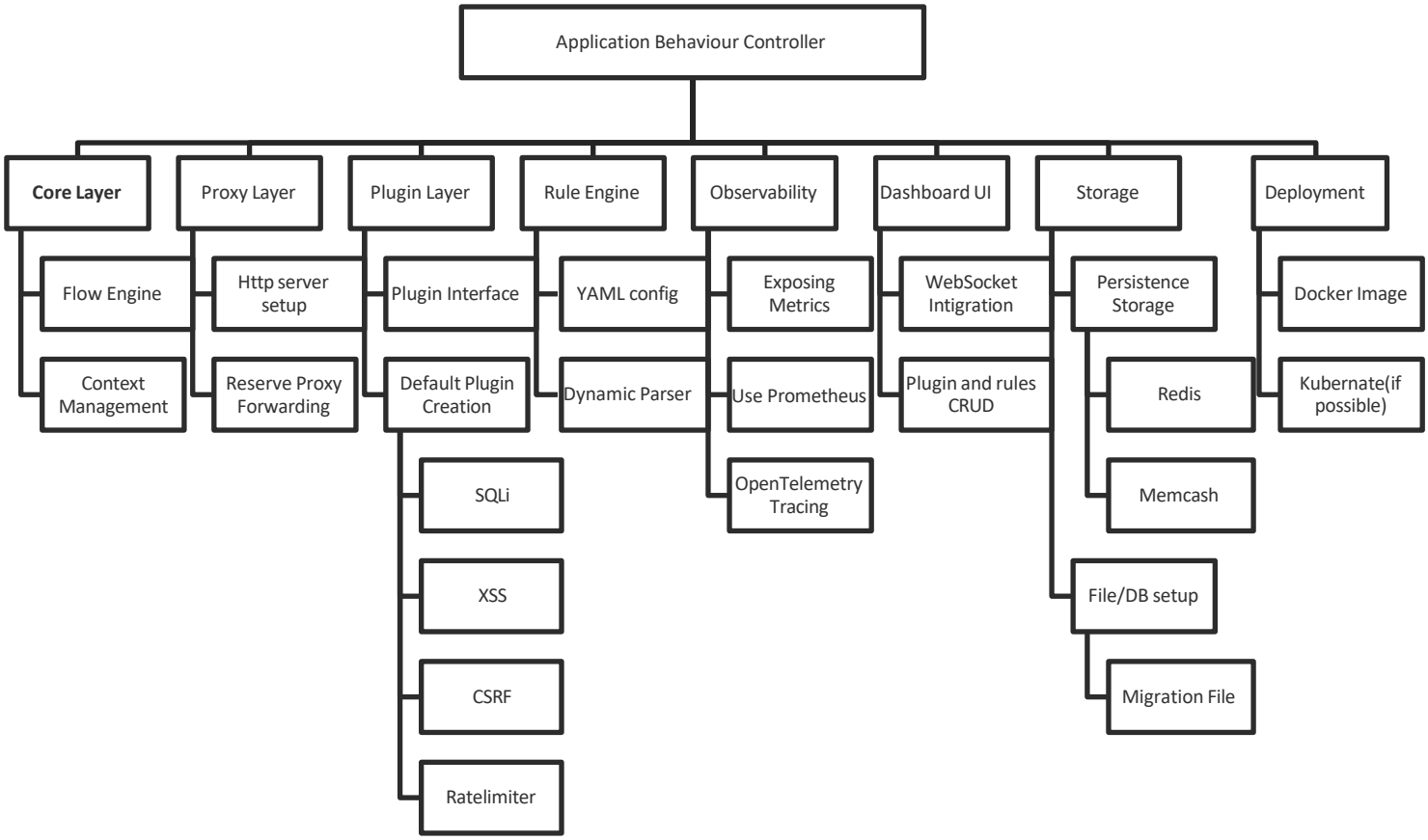


Figure 4: Work Breakdown

CHAPTER 2: SOFTWARE REQUIREMENTS SPECIFICATION (SRS)

2.1 Functional Requirements

Here are the Functional requirements outline the behaviors and features that ABC Proxy will provide.

Proxy Request Handling

FR-01	Proxy Request Handling
Description	Intercept and forward HTTP requests between clients and backend services.
Stakeholder	System Administrator

Plugin Execution

FR-02	Plugin Execution
Description	Execute plugins in a defined order for request/response validation.
Stakeholder	Application Developers

Dynamic Plugin Management

FR-3	Dynamic Plugin Management
Description	Add, remove, or update plugins at runtime without restarting the system.
Stakeholder	Security Analysts

Rule-Based Filtering

FR-4	Rule-Based Filtering
Description	Apply custom security rules for request validation (e.g., SQLi, XSS prevention).
Stakeholder	Security Analysts

Rate Limiting

FR-5	Rate Limiting
Description	Detect and block excessive requests from malicious sources.
Stakeholder	System Administrator

Authentication Plugin

FR-6	Authentication Plugin
Description	Verify JSON Web Tokens before forwarding requests to the backend.
Stakeholder	Application Developers

Dashboard Management

FR-7	Dashboard Management
Description	Provide a UI for monitoring metrics, managing plugins, and configuring rules.
Stakeholder	System Administrator

Dashboard Management

FR-8	Dashboard Management
Description	Collect logs, metrics, and traces for monitoring and debugging.
Stakeholder	Security Analysts

Configuration Persistence

FR-9	Configuration Persistence
Description	Store configuration in persistent storage (e.g., file, Redis).
Stakeholder	System Administrator

Reverse Proxy Performance Tuning

FR-10	Reverse Proxy Performance Tuning
Description	Ensure low-latency request forwarding with high throughput.
Stakeholder	Application Developers

2.2 Non-Functional Requirements

Non-functional requirements describing how my system will operate.

ID	Requirement	Description	Example
NFR-01	Performance	Requests must be processed with < 50ms latency under normal load.	1000 requests/sec handled with 40ms avg latency.
NFR-02	Scalability	System must scale horizontally via container orchestration (e.g., Kubernetes).	Deploying 3+ replicas under load balancing.
NFR-03	Availability	Maintain 99.9% uptime with redundancy and failover mechanisms.	Auto failover between nodes.
NFR-04	Security	All communication must be encrypted via TLS.	HTTPS-only proxy communication.
NFR-05	Usability	Dashboard must be intuitive for both developers and security analysts.	Configure a new plugin in under 2 minutes.

NFR-06	Maintainability	Codebase must follow modular architecture for easy updates.	Adding a new plugin without modifying core logic.
NFR-07	Logging	All requests and plugin outputs must be logged for audit purposes.	Centralized log storage with Kibana.
NFR-08	Interoperability	Must support integration with different backend stacks (Laravel, Go, Node.js).	Proxy Laravel API and Go microservices without config changes.

2.3 Software Requirements

Software Component	Version/Details
Programming Language	Golang 1.21+
Frameworks/Libraries	net/http, Gorilla Mux, OpenTelemetry
Dashboard Framework	React.js/Vue.js with Tailwind CSS
Metrics & Monitoring	Prometheus, Grafana
Containerization	Docker 24+
Orchestration	Kubernetes 1.28+

2.4 Hardware Requirements

Component	Minimum Specification	Recommended Specification
CPU	2 cores	4+ cores
RAM	2 GB	8 GB
Storage	10 GB SSD	50 GB SSD
Network	1 Gbps	10 Gbps

2.5 Dependability Requirements

Reliability

ID	Reliability
Description	Proxy must not crash under high load; use panic recovery and request timeouts.
Stakeholder	System Administrator

Fault Tolerance

ID	DEP-02
Description	Fail gracefully if a plugin fails, skipping it while continuing request processing.
Stakeholder	Application Developers

Data Integrity

ID	DEP-03
Description	Ensure configuration files and rules are validated before applying changes.
Stakeholder	Security Analysts

Backup & Recovery

ID	DEP-04
Description	Maintain backups of configurations and rules for disaster recovery.
Stakeholder	System Administrator

Auditability

ID	DEP-05
Description	Maintain complete logs for compliance and forensic analysis.
Stakeholder	Security Analysts

CHAPTER 3: SYSTEM ANALYSIS

3.1 USE CASE DIAGRAM

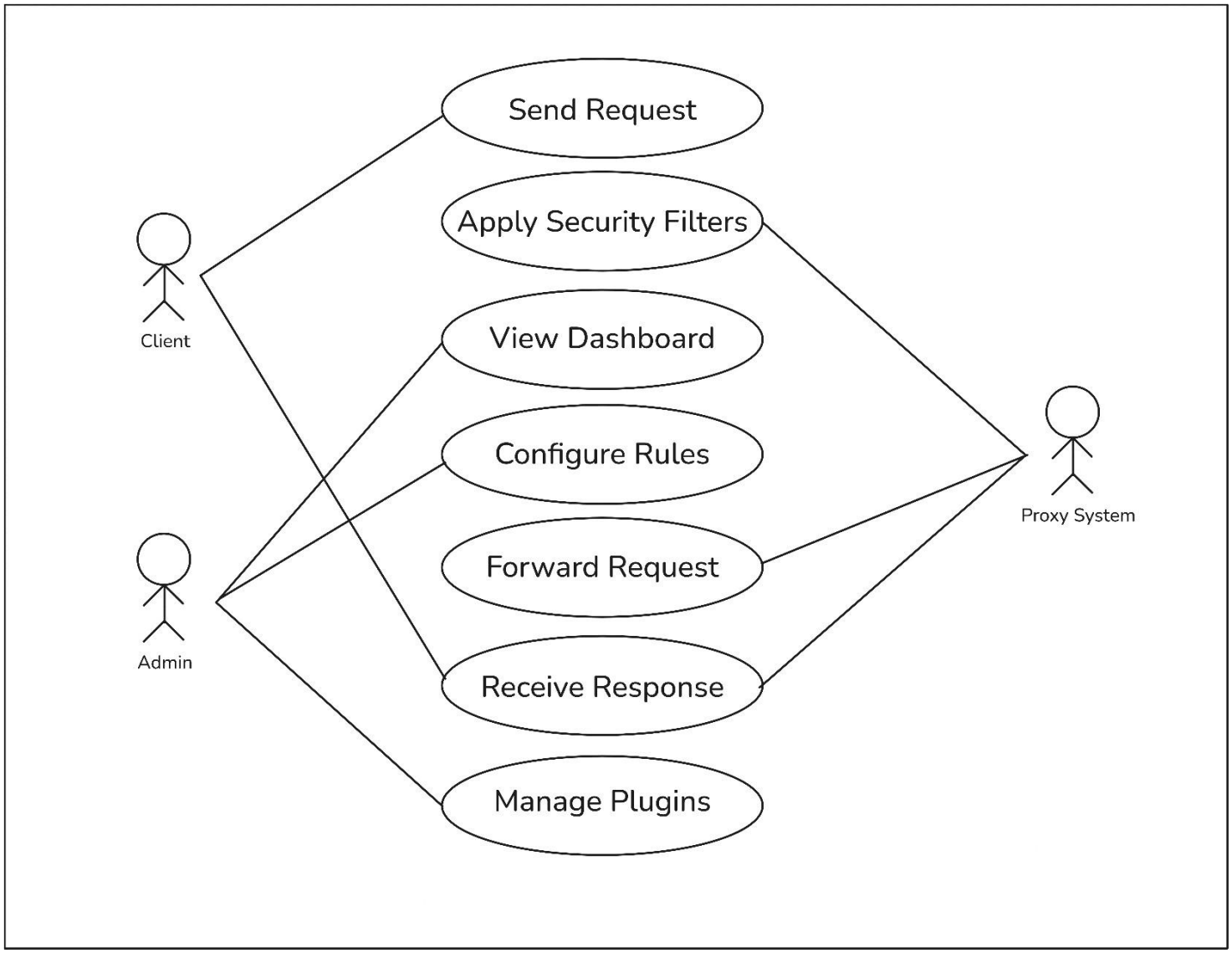


Figure 5: Use Case Diagram

3.2 USE CASE DESCRIPTION

3.2.1 Send Request

Clients send HTTP requests to the proxy server, which validates and forwards them to the backend.

