Thesis submitted to the

**UNIVERSITY OF MOHAMED BOUDIAF - M'SILA, ALGERIA**


**FACULTY OF MATHEMATICS AND COMPUTER SCIENCE**

**DEPARTMENT OF COMPUTER SCIENCE**

in Partial Fulfillment of the Requirements for the Degree of:

# Master's in Computer Science

**Networks and Information and Communication Technologies**

By

**Maram, AZOUZI**

**Linda, AOUFI**

Entitled

---

# Proxy Server System

---

Under the supervision of

**Makhlouf BENAZI**

Jury Members

| | | |
|---|---|---|
| **Mr. Mohamed Kamel** | University of M'sila | President |
| **Mr. Abdelbasset Barkat** | University of M'sila | Examiner |

June, 2026

# الملخص

في ظل التزايد المستمر للاعتماد على الشبكات والموارد الإلكترونية داخل المؤسسات، أصبحت الحاجة إلى آليات فعالة لمراقبة حركة الويب والتحكم فيها أمرًا ضروريًا لتعزيز أمن المعلومات وضمان الامتثال لسياسات الاستخدام. تهدف هذه المذكرة إلى تصميم وتطوير خادم وكيل لتصفية الويب (Web Filtering Proxy Server) مزود بلوحة إدارة مركزية، وذلك بالاعتماد على لغة البرمجة Python وإطار العمل Flask.

يرتكز النظام المقترح على تقنيات برمجة المقابس (Socket Programming) لاعتراض طلبات HTTP وHTTPS الصادرة من المستخدمين ومعالجتها في الزمن الحقيقي، مع استخراج معلومات الاتصال اللازمة لاتخاذ قرارات التصفية.

ويعتمد النظام على محرك تصفية قائم على مجموعة من القواعد المحددة مسبقًا، حيث تتم مقارنة الطلبات الواردة بقوائم الحظر والكلمات المفتاحية المخزنة، مما يسمح بحظر المواقع غير المصرح بها والحد من الوصول إلى المحتويات المخالفة لسياسات المؤسسة.

ولتحقيق كفاءة التشغيل واستيعاب الاتصالات المتزامنة، تم اعتماد بنية متعددة الخيوط (Multi-threaded Architecture) تضمن معالجة عدد كبير من الطلبات مع الحفاظ على استقرار الأداء. كما يوفر النظام لوحة إدارة تفاعلية تُمكّن المسؤولين من مراقبة حركة المرور الشبكية في الزمن الحقيقي، وإدارة قوائم الحظر من خلال عمليات الإنشاء والإضافة والتعديل والحذف (CRUD)، بالإضافة إلى عرض السجلات والإحصائيات المتعلقة بالنشاط الشبكي.

أظهرت نتائج الاختبارات والتقييمات العملية قدرة النظام على توفير بيئة مركزية وفعالة للتحكم في الوصول إلى الويب ومراقبة الأنشطة الشبكية، مع تحقيق مستوى جيد من الأداء والمرونة، مما يجعله بديلًا مناسبًا للحلول التجارية مرتفعة التكلفة في البيئات التعليمية والمؤسسات الصغيرة والمتوسطة.



**الكلمات المفتاحية:** خادم وكيل، تصفية الويب، التحكم في الوصول إلى الإنترنت، مراقبة حركة المرور الشبكية، برمجة المقابس، أمن الشبكات، Flask.


---


# Abstract

Modern network environments demand robust, granular, and centralized security monitoring mechanisms to regulate web traffic and mitigate organizational data risks. This Master's thesis presents the design and implementation of a centralized web filtering proxy server integrated with an interactive management dashboard developed using Python and the Flask framework. The core system leverages low-level socket programming to intercept outbound client HTTP and HTTPS requests, extracting hostnames and protocol layers in real time. Enforced by a custom-built rule matching mechanism, the filtering engine cross-references these connection streams against flat-file configurations to block restricted domains and optimize content compliance via text-based keyword matching. Concurrently, a multi-threaded archi-

tecture ensures fluid bidirectional socket relays and structural HTTPS tunneling via protocol-transparent handshakes. To eliminate command-line complexities for system administrators, the integrated Flask dashboard provides a centralized interface for real-time traffic auditing, interactive CRUD management of blacklists, and automated activity log visualization. Empirical validation shows that the prototype establishes a responsive, lightweight, and single-pane-of-glass solution for boundary network access control without commercial infrastructure overhead.

**Keywords:**Web Filtering Proxy Server, Content Filtering, Internet Access Control, Network Traffic Monitoring, Socket Programming, Flask Dashboard, Network Security.

# Dedication

*"First and foremost, I dedicate this work to my beloved parents. Their unconditional love, endless sacrifices, wise guidance, and constant prayers have been the driving force behind every achievement in my life.*

*I also dedicate this accomplishment to my dear brothers and sisters, whose unwavering support, encouragement, and belief in me have always been a source of strength and motivation.*

*May this work be a small expression of my gratitude and appreciation for everything they have done for me.*

*With love and deepest gratitude."*

***Maram AZOUZI***

***" In the name of ALLLAH ,the most gracious, the most merciful.***

*I dedicate this work to:*

*My beloved parents, whose unconditional love, sacrifices, guidance, and unwavering support have been the cornerstone of my success.*

*To my dear sister and brothers, thank you for your constant encouragement, understanding, and support throughout my academic journey.*

*To my nephews and nieces, whose joy and affection have always been a source of inspiration and happiness.*

*All my friends and family.*

*With deepest gratitude and sincere appreciation"*

***Linda AOUFI***

# Acknowledgements

We are deeply indebted to our thesis supervisor **Makhlouf BENAZI** whose steadfast support and inspiration made this project a success. In a very special way, we thank them for their continuous guidance and encouragement throughout this challenging study.