Use Case Name	**Send Request**
Goal	Enable clients to send HTTP requests through the proxy for validation and forwarding
Preconditions	Client is authenticated or allowed to send requests
Primary Actor	Client
Secondary Actor	Proxy System
Trigger	Client sends an HTTP request to the proxy
Description / Main Success Scenario	1. Client sends an HTTP request to the proxy. 2. Proxy receives the request and starts processing.
Post Condition	Request is accepted by the proxy system
Alternative Flow	Request is rejected if validation fails

3.2.2 Apply Security Filters

The proxy applies security filters/plugins on the incoming requests.

Use Case Name	**Apply Security Filters**
Goal	Validate incoming requests using security plugins such as JWT, rate limiting, XSS, SQLi detection
Preconditions	Proxy has received a client request
Primary Actor	Proxy System
Secondary Actor	N/A
Trigger	Receipt of client request
Description / Main Success Scenario	1. Proxy runs the request through the plugin chain. 2. Each plugin validates or modifies the request accordingly.
Post Condition	Request is either passed or blocked
Alternative Flow	Request blocked on any security violation

3.2.3 View Dashboard

Admins view the system status, logs, and metrics via the dashboard.

Use Case Name	View Dashboard
Goal	Allow Admins to monitor proxy system health and plugin states
Preconditions	Admin is authenticated
Primary Actor	Admin
Secondary Actor	Proxy System
Trigger	Admin opens the dashboard UI
Description / Main Success Scenario	1. Admin logs in. 2. Dashboard shows real-time system metrics and plugin status.
Post Condition	Admin is informed about system health
Alternative Flow	Error shown if dashboard data fails to load

3.2.4 Configure Rules

Admins configure security rules and plugin settings via the dashboard.

Use Case Name	Configure Rules
Goal	Enable Admins to add/edit/remove security rules and plugins dynamically
Preconditions	Admin is logged in
Primary Actor	Admin
Secondary Actor	Proxy System
Trigger	Admin submits rule or plugin changes
Description / Main Success Scenario	1. Admin edits rules/plugins. 2. Proxy validates and applies changes in real-time.
Post Condition	New rules/plugins are active and saved
Alternative Flow	Rejected if input validation fails

3.2.5 Forward Request

The proxy forwards validated requests to the backend services.

Use Case Name	Forward Request
Goal	Send validated client requests to backend services
Preconditions	Request passed all security filters
Primary Actor	Proxy System
Secondary Actor	Backend Services
Trigger	Security validation successful
Description / Main Success Scenario	1. Proxy sends the HTTP request to backend. 2. Backend acknowledges the request.
Post Condition	Request is received by backend for processing
Alternative Flow	Request forwarding fails if backend is unreachable

3.2.6 Receive Response

The proxy receives backend responses and sends them back to clients.

Use Case Name	**Receive Response**
Goal	Receive responses from backend and relay to clients
Preconditions	Backend has processed the request
Primary Actor	Proxy System
Secondary Actor	Backend Services
Trigger	Backend sends response
Description / Main Success Scenario	1. Proxy receives response from backend. 2. Proxy forwards response to the client.
Post Condition	Client receives the backend response
Alternative Flow	Proxy returns error if backend response missing or corrupted

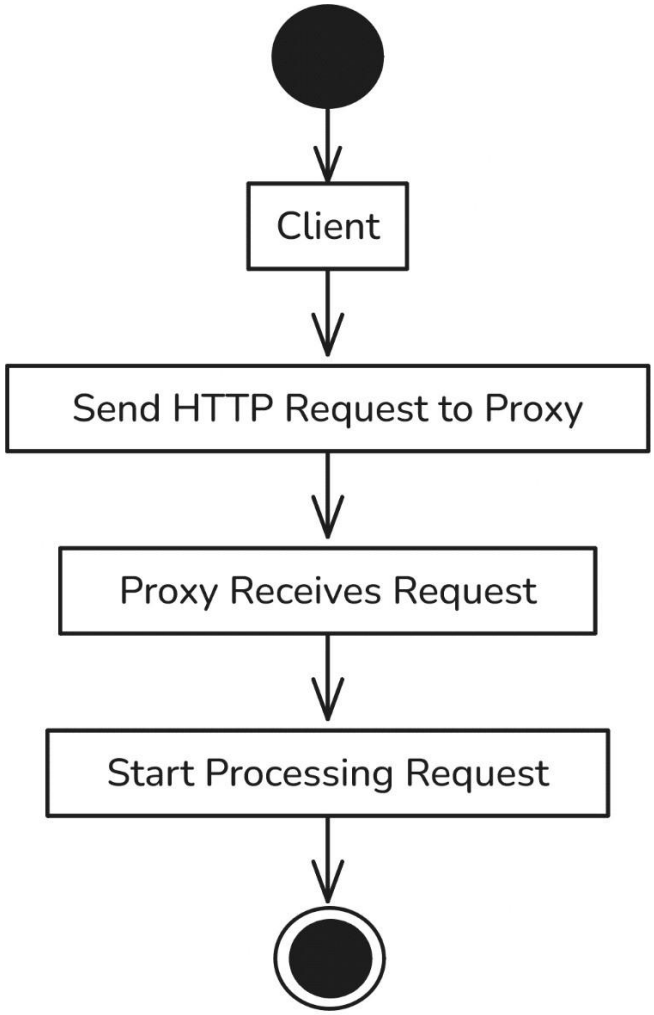
3.2.7 Manage Plugins

Admins add, update, or remove security plugins dynamically at runtime.

Use Case Name	**Manage Plugins**
Goal	Allow Admins to manage plugins without downtime
Preconditions	Admin is logged in
Primary Actor	Admin
Secondary Actor	Proxy System
Trigger	Admin submits plugin changes
Description / Main Success Scenario	1. Admin adds/updates/removes plugins. 2. System validates and applies plugin changes live.
Post Condition	Plugins updated and active
Alternative Flow	Changes rejected if plugin incompatible or invalid

3.3 ACTIVITY DIAGRAM

I created activity diagrams using the system's use cases as a guide. Each of the system's essential functions are shown in these diagrams.



Send Request

Figure 6: Activity Diagram

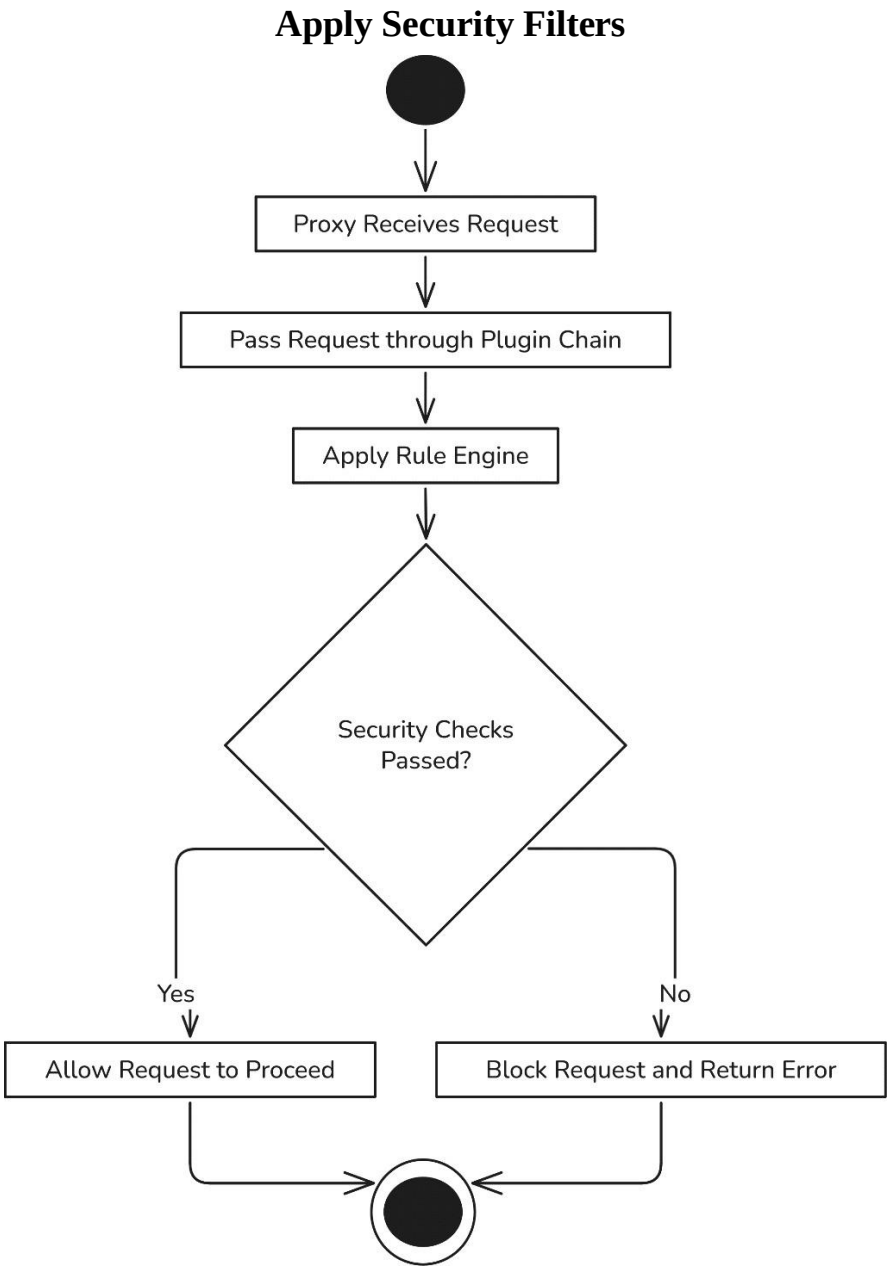


Figure 7: Activity Diagram of Apply Security Filters

View Dashboard

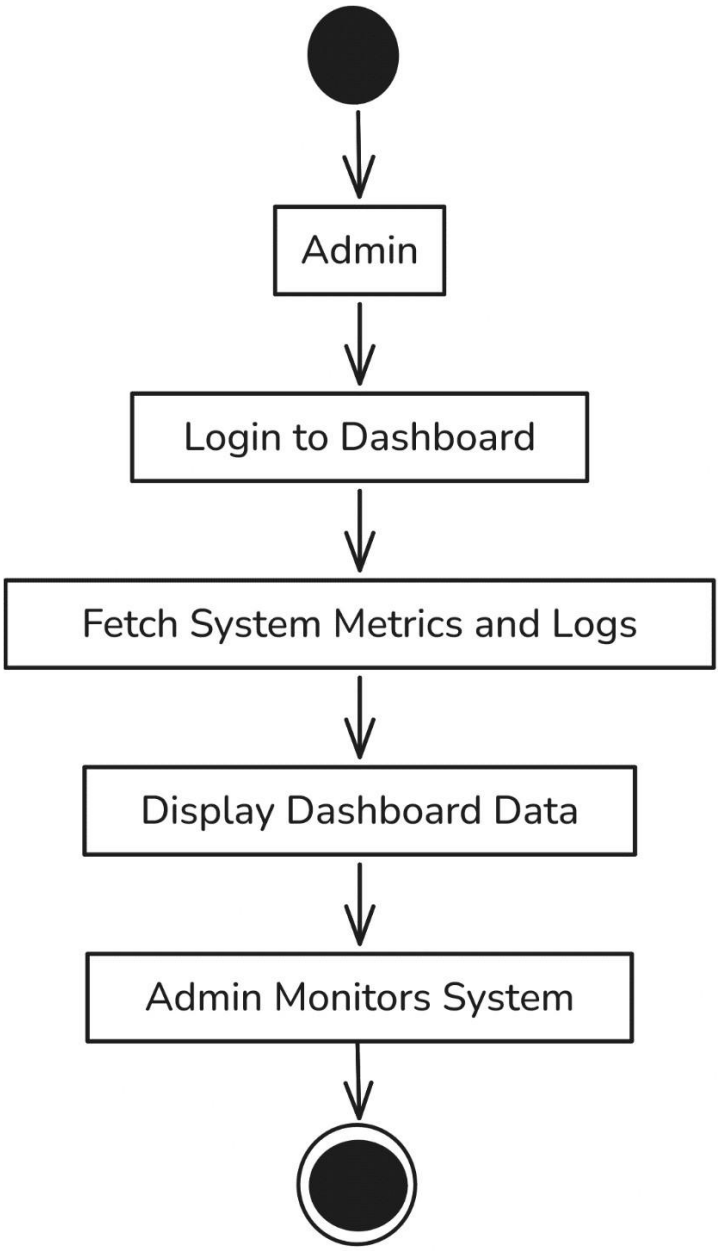


Figure 8: View Dashboard

Configure Rules

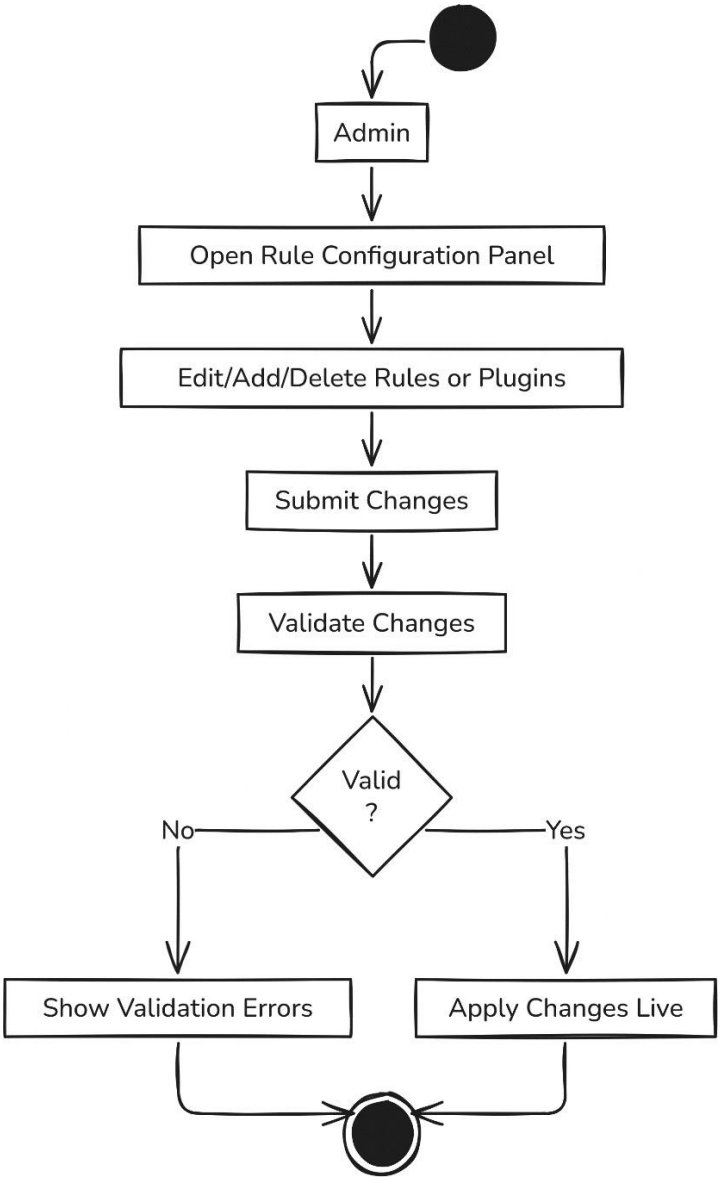


Figure 9: Configure Rules

Forward Request

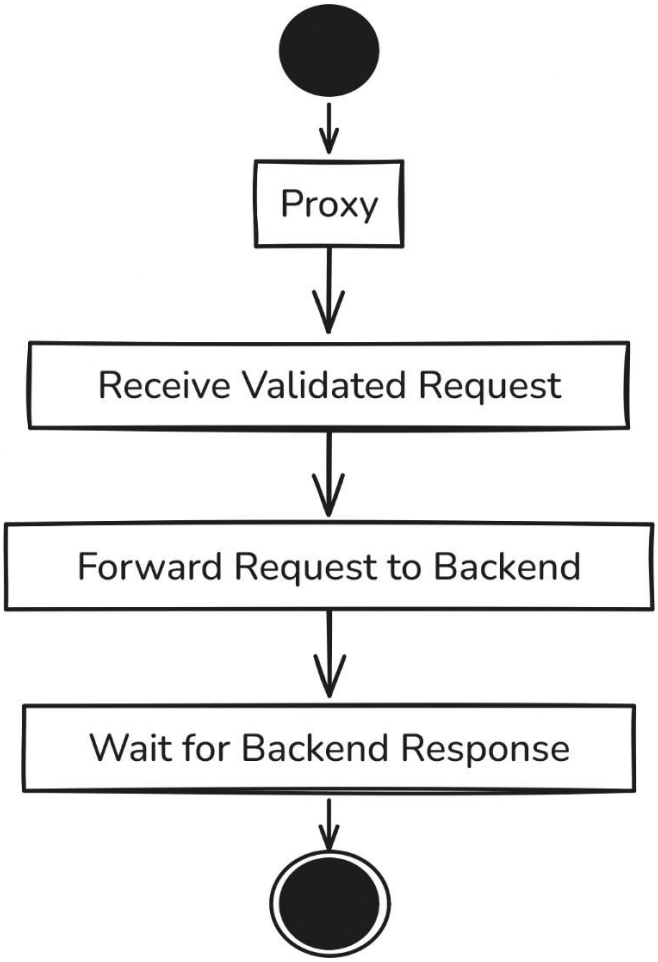


Figure 10: Forward Request