Special thanks go to our friends and families who endured the hectic moments and supported us during the course of the research.

We would like to express our sincere gratitude to the University of M'Sila, especially the Faculty of Mathematics and Computer Science, for providing us with a supportive academic environment. We also extend our heartfelt thanks to all our professors for their guidance, dedication, and valuable contributions to our education and the successful completion of this work.

# Contents

# List of Tables

# List of Figures

# General Introduction

The field of network security and traffic management has undergone remarkable transformation in recent years, becoming a cornerstone for boundary defense and content regulation across organizational domains. As data complexity and corporate network volume continue to grow exponentially, the need for robust, dynamic, and centralized web filtering frameworks has never been more critical. This research addresses fundamental challenges in network socket interception and request compliance while proposing an innovative, centralized proxy architecture for real-world administrative applications.

The key research aspects explored in this work span the current state of network proxy boundaries in access control and perimeter security, the critical challenges associated with real-time multi-threaded traffic parsing, and the emerging needs for centralized administrative auditing environments capable of meeting modern organizational demands.

Among the core problems identified, three stand out as particularly significant: the high computational latency encountered during deep payload keyword checking, the complexity of command-line interface rule configurations for network administrators, and the lack of unified logging platforms that bridge low-level packet data with modern web interfaces. These challenges collectively hinder the deployment of effective and manageable proxy solutions in real-world settings.

To address these issues, this project pursues three primary objectives. The first is the development of a lightweight, multi-threaded socket-based web filtering proxy architecture using Python. The second is the implementation of a rule-based content filtering engine targeting domains and keyword combinations. The third is the design of an interactive, centralized dashboard using the Flask framework for real-time event visualization and log monitoring.

The methodological framework adopted in this research follows a structured progression beginning with the theoretical foundation development of proxy topologies and application-layer protocols, followed by prototype design and socket synchronization testing under concurrent client simulation, and concluding with a comprehensive functional verification and performance evaluation process conducted via simulated web traffic.

Recent cybersecurity studies demonstrate that over 72% of corporate environments struggle with unauthorized data exposures and web-based security vulnerabilities, particularly in enterprise private networks where preliminary security audits reveal high vulnerability rates during unmonitored user browsing. These empirical findings underscore the urgent need for the centralized perimeter access control and auditing solutions proposed and validated in this work.

The remainder of this thesis is organized as follows: Chapter 1 presents the theoretical

background on proxy servers and related concepts; Chapter 2 covers system design; and Chapter 3 details the implementation, testing, and evaluation of the proposed system.

# Chapter 1: Introduction to Proxy Servers

## 1.1 Introduction to Proxy Servers

A proxy server is an intermediary system that sits between client devices and destination servers, acting as a gateway for requests [1] [2]. The term 'proxy' derives from the concept of a representative or substitute in networking terms, a proxy acts on behalf of clients when communicating with servers, or vice versa.

In essence, when a client makes a request to access a resource on the internet, the request first goes to the proxy server. The proxy then forwards this request to the destination server, retrieves the response, and sends it back to the client. This intermediary position enables proxy servers to provide various services including caching, filtering, load balancing, and security enhancement [3] [4].

Proxy servers have evolved from simple forwarding mechanisms to sophisticated systems that handle millions of requests daily, implementing complex routing logic, content transformation, and security policies [5] [6]. They form a critical component of modern internet infrastructure, deployed across enterprises, telecommunications networks, and cloud platforms worldwide [7] [8].

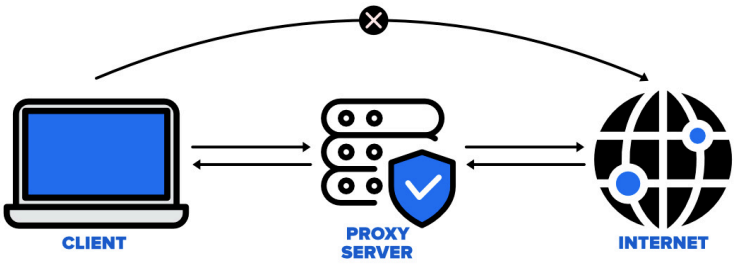


Figure 1.1: Proxy Servers

## 1.2 How Proxy Servers Work

The operational mechanism of a proxy server follows a request-response pattern with several key stages :

### 1.2.1 Request Interception

When a client application is configured to use a proxy, all network requests are directed to the proxy server instead of going directly to the destination [9]. This configuration can be explicit (manually specified in browser or system settings) or transparent (automatically redirected by network infrastructure) [3] [4].

### 1.2.2 Request Processing

Upon receiving a request, the proxy server performs several operations : validating the request format, checking access control policies, examining cache for previously fetched resources, and applying any content filtering rules [3]. If the requested resource exists in cache and is still valid, the proxy returns it immediately, bypassing the origin server entirely [10] [11].

### 1.2.3 Request Forwarding

If the resource is not cached or requires validation, the proxy establishes a connection with the destination server [12] [1]. It may modify request headers to add information about the proxy chain, remove sensitive client details, or inject additional parameters [4] [5]. The proxy maintains state information to correlate the eventual response with the original client request [2] [3].

### 1.2.4 Response Handling

When the response arrives from the origin server, the proxy processes it according to configured policies [3] [13]. This may include caching the response for future requests, scanning content for malware or policy violations, compressing data to reduce bandwidth, or modifying headers [11] [14] [15]. The processed response is then forwarded to the client [1] [2].

## 1.3 Proxy Protocols and Standards

Several protocols and standards govern proxy server operations, each designed for specific use cases and network architectures.

### 1.3.1 HTTP/HTTPS Proxy

The most common proxy type, operating at the application layer (Layer 7) of the OSI model [1]. HTTP proxies understand and can modify HTTP traffic, enabling features like content

filtering and caching [3] [4]. For HTTPS traffic, proxies typically use the CONNECT method to establish a tunnel, allowing encrypted data to pass through without inspection [12] [16].

### 1.3.2 SOCKS Protocol

SOCKS (Socket Secure) operates at a lower level than HTTP proxies, working at the session layer (Layer 5) [17] [2]. SOCKS5, the current version, supports various authentication methods and can handle any type of traffic, not just HTTP . This protocol is particularly useful for applications requiring UDP support or those that don't use HTTP [17] [1].

### 1.3.3 PAC (Proxy Auto-Configuration)

PAC files contain JavaScript functions that determine whether requests should go through a proxy and which proxy to use [9]. This standard, originally developed by Netscape, enables dynamic proxy selection based on URL patterns, destination domains, or network conditions [3]. Organizations use PAC files to implement sophisticated routing policies without manual client configuration [9] [18].

### 1.3.4 WPAD (Web Proxy Auto-Discovery)

WPAD automates the discovery of PAC files on a network using DHCP or DNS queries [3]. This protocol allows clients to automatically configure proxy settings without manual intervention, simplifying deployment in large organizations [18]. However, security concerns have led to decreased adoption in recent years [15] [19].

## 1.4 Types of Proxy Servers

### 1.4.1 Forward Proxy

A forward proxy sits between client devices and the internet, forwarding client requests to external servers [1]. Organizations deploy forward proxies to control internet access, enforce security policies, cache frequently accessed content, and hide internal IP addresses [18] [20]. Users must explicitly configure their applications to use a forward proxy [9] [3].

### 1.4.2 Reverse Proxy

Reverse proxies sit in front of web servers, intercepting requests from the internet before they reach the origin servers . They provide load balancing across multiple servers, SSL termination to offload encryption overhead, caching to reduce server load, and protection

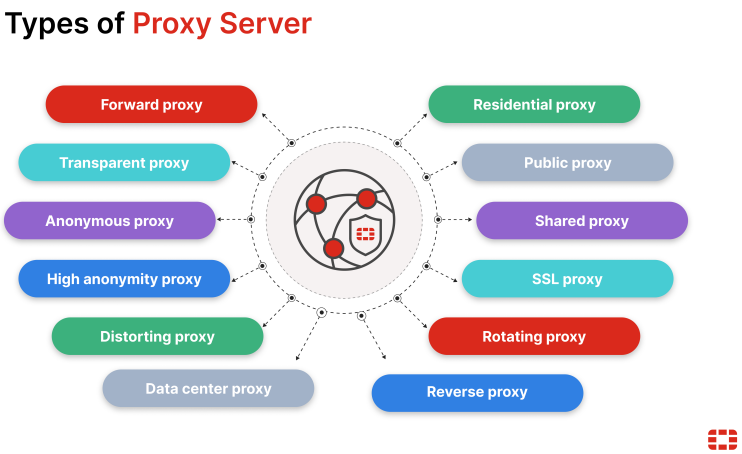


Figure 1.2: Types of Proxy Servers

against attacks. Popular reverse proxy software includes NGINX, HAProxy, and Apache Traffic Server [4] [5] [7] [13].

### 1.4.3 Transparent Proxy

Transparent proxies intercept network traffic without requiring client configuration [3]. Implemented at the network layer through routing or firewall rules, they are invisible to end users [2]. Internet Service Providers often use transparent proxies for caching and traffic management, though privacy concerns have limited their adoption [19].

### 1.4.4 Anonymous and Elite Proxy

Anonymous proxies hide the client's IP address but identify themselves as proxies in HTTP headers . Elite (or high-anonymity) proxies go further by neither revealing the client IP nor identifying as a proxy . These types are primarily used for privacy protection, bypassing geo-restrictions, and web scraping operations [21] [19].

## 1.5 Proxy Server Architecture and Components

Modern proxy servers comprise several architectural layers and components working in concert :

### 1.5.1 Connection Handler

The connection handler manages incoming client connections, typically using non-blocking I/O or event-driven architectures to handle thousands of concurrent connections efficiently

[4] [6]. It implements connection pooling, keepalive mechanisms, and timeout management to optimize resource utilization [5] [13].

### 1.5.2 Request Router

This component analyzes incoming requests and determines their routing based on URL patterns, headers, or custom rules [6]. Advanced routers implement content-based routing, geographic load balancing, and failover mechanisms [8] [22]. They maintain routing tables and health check results to make intelligent forwarding decisions [4] [5].

### 1.5.3 Cache Manager

The cache manager stores frequently accessed resources in memory or on disk, implementing cache eviction policies like LRU (Least Recently Used) or LFU (Least Frequently Used) [3] [11]. It validates cached content using ETags, Last-Modified headers, and Cache-Control directives [12] [11]. Sophisticated caching systems support cache hierarchies and distributed caching across multiple nodes [3] [8].

### 1.5.4 Security Module

Security components enforce authentication, authorization, and access control policies [20]. They implement SSL/TLS termination, certificate validation, and may include intrusion detection capabilities [14] [7]. Modern security modules integrate with threat intelligence feeds to block malicious sources and implement rate limiting to prevent abuse [7] [15].

### 1.5.5 Logging and Monitoring

Comprehensive logging captures request details, response codes, bandwidth usage, and error conditions [4] [5]. Monitoring systems track performance metrics like response times, cache hit rates, and connection counts [6] [8]. This data feeds into analytics platforms for capacity planning and security analysis [7] [18].

## 1.6 Uses of Proxy Servers

### 1.6.1 Security and Access Control

Organizations deploy proxies as a security perimeter, filtering malicious content, blocking access to dangerous websites, and preventing data exfiltration [14] [20]. Proxies inspect traffic for malware, enforce acceptable use policies, and provide detailed audit logs for compliance requirements [7] [18] [15].