Receive Response

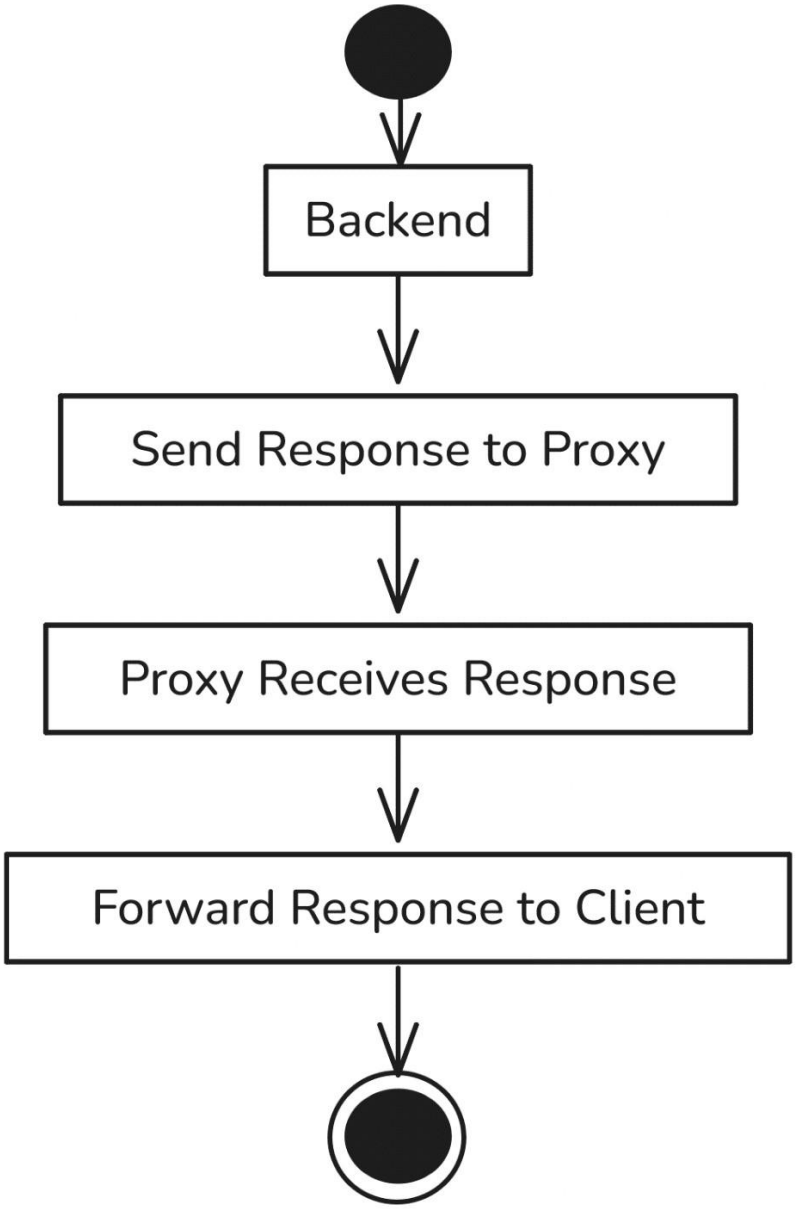


Figure 11: Receive Response

Manage Plugins

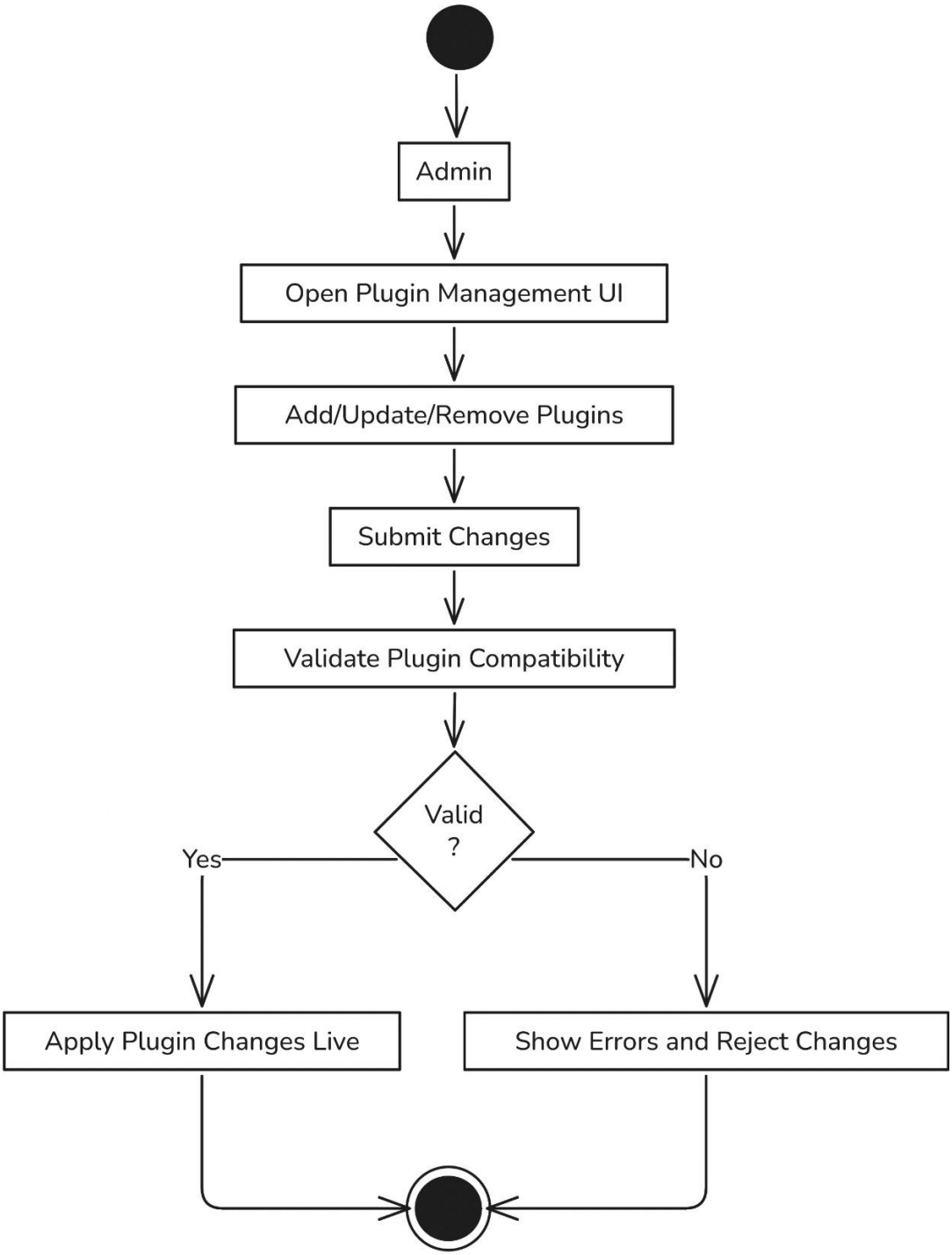


Figure 12: Manage Plugin

Activity Diagram Combined Flow

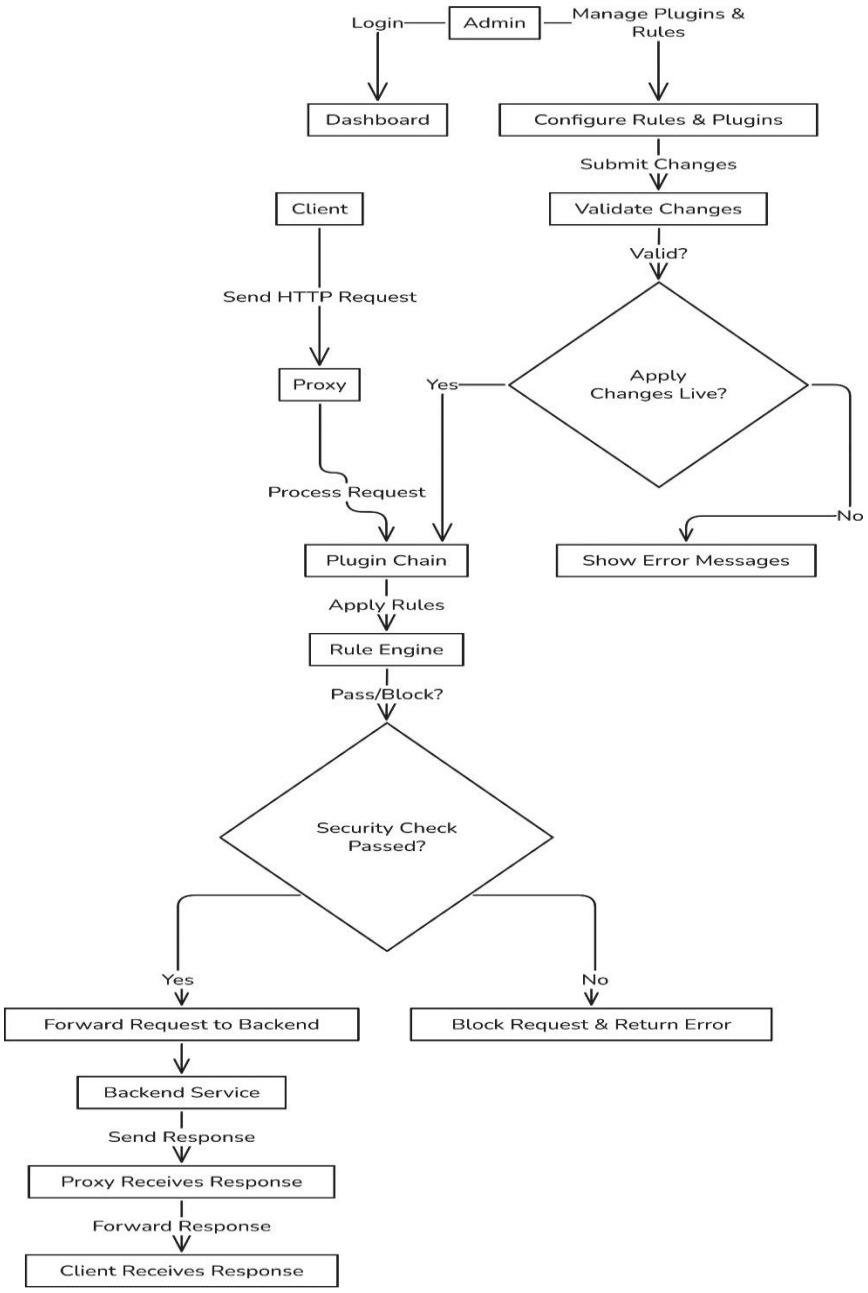


Figure 13: Combined Flow

SEQUENCE DIAGRAM

Send Request

The sequence diagram shows how the system components interact step-by-step to complete each process:

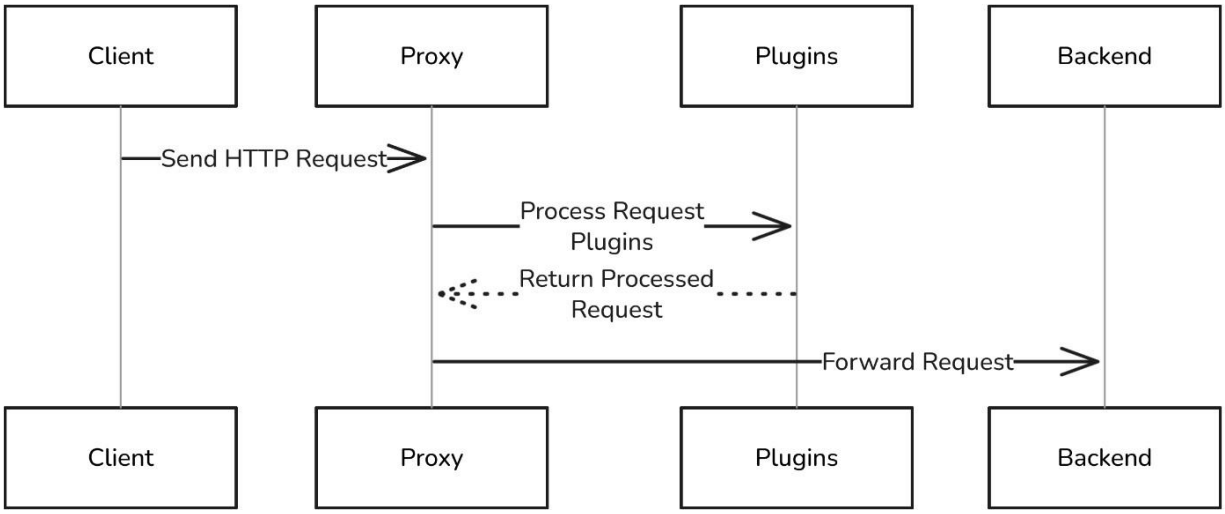


Figure 14: Send Request

Apply Security Filters

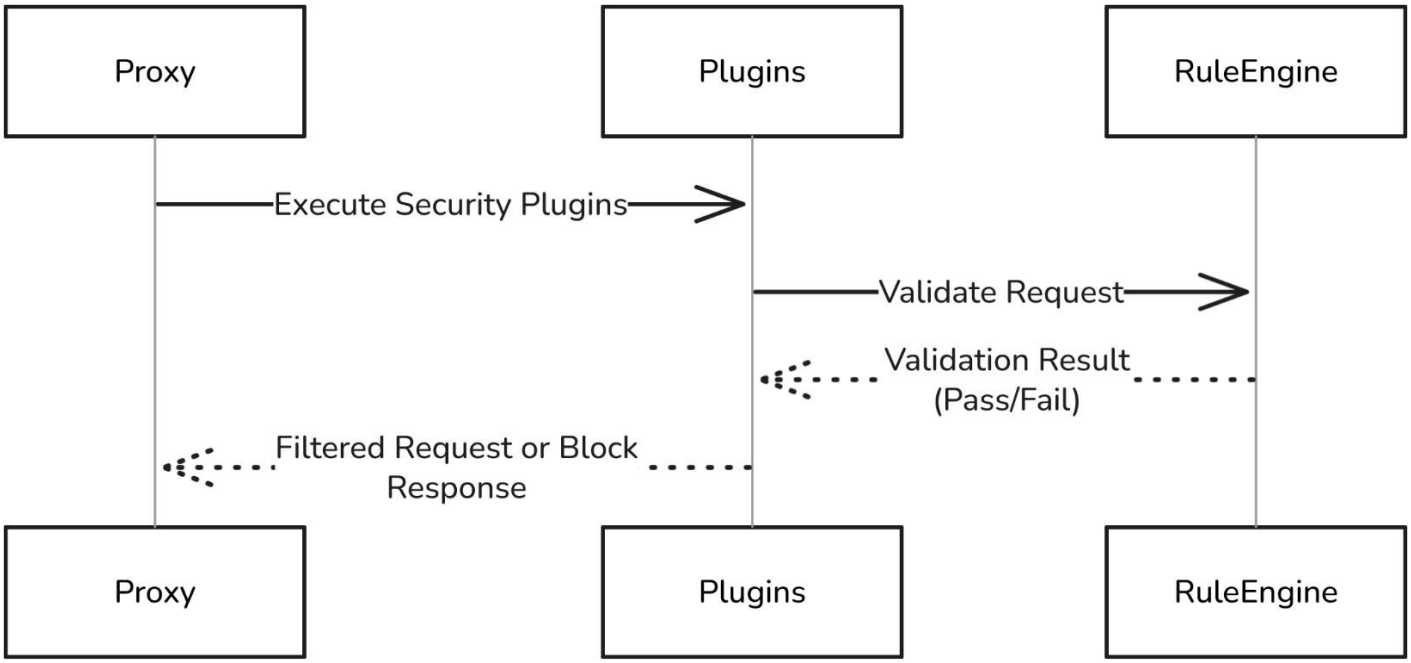


Figure 15: Apply Security Filter

View Dashboard

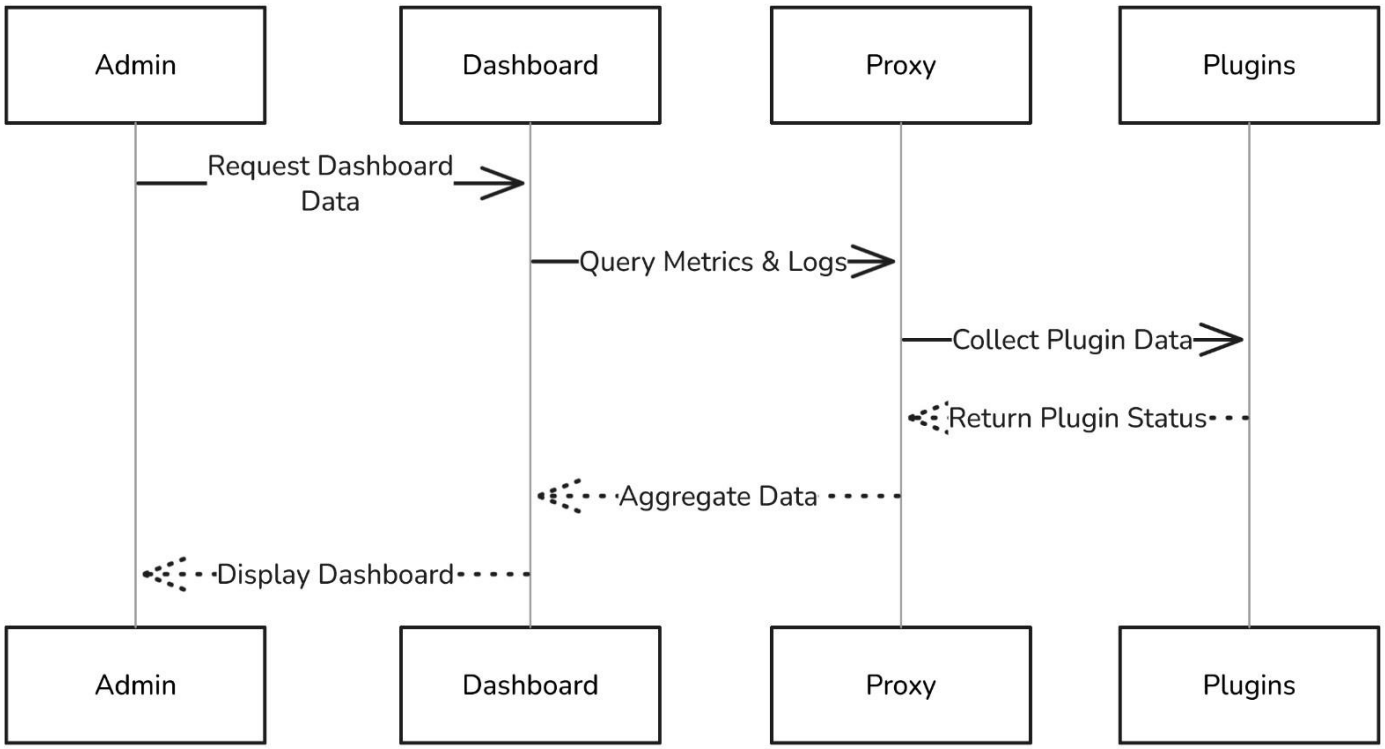


Figure 16: View Dashboard

Configure Rules

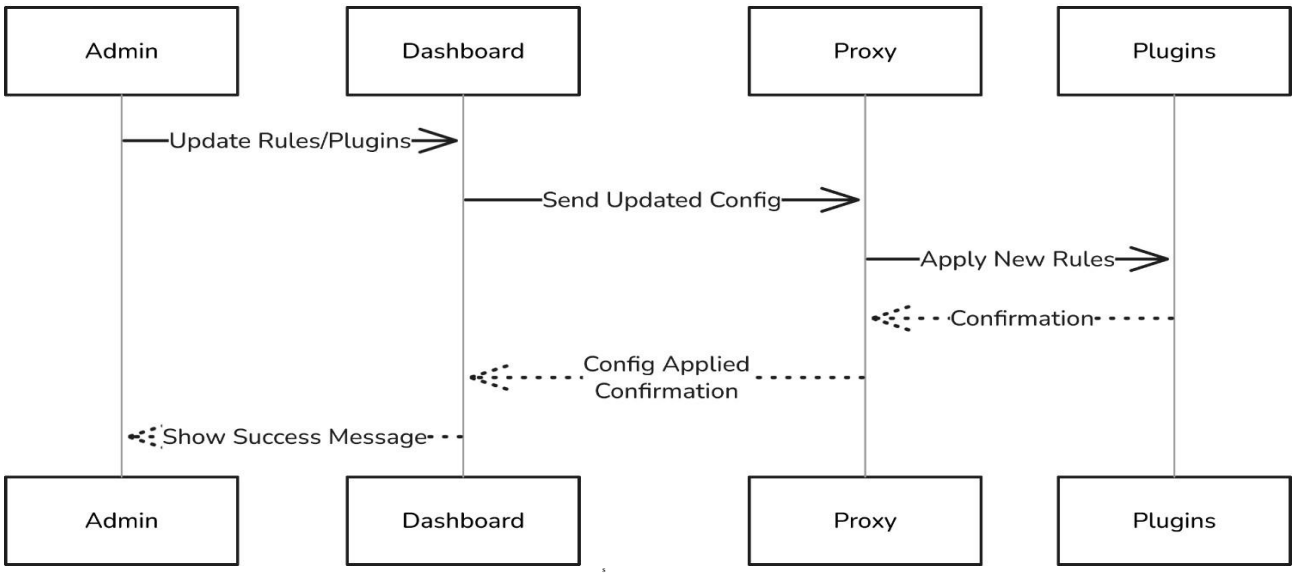


Figure 17: Configure Rules

Forward Request

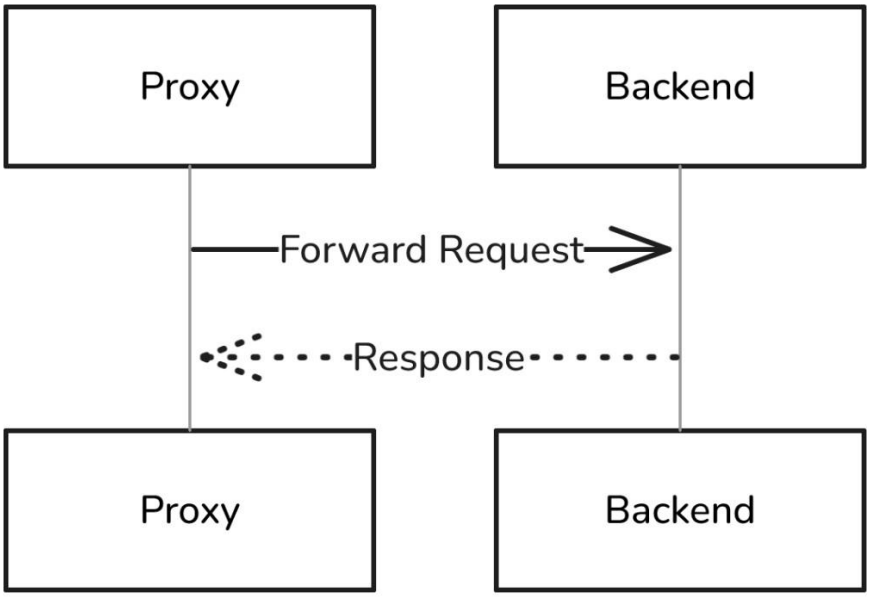


Figure 18: Forward Request

Receive Response

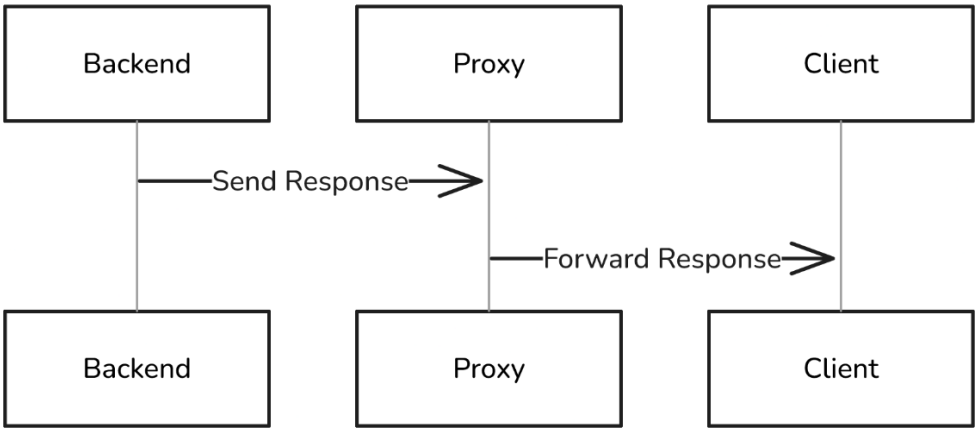


Figure 19: Receive Response

Manage Plugins

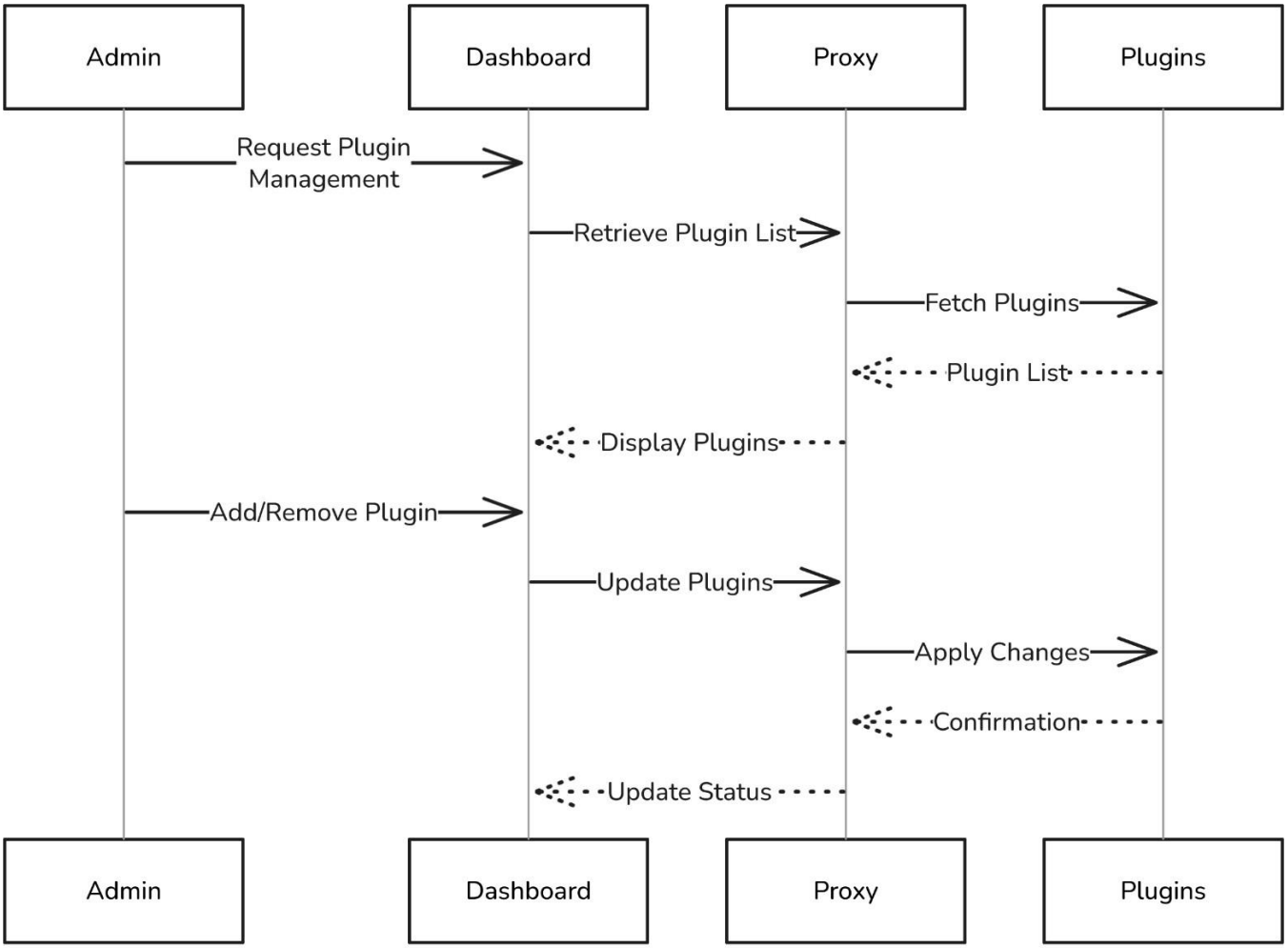


Figure 20: Manage Plugins

Sequence Diagram Combined Flow

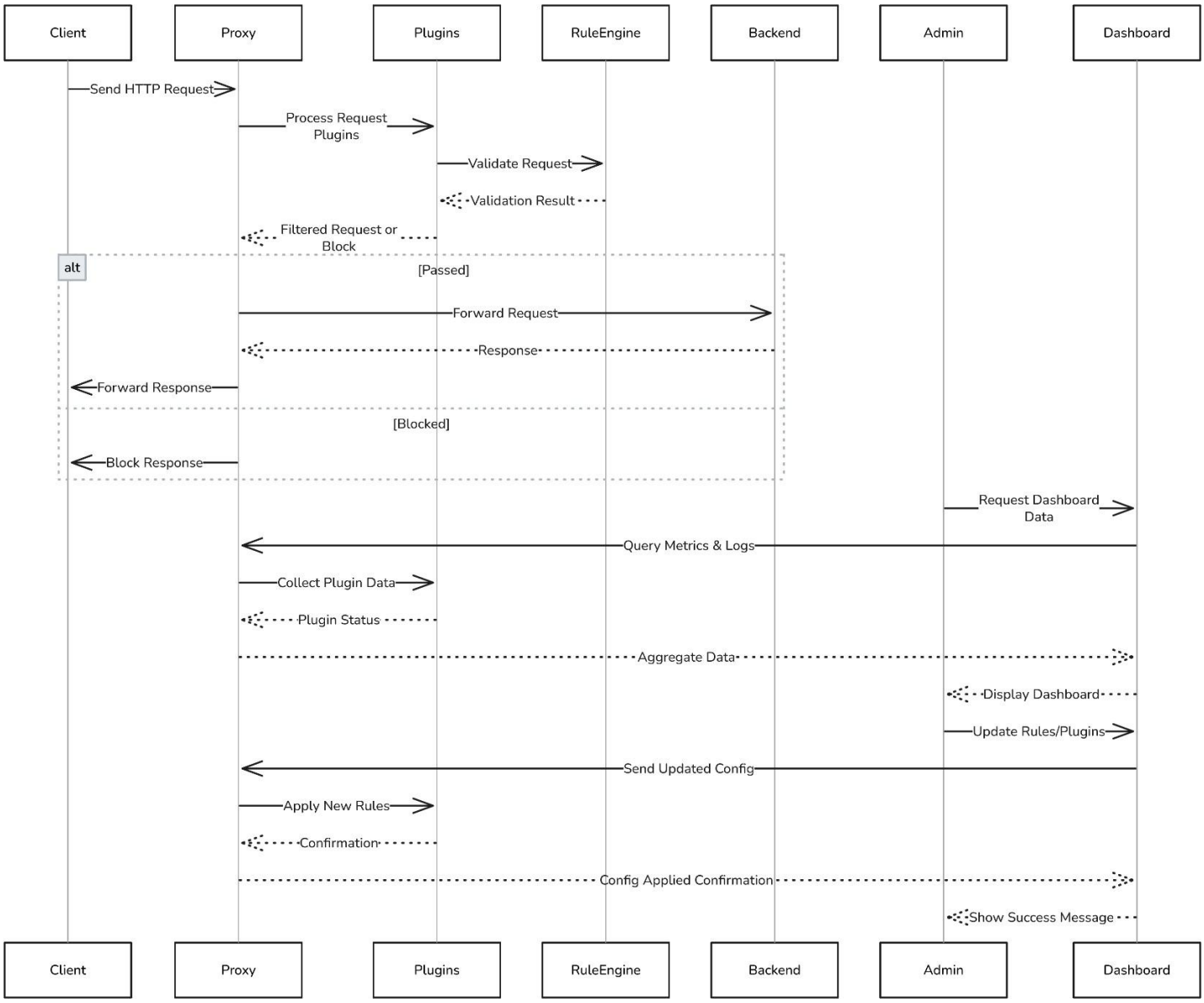


Figure 21: Sequence Diagram Combined Flow

3.4 Data Flow Diagram

DFD Level 0

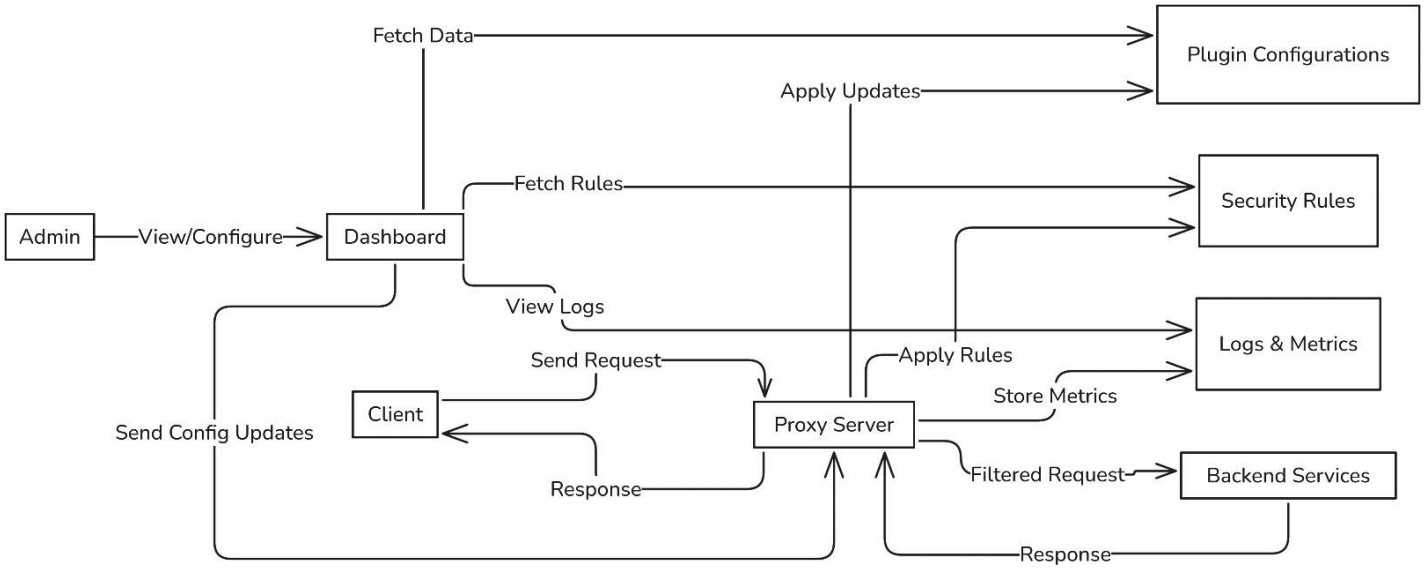


Figure 22: DFD Level 0

DFD Level 1

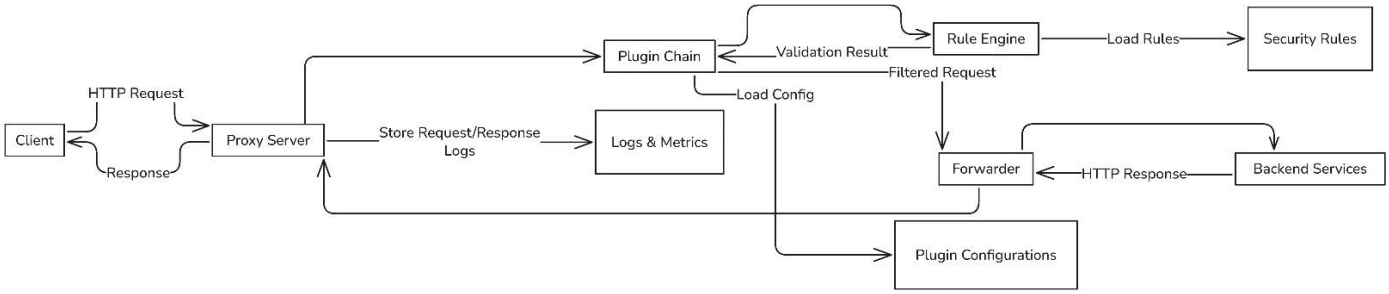