### 1.6.2 Performance Optimization

Caching proxies significantly improve response times and reduce bandwidth consumption by serving frequently accessed content from local storage [3] [10] [11]. They implement compression to reduce data transfer volumes and connection multiplexing to optimize network utilization [12] [23] [24]. Content Delivery Networks (CDNs) leverage proxy architectures extensively [7] [8].

### 1.6.3 Load Balancing

Reverse proxies distribute incoming requests across multiple backend servers, preventing any single server from becoming overwhelmed [13]. They implement various load balancing algorithms including round-robin, least connections, and weighted distribution [6]. Health checks ensure traffic only routes to healthy servers [4] [5].

### 1.6.4 Privacy and Anonymity

Privacy-focused proxies hide user IP addresses and browsing patterns from destination servers [19]. They enable access to geo-restricted content and protect against tracking . Anonymous proxy networks like Tor use multiple proxy layers to provide strong anonymity guarantees [21].

### 1.6.5 Content Filtering

Educational institutions and enterprises use proxies to enforce content policies, blocking inappropriate material and restricting access during work hours [15]. Advanced systems implement category-based filtering, SSL inspection for encrypted traffic, and integrate with user directories for policy enforcement [7] [18] [20].

## 1.7 Advantages and Limitations

### 1.7.1 Advantages

Proxy servers offer numerous benefits: enhanced security through centralized traffic inspection and policy enforcement [14] [15] [20]; improved performance via caching and compression [3] [10]; bandwidth savings by reducing redundant data transfers [11] [8]; centralized logging and monitoring for compliance and troubleshooting [4]; and the ability to implement sophisticated routing and load balancing strategies [5] [6]. They provide a single point for implementing organization-wide policies and enable gradual migration between infrastructure changes [18] [25].

### 1.7.2 Limitations

However, proxies introduce certain limitations: they add latency to request processing, creating a potential bottleneck if not properly scaled [1] [2]; they represent a single point of failure requiring redundancy and high availability measures [5] [25]; SSL/TLS inspection raises privacy concerns and adds complexity [19] [26]; configuration errors can cause widespread connectivity issues [3]; and they increase infrastructure costs and operational complexity [18] [27]. Encrypted traffic presents challenges for content inspection, and sophisticated applications may not work correctly through certain proxy types [12] [23] [26].

## 1.8 Proxy vs Related Technologies

### 1.8.1 Proxy vs VPN

While both proxies and VPNs intermediate network traffic, they differ fundamentally [2]. VPNs create encrypted tunnels at the network layer, routing all traffic from a device through the tunnel regardless of application . Proxies typically operate at the application layer, handling traffic from specific applications configured to use them [12] [17] [1]. VPNs provide stronger security through encryption but add overhead, while proxies offer more granular control and caching capabilities [3] [11] [20].

### 1.8.2 Proxy vs Firewall

Firewalls operate at the network layer, making decisions based on IP addresses, ports, and protocols [2]. They block or allow traffic without understanding application-level content . Proxies understand protocols like HTTP, enabling content-aware decisions and modifications [12] [3] [15]. Modern deployments often use both: firewalls for network-level security and proxies for application-level control [14] [20].

### 1.8.3 Proxy vs NAT

Network Address Translation (NAT) maps internal IP addresses to external ones, enabling multiple devices to share a single public IP [2]. While NAT provides basic IP hiding similar to proxies, it operates transparently at the network layer without application awareness [1] [2]. Proxies offer richer features including caching, content filtering, and protocol-specific optimizations that NAT cannot provide [3] [11].

### 1.8.4 Proxy vs API Gateway

API gateways are specialized proxies designed for managing API traffic . They provide features like rate limiting, authentication, request transformation, and API versioning . While traditional proxies handle general HTTP traffic, API gateways understand API semantics and implement API-specific policies [16] [22]. Modern API gateways often build upon proxy servers, adding higher-level API management capabilities [6] [25].

## 1.9 Real-World Applications

### 1.9.1 Enterprise Networks

Large organizations deploy forward proxies to control employee internet access, enforce security policies, and reduce bandwidth costs through caching .Companies like Zscaler and Cisco Umbrella provide cloud-based proxy services that protect remote workers . Enterprise proxies integrate with Active Directory for user-based policies and generate detailed reports for compliance auditing [18] [20] [27].

### 1.9.2 Content Delivery Networks

CDNs like Cloudflare, Akamai, and Amazon CloudFront operate massive distributed networks of reverse proxies . These proxies cache content at edge locations worldwide, serving users from geographically nearby servers to reduce latency . They also provide DDoS protection, SSL termination, and web application firewall capabilities, protecting origin servers from attacks and overload [7] [8] [15].

### 1.9.3 Web Scraping and Testing

Developers use proxy pools to distribute web scraping requests across multiple IP addresses, avoiding rate limits and blocks [17]. Testing frameworks leverage proxies to simulate traffic from different geographic locations and network conditions [3] [4]. Services like BrightData and Oxylabs provide residential and datacenter proxy networks for these purposes.

### 1.9.4 Microservices Architecture

Modern microservices deployments use service mesh proxies like Envoy and Linkerd to handle inter-service communication . These proxies provide service discovery, load balancing, circuit breaking, and observability . They implement sophisticated traffic management policies, enabling canary deployments, A/B testing, and gradual rollouts without application code changes [6] [25] [28] [22].

### 1.9.5 Privacy Networks

Privacy-focused networks like Tor route traffic through multiple proxy layers operated by volunteers worldwide . Each proxy layer encrypts the traffic, and no single proxy knows both the source and destination . This onion routing provides strong anonymity guarantees, enabling secure communication in surveillance-heavy environments and protecting whistleblowers and journalists [21] [19].