Figure 23: DFD Level 1

3.6 ENTITY RELATIONSHIP DIAGRAM

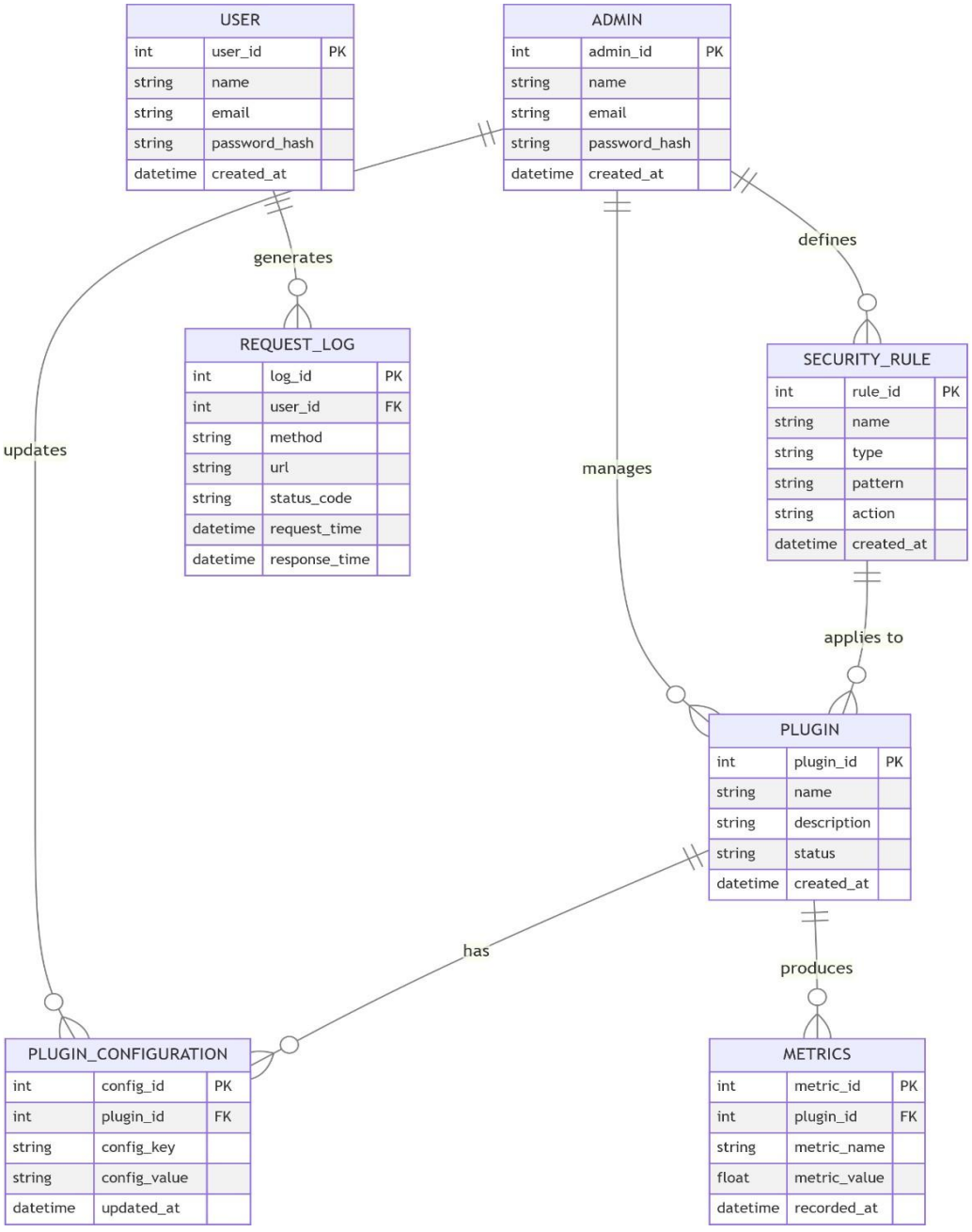


Figure 24: Entity Relationship Diagram

CHAPTER 4 : SYSTEM TESTING

4.1 Introduction to System Testing

System testing verifies that the **ABC Proxy Security Gateway** works as intended when all modules including the proxy, plugin chain, rule engine, dashboard, and backend integrations operate together.
The goal is to ensure:

- Functional correctness
- Security rule enforcement
- Plugin execution flow
- Performance and reliability under load

4.2 Testing Strategies

4.2.1 Test Approach

- **Unit Testing** for each plugin, rule, and core module.
- **Integration Testing** between proxy, plugins, and backend.
- **System Testing** for end-to-end request flow.
- **Security Testing** for vulnerabilities (SQLi, XSS, CSRF).
- **Performance Testing** under simulated traffic.