## 1.10 Future Trends in Proxy Technology

### 1.10.1 HTTP/3 and QUIC Protocol Support

The adoption of HTTP/3 and its underlying QUIC protocol presents both challenges and opportunities for proxy servers . QUIC's encryption and multiplexing at the transport layer require proxies to evolve beyond traditional TCP-based architectures . Future proxies will need to support QUIC connection migration and integrate with the protocol's built-in encryption while maintaining visibility and control capabilities [23] [24] [26].

### 1.10.2 AI-Powered Traffic Analysis

Machine learning models are increasingly integrated into proxy servers for intelligent caching decisions, anomaly detection, and threat prevention [8]. AI systems analyze traffic patterns to predict popular content, optimize routing decisions, and identify zero-day attacks [14]. Future proxies will leverage AI to automatically adapt policies based on learned user behavior and evolving threat landscapes [7] [15].

### 1.10.3 Edge Computing Integration

As edge computing grows, proxy functionality is being pushed closer to end users . Edge proxies will execute application logic, process data locally, and make autonomous decisions without contacting centralized servers . This trend enables ultra-low latency applications, reduces bandwidth costs, and improves resilience against network failures [8] [29].

### 1.10.4 Zero Trust Architecture

Zero Trust security models are transforming proxy deployment patterns. Rather than traditional perimeter-based security, Zero Trust proxies continuously verify every request regardless of origin . They implement micro-segmentation, least-privilege access, and continuous authentication . Cloud-based Zero Trust proxies enable secure access to applications without traditional VPNs [27] [20].

### 1.10.5 Privacy-Enhancing Technologies

Growing privacy concerns drive development of privacy-preserving proxy technologies . Encrypted ClientHello (ECH) prevents SNI-based surveillance, while oblivious DNS over HTTPS hides DNS queries from proxies . Future proxies will balance security and monitoring requirements with user privacy through techniques like differential privacy and homomorphic encryption [19] [26].

### 1.10.6 Containerization and Kubernetes Native

Proxy servers are evolving to be cloud-native, designed specifically for containerized environments . Kubernetes-native proxies like Envoy and NGINX Ingress Controller integrate deeply with orchestration platforms, automatically adapting to dynamic service scaling . The future sees proxies as lightweight sidecars deployed alongside applications, providing per-service policies and observability [6] [25] [28] [22] [29].

### 1.10.7 WebAssembly Extensibility

WebAssembly (Wasm) is emerging as a standard for extending proxy functionality . Proxies like Envoy support Wasm filters, allowing developers to write custom logic in languages like Rust or Go that runs safely within the proxy . This enables rapid deployment of custom features without recompiling proxy software, accelerating innovation and customization [6] [30] [29].

## 1.11 Conclusion

Proxy servers have evolved from simple forwarding mechanisms into sophisticated systems that form the backbone of modern internet infrastructure . Their ability to provide security, enhance performance, and enable complex routing policies makes them indispensable in contemporary network architectures .

As networking technologies continue to evolve, proxy servers adapt to meet new challenges [23]. The integration of artificial intelligence, adoption of modern protocols like QUIC, and alignment with Zero Trust security models demonstrate the ongoing relevance and innovation in proxy technology . Whether deployed in enterprise networks, content delivery systems, or microservices architectures, proxy servers remain essential tools for managing, securing, and optimizing network traffic [18] [22].

The future of proxy technology lies in balancing increasingly stringent security and privacy requirements with the need for visibility and control [19]. As encrypted traffic becomes ubiquitous and edge computing proliferates, proxy servers will continue to evolve, finding

new ways to provide value while respecting user privacy and adapting to changing network paradigms .

Understanding proxy server technology its protocols, architectures, applications, and future directions provides essential knowledge for network engineers, security professionals, and anyone involved in building and maintaining modern internet infrastructure . As the digital landscape grows more complex, the role of proxy servers in managing this complexity only becomes more critical [7] [8] [1] [2] [20] [26].

# Chapter 2: System Design

## 2.1 Introduction

This chapter presents the overall design of the proposed system and explains how the identified requirements are transformed into technical models that guide the development process. The design phase is an essential step, as it helps clarify system functionalities and defines the relationships between different components.

In software development, system design plays an important role in connecting the analysis phase to the implementation phase. It provides a clear view of how the system is structured, how its components interact, and how data moves within the application. This makes the development process more organized and reduces the risk of errors.

In this chapter, we focus on the use of Unified Modeling Language (UML) diagrams to represent the system. These diagrams are used to describe both the structure and behavior of the system in a visual way. They also help improve communication between stakeholders and provide a better understanding of the system before implementation.

## 2.2 Overview of UML

Unified Modeling Language (UML) is a standardized modeling language used in software engineering to represent, visualize, and document the structure and behavior of complex systems. It provides a set of graphical notations that enable developers, analysts, and system designers to describe system components, their relationships, and interactions in a clear and organized manner.

UML supports both the analysis and design phases of the software development life-cycle by offering multiple types of diagrams, such as use case, class, sequence, and state diagrams. These diagrams help in modeling both the static aspects (structure) and dynamic aspects (behavior) of a system.

Moreover, UML establishes a common communication language among stakeholders, including developers, analysts, and project managers, facilitating better understanding and collaboration throughout the project. Its flexibility allows it to be applied to various types of systems, not limited to software, and across different programming languages and platforms.

## 2.3 Use Case Diagram

### 2.3.1 Definition

A Use Case Diagram is a type of UML diagram used to represent the functional requirements of a system from the user's perspective. It illustrates how different users, known as actors, interact with the system through various use cases, which represent the services or functionalities provided by the system.