4.2.2 Pass / Fail Criteria

Pass: The feature meets requirements without defects.

Fail: The feature deviates from expected results or causes system errors.

4.2.3 White Box Testing

Test internal code logic, plugin execution order, and rule validation functions.

Example: Ensure XSSFilterPlugin correctly sanitizes input before forwarding.

4.2.4 Black Box Testing

Test request/response behavior without viewing internal code.

Example: Send malicious payloads and verify they are blocked.

4.3 Testing Schedule

Phase	Start Date	End Date
Unit Testing	Aug 2025	Aug 2025
Integration Testing	Aug 2025	Aug 2025
System Testing	Aug 2025	Aug 2025
Performance & Security Testing	Aug 2025	Aug 2025

4.4 System Test Case

Test Case	Designed By	Design Date	Executed By	Executed Date	Pass/Fail	Comment
1. User Registration	Walid	15 June 2025	Walid	22 June 2025	Pass	Registration form works correctly with valid input.
2. User Login	Walid	15 June 2025	Walid	22 June 2025	Pass	Login redirects to dashboard; error shown for wrong credentials.
3. Send Request via Proxy	Walid	16 June 2025	Walid	23 June 2025	Pass	Requests are correctly routed through the proxy to the backend.
4. Apply Security Filters	Walid	16 June 2025	Walid	23 June 2025	Pass	Malicious payloads (XSS, SQLi) are blocked successfully.
5. View Dashboard	Walid	17 June 2025	Walid	24 June 2025	Pass	Dashboard displays live metrics and request logs.
6. Configure Rules	Walid	17 June 2025	Walid	24 June 2025	Pass	Rules can be created, updated, and applied without error.
7. Forward Request	Walid	18 June 2025	Walid	25 June 2025	Pass	Requests are forwarded only if they pass security filters.
8. Receive Response	Walid	18 June 2025	Walid	25 June 2025	Pass	Response from backend is returned correctly to client.
9. Manage Plugins	Walid	18 June 2025	Walid	25 June 2025	Pass	Plugins can be activated, deactivated, and configured dynamically.

CHAPTER 5: DEVELOPMENT TOOL AND TECHNOLOGY

5.1 Development Technology

- **Golang (Go):**
  Primary backend programming language for building the proxy server, plugin system, and rule engine with high concurrency support.
- **JavaScript (React/Vue)**
  Frontend framework for building the interactive admin dashboard, rule configuration panel, and live request monitoring.
- **HTML5:**
  Structured the dashboard and UI components.
- **CSS3 + Tailwind CSS:**
  Styled the dashboard with a clean, responsive layout.
- **PostgreSQL:**
  Used for storing rules, plugin configurations, and admin user data.
- **REST API (JSON):**
  Communication between frontend dashboard and backend proxy for rule management and monitoring.

5.2 Development Tools and Platforms

- **Visual Studio Code (VS Code):**
  Main code editor with Go, JavaScript, and Docker extensions for fast development.
- **Google Chrome DevTools:**
  Debugged and tested dashboard UI and API responses.
- **Command Line Interface (CLI):**
  Used for running Go commands, managing Git, and executing Docker builds.
- **Git & GitHub:**
  Version control for source code, collaborative development, and backup.
- **XAMPP:**
  Used for quick local testing of dashboard integrations.
- **Postman:**
  Tested and verified backend API endpoints.
- **Docker:**
  Containerized the proxy and dashboard for deployment in a microservices architecture.
- **Kubernetes (K8s):**
  Managed scalable deployments in a cloud environment.
- **Prometheus/Grafana** for observability