This diagram helps identify the main interactions between external users and the system, making it easier to understand what the system is expected to do without focusing on internal implementation details. It is commonly used during the analysis phase to define system boundaries and clarify user needs.

Additionally, Use Case Diagrams improve communication between stakeholders by providing a simple and visual representation of system functionalities. They serve as a foundation for further design and development activities.

### 2.3.2 Identification of Actors

The proposed system involves two main actors whose roles are defined based on their level of access and interaction with the system.

Table 2.1: System Actors Description.

| Actor | Type | Description |
|---|---|---|
| **Admin** | Primary | The privileged administrator responsible for the full management of the web filtering proxy. The Admin manages blocked sites and keywords, monitors system logs, tests the filtering engine, and customises the display mode. All administrative actions require prior authentication. |
| **User** | Primary | A standard end-user who accesses the internet through the proxy. The User's browsing requests are automatically subject to the content filtering rules enforced by the system. |

### 2.3.3 Use Cases Description

To provide a clear and structured understanding of the system's behavior, the identified use cases are detailed through a combination of structured templates for core functionalities and focused operational descriptions for auxiliary operations.

**Core Use Cases Descriptions**

Table 2.2: Login Use Case Description.

| **Use Case Name** | Login |
|---|---|
| **Actor** | Admin |
| **Description** | Allows the Admin to authenticate and gain access to the system's administrative interface. |
| **Pre-condition** | The Admin is not yet authenticated. |
| **Main Flow** | Admin enters credentials → System validates → Access granted. |
| **Alternative Flow** | Invalid credentials → System displays error message → Admin retries. |
| **Post-condition** | Admin is authenticated and redirected to the dashboard. |

Table 2.3: Monitor Logs Use Case Description.

| **Use Case Name** | Monitor Logs |
|---|---|
| **Actor** | Admin |
| **Description** | Allows the Admin to supervise system activity through three sub-functions: Consult Logs (view/search/clear), Export Logs, and identification of connected users through their IP addresses recorded in the activity log. |
| **Pre-condition** | Admin is authenticated. |
| **Main Flow** | Admin navigates to the logs section → Views, exports records, or identifies connected clients via logged IP addresses. |
| **Alternative Flow** | No logs available → System displays an empty state message. |
| **Post-condition** | Logs are consulted, exported, or cleared as requested. |

Table 2.4: Browsing Use Case Description.

| **Use Case Name** | Browsing |
|---|---|
| **Actor** | User |
| **Description** | Allows the User to navigate the web through the proxy. All requests are automatically intercepted and filtered against the blocked sites and keywords lists before being forwarded or denied. |
| **Pre-condition** | User is connected to the network and routes traffic through the proxy. |
| **Main Flow** | User requests a URL → Proxy checks domain blocklist → Proxy checks keyword blocklist → Page is delivered. |
| **Alternative Flow** | URL or content matches a blocked entry → Access denied page is returned to the User. |
| **Post-condition** | Requested page is displayed or access is blocked. |

**Secondary Use Cases Operational Descriptions**

To maintain a concise and professional documentation layout, the secondary and straightforward administrative behaviors are formalized through focused contextual descriptions rather than full textual templates:

- **Management Operations (CRUD):** The operations including *Add*, *Search*, *Modify/Delete*, and *Import .txt* embedded within both *Manage Blocked Sites* and *Manage Keywords* follow identical backend behaviors. The authenticated Admin interacts with the dashboard interfaces to issue data updates. The Flask *Backend* validates the format constraint (e.g., domain structure syntax) before modifying the configuration stores, followed by an asynchronous data push to guarantee immediate synchronization with the active core Python filtering architecture.

- **Test Filter (Simulator):** This operational flow serves diagnostic and evaluation purposes. It permits the authenticated Admin to manually input an arbitrary URL or string payload directly into a dashboard simulation field. The system processes the target query through the inner validation routines without modifying the operational system parameters, intercepting live sessions, or appending records to the system history log.

- **Choose Display Mode:** This front-end feature optimizes the administrative user experience. The Admin triggers the theme toggle button to switch between light and dark visualization layouts. The execution alters the CSS properties exclusively via browser client-side logic (*localStorage*), removing any overhead or communication requirements with the Python web server.

- **Logout Use Case:** This process guarantees secure context termination. When the Admin triggers the session termination, the system immediately clears active cookies, invalidates server-side storage structures, and forces a browser context redirection back to the *Login* view.

### 2.3.4 Use Case Diagram of Our System

The following figure presents the use case diagram of the proposed web filtering proxy system, illustrating all interactions between the system's actors and its core functionalities.

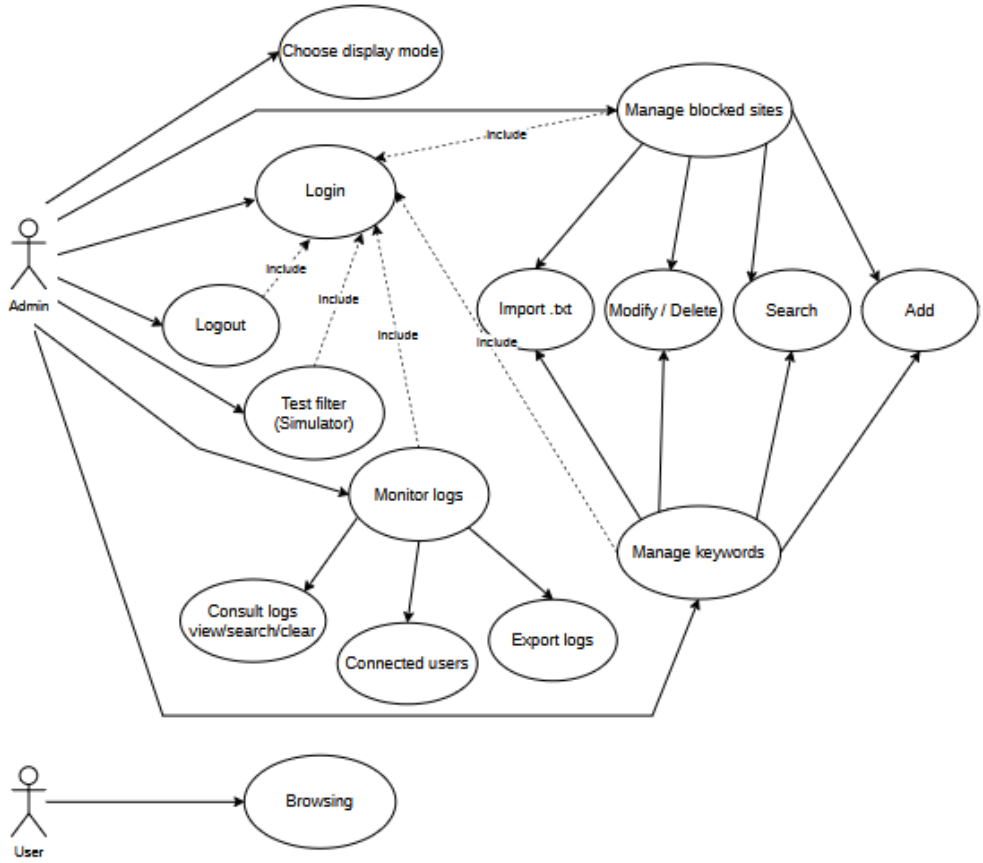


Figure 2.1: System Use Case Diagram.

The diagram involves two actors: the **Admin** and the **User**. The `<<include>>` relationships model mandatory dependencies, meaning that administrative actions such as Manage Blocked Sites, Manage Keywords, Monitor Logs, and Test Filter all require a prior successful Login. The **User** actor is associated solely with the Browsing use case, where all outgoing requests are subject to the filtering rules enforced by the proxy.

## 2.4 Class Diagram

### 2.4.1 Definition

A Class Diagram is one of the main diagrams in UML used to represent the static structure of a system. It describes the system by showing its main classes, along with their attributes, methods, and the relationships that exist between them.

This diagram is widely used during the design phase, as it helps organize the system into logical components and provides a clear view of how data and functionalities are structured. In object oriented development, classes are considered the basic building blocks of the system, where each class groups related data and behaviors.

In addition, the relationships between classes such as association, inheritance, and composition define how different parts of the system interact and collaborate. Therefore, the Class Diagram serves as a foundation for implementation by guiding developers in structuring the code in a consistent and maintainable way.

### 2.4.2 Our Class Diagram

The class diagram of our web filtering proxy system is presented in the figure below. It models all system entities, their attributes, their methods, and their inter-class relationships.

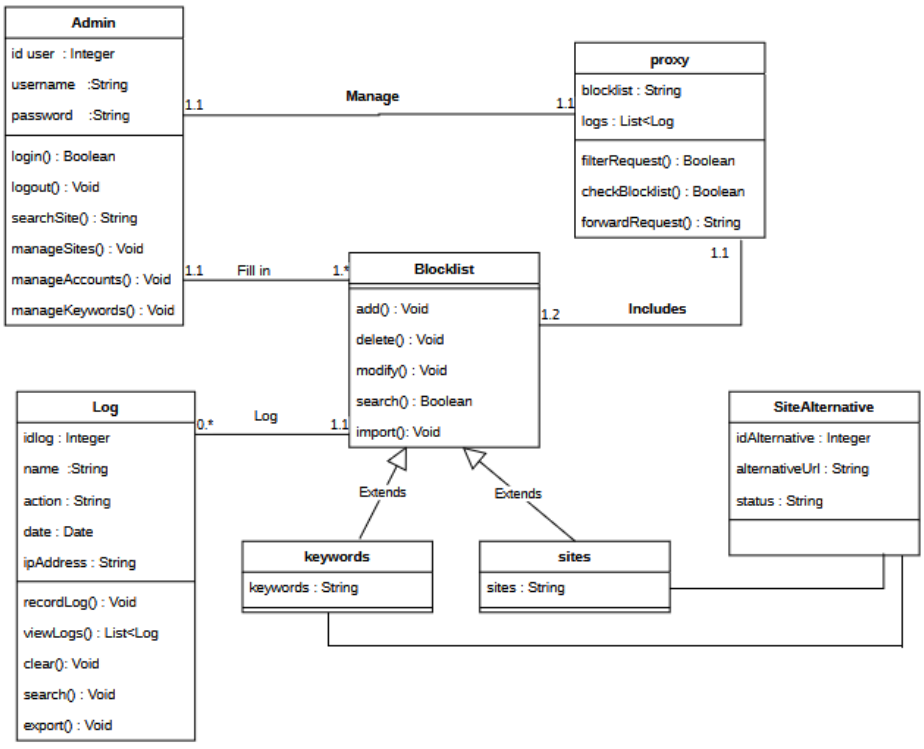


Figure 2.2: System Class Diagram.

### 2.4.3 Description of Classes and Relationships

The class diagram of the proposed system is structured around two central entities: the Admin and the Proxy. The Admin is responsible for the overall governance of the system and is linked to the Proxy through a bidirectional one-to-one Manage association, reflecting a strict administrative dependency between the two. The Admin further interacts with the Blocklist class via a Fill in relationship, through which blocked content entries are registered and maintained.

The Proxy, acting as the core request filtering component, is connected to the Blocklist through an Includes dependency with a multiplicity of 1..2, indicating that it references two specialised subclasses derived from the Blocklist: Keywords and Sites, each encapsulating a distinct category of restricted content.