CHAPTER 6: USER INTERFACE

Figure 25: Dashboard

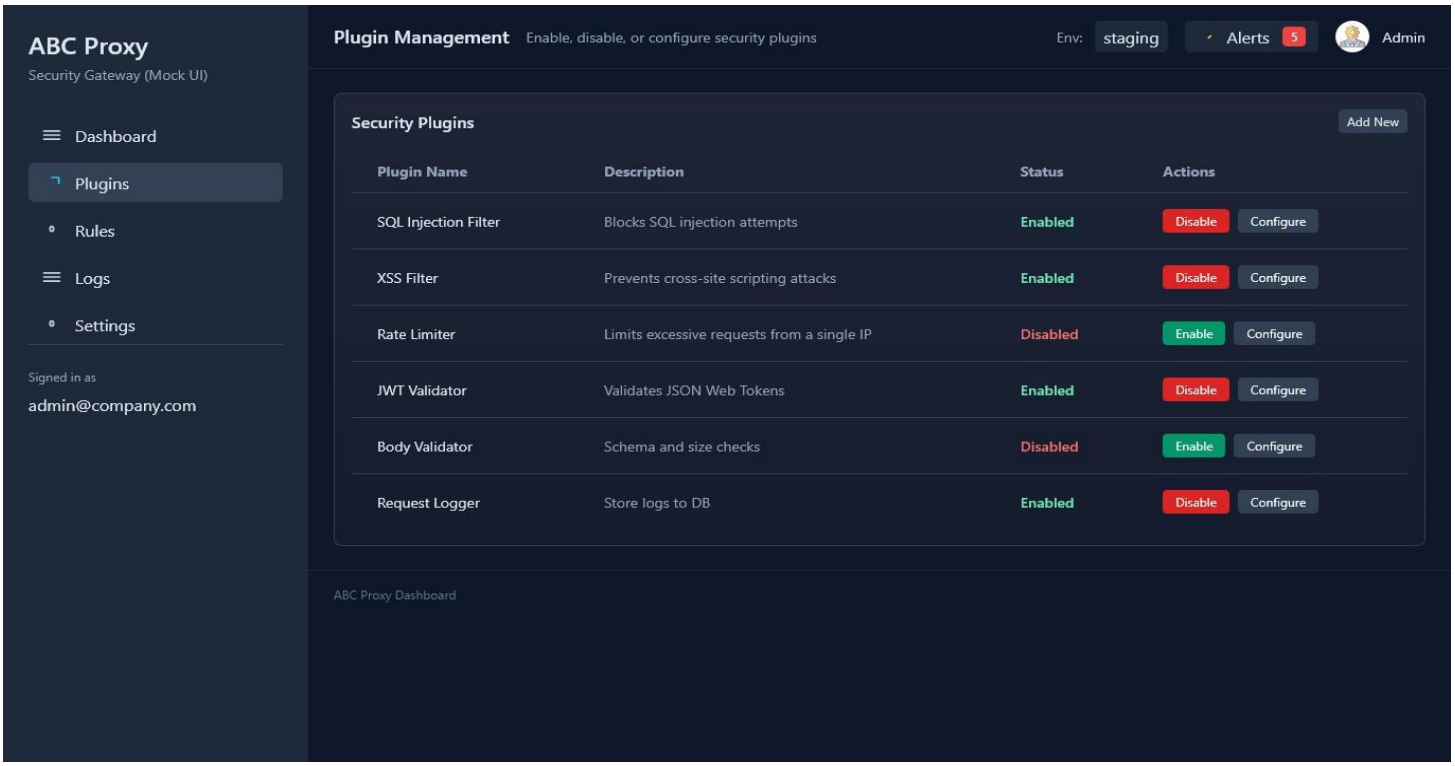


Figure 26: Plugins

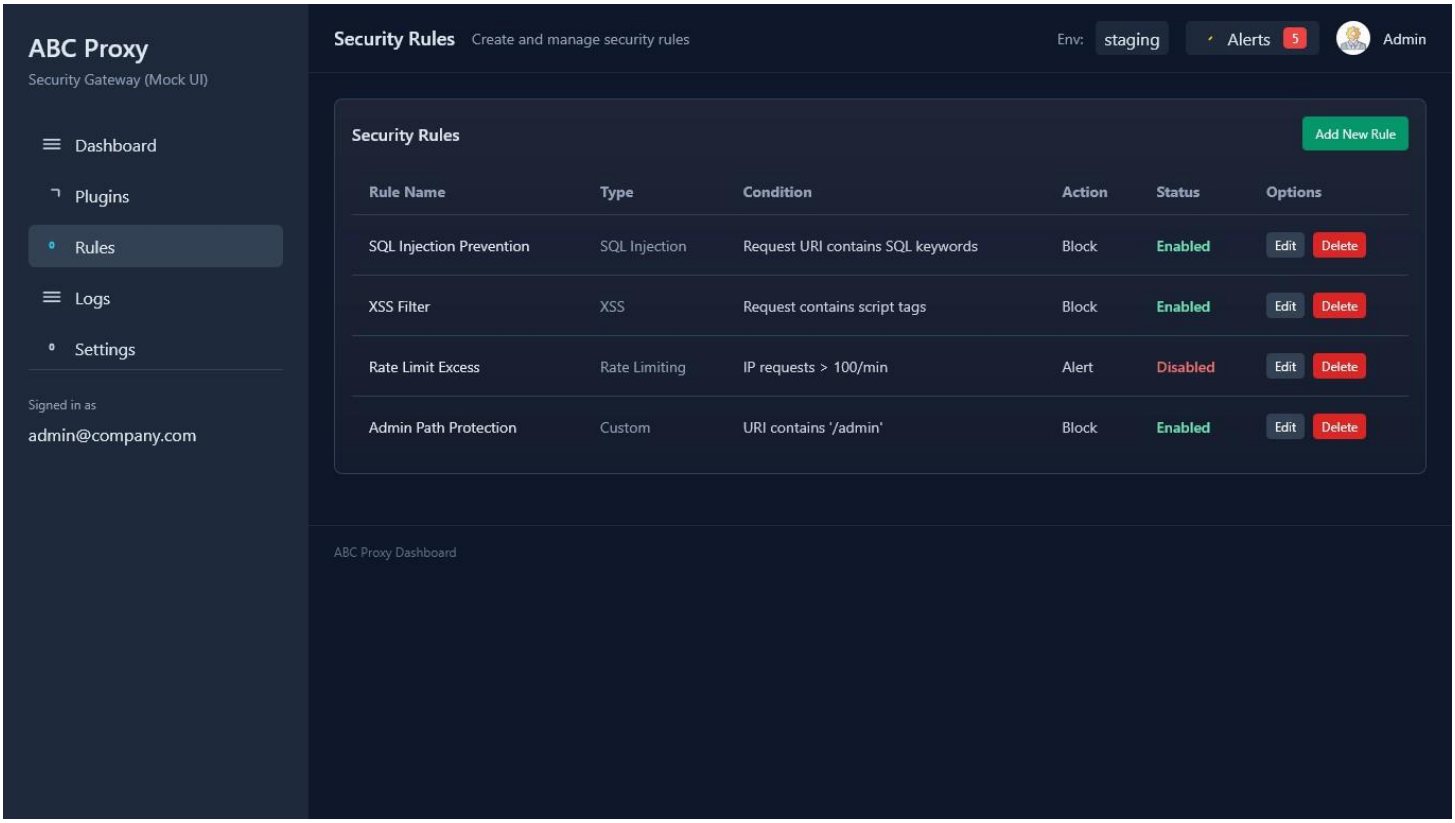


Figure 27: Rules

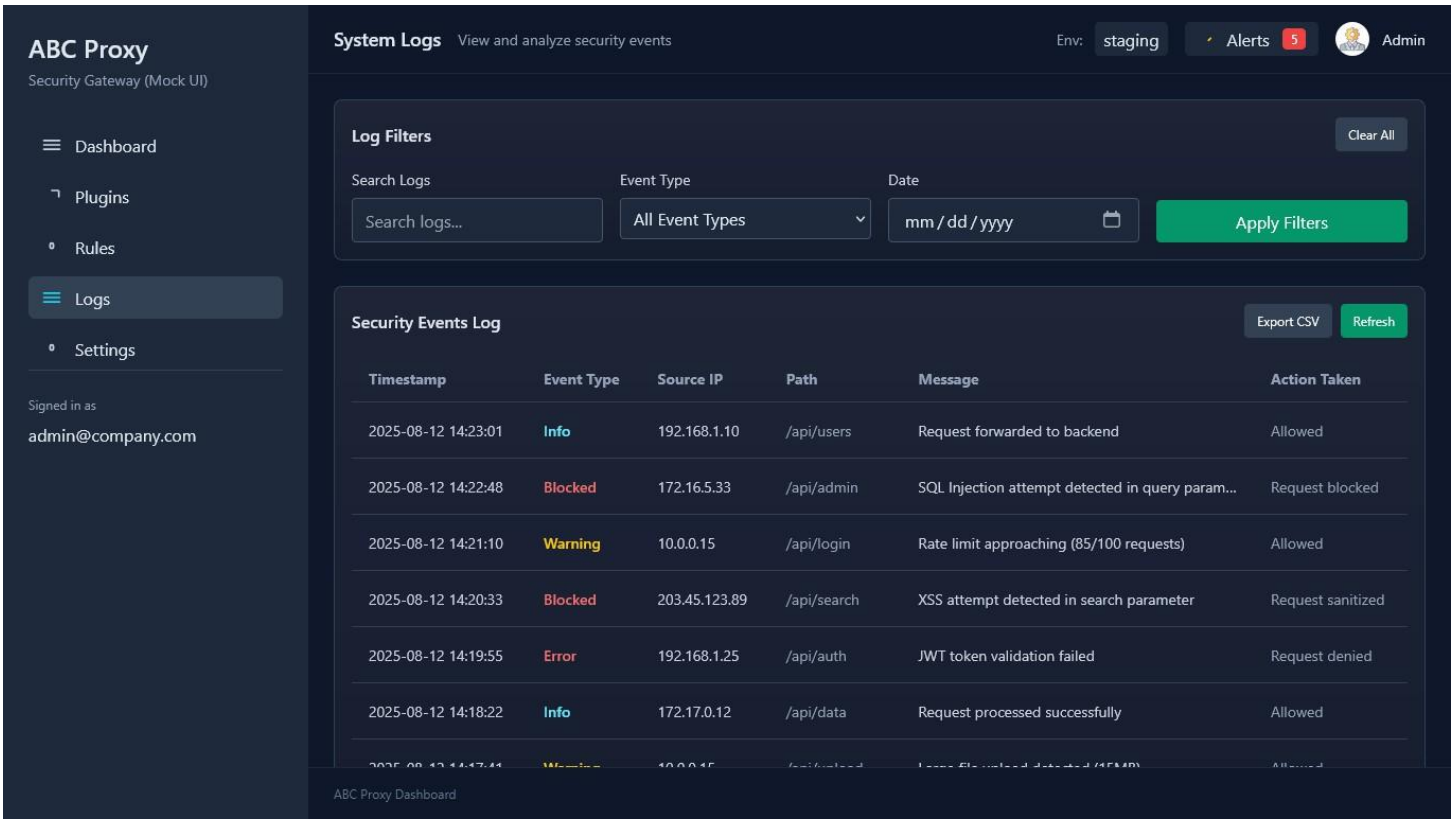


Figure 28: Loggers

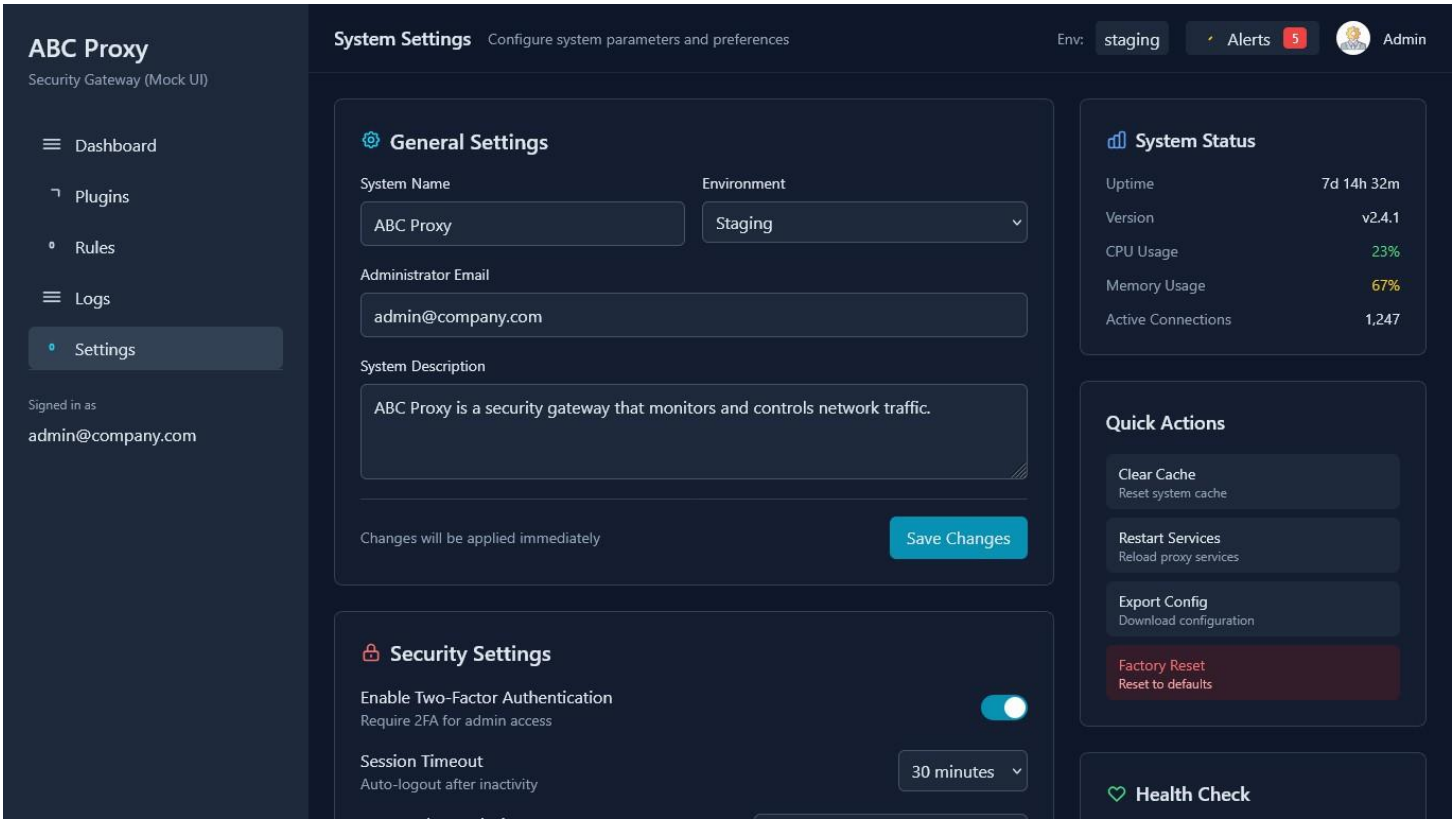


Figure 29: Settings

CHAPTER 7: PROJECT SUMMARY

7.1 Overview

The **ABC Proxy Security Gateway** is a modular, pluggable reverse proxy system designed to intercept, validate, and forward client requests to backend services. It includes a robust plugin architecture for applying security rules such as SQL injection prevention, XSS filtering, and rate limiting. The system is complemented by an **interactive dashboard** for real-time monitoring, rule configuration, and plugin management.

7.2 Achievements

- Designed and implemented a high-performance, pluggable proxy architecture in Golang.
- Built a rule engine to enforce customizable security policies.
- Developed an admin dashboard using React/Vue configuration and monitoring.
- Implemented observability features for tracking requests, responses, and security actions.
- Integrated PostgreSQL database for persistent storage of rules, plugin settings, and admin credentials.
- Enabled Docker and Kubernetes compatibility for scalable deployment.

7.3 Limitations

- Cannot inspect encrypted payloads without performing TLS termination**.**
- Requires manual rule creation; no built-in automated threat intelligence integration.
- Plugin runtime configuration changes require dashboard or API interaction.
- High-performance under heavy loads is dependent on server hardware and optimization.

7.4 Future Enhancements

- AI-based anomaly detection to identify threats beyond predefined rules.
- Multi-language plugin support to allow plugins in Python, Rust, or Node.js alongside Go.
- TLS termination support for inspecting encrypted traffic securely.
- Distributed mode for multi-node scaling across different regions.
- Self-healing and auto-scaling capabilities using Kubernetes advanced features.

7.5 Conclusion

The ABC Proxy Security Gateway successfully delivers on its goal of providing a **flexible, pluggable, and secure** middleware solution for request interception and filtering. With its modular plugin architecture and powerful rule engine, it enables security analysts and administrators to manage web application security dynamically. While there are some limitations—particularly in encrypted traffic handling and automated detection—the foundation is strong for future expansion into AI-driven security and distributed deployments.

CHAPTER 8: REFERENCES

- Coraza, "Coraza WAF Documentation," coraza.io. [link].
- GitHub, "Coraza - OWASP CRS compatible WAF," GitHub. [link].
- knqyf263, "go-plugin: Plugin system over WebAssembly," GitHub. [link].
- Reddit, "How to implement dynamic plugins in Go," Reddit. [link].
- Plugins in Go [link]
- gobyexample - an unofficial example-driven tutorial for Go [link]
- Golang backend Development [link]

Plagiarism Check

212-35-730

ORIGINALITY REPORT

17%	15%	1%	13%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	dspace.daffodilvarsity.edu.bd:8080 Internet Source	7%
2	Submitted to Daffodil International University Student Paper	4%
3	123dok.com Internet Source	2%
4	Submitted to Multimedia University Student Paper	1%
5	Submitted to DISTED College Student Paper	1%
6	Submitted to Southern New Hampshire University - Continuing Education Student Paper	<1%
7	Douglas Jacobson. "Introduction to Network Security", Chapman and Hall/CRC, 2019 Publication	<1%
8	Submitted to Regular Faculty-I Student Paper	<1%
9	Submitted to University of Wales Institute, Cardiff Student Paper	<1%
10	Submitted to University of South Australia Student Paper	<1%
11	journal.umg.ac.id Internet Source	<1%
12	Submitted to University of Central Lancashire Student Paper	<1%

Library Clearance

Account Clearance