In addition, the Proxy maintains a one-to-many association with the Log class, whereby every processed request generates a corresponding audit record, ensuring the full traceability of system activity.

Finally, the Blocklist includes a `SiteAlternative` class in the design model, which represents a planned extension enabling the system to propose a redirection URL as an alternative when access to a requested domain is denied. This feature is identified as a future development direction and is not implemented in the current version of the system.

## 2.5 Sequence Diagram

### 2.5.1 Definition

A Sequence Diagram is a type of UML diagram used to represent the dynamic behavior of a system by illustrating how different objects interact with each other over time. It focuses on the exchange of messages between system components in order to accomplish a specific function or scenario.

This diagram shows the sequence in which interactions occur, starting from an initial action and continuing through the various steps until the process is completed. Each object involved is represented by a lifeline, and the communication between them is illustrated through messages.

Sequence Diagrams are particularly useful during the design phase, as they help in understanding the flow of operations, clarifying the order of events, and identifying how different components collaborate. They also support the detailed analysis of use cases and contribute to improving system organization and performance.

### 2.5.2 Sequence Diagrams of Our System

The following sequence diagrams illustrate the dynamic interactions between the system components for each main functional scenario. Both Admin and User scenarios are covered.

**a. Admin Scenarios:**

**1. Add Domain**

The following diagram models the interaction triggered when the Admin adds a new domain to the blocked sites list. The participants are the Admin, the Graphical Interface, the Backend, and the Blocked Sites data store.

The Admin first requests the domains dashboard, which causes the Graphical Interface to fetch the current blocked sites list from the Backend. Once the dashboard is displayed, the Admin submits an add domain request. The Backend then performs an internal check to verify whether the domain already exists. An **Alt** combined fragment governs the outcome:

- **[Domain exists]:** The Backend returns a *Domain already exists* error, which is propagated back to the Admin via the Interface.
- **[Else]:** The Backend saves the new domain to the Blocked Sites store and returns a *Domain added successfully* confirmation, which is displayed to the Admin.

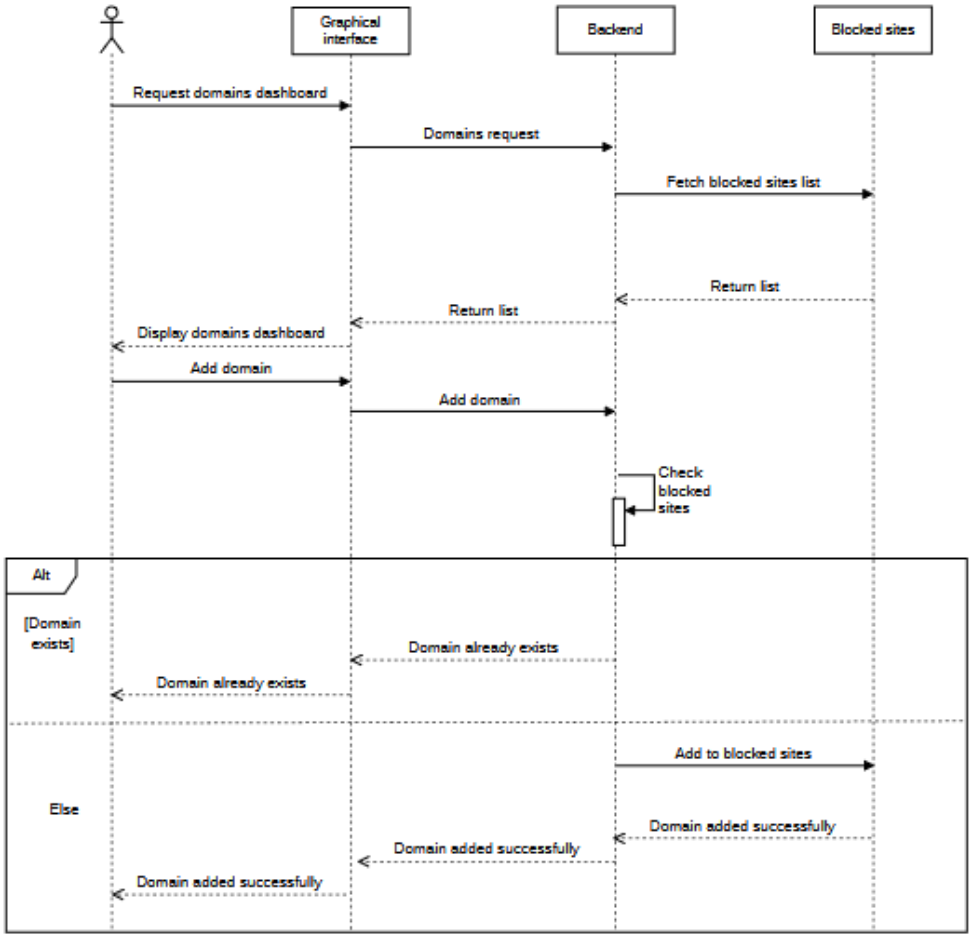


Figure 2.3: Add Domain Sequence Diagram.

**2. Import Domain**

The following diagram illustrates the bulk import process, triggered when the Admin uploads a `.txt` file containing a list of domain names. The participants are the Admin, the Graphical Interface, the Backend, and the Blocked Sites data store.

The Admin selects and uploads a `.txt` file. The Graphical Interface forwards the file payload to the Backend via a `POST /Domain/import` request. The Backend parses the file and extracts the domain list. A **Loop** fragment then iterates over each domain in the list, and for each entry the Backend checks whether the domain already exists:

- **[true --- already exists]:** The domain is skipped and logged.
- **[false --- new domain]:** The domain is saved to the Blocked Sites store.

Once all entries are processed, the Backend returns an execution summary, and the Interface displays an *imported new domains successfully* notification to the Admin.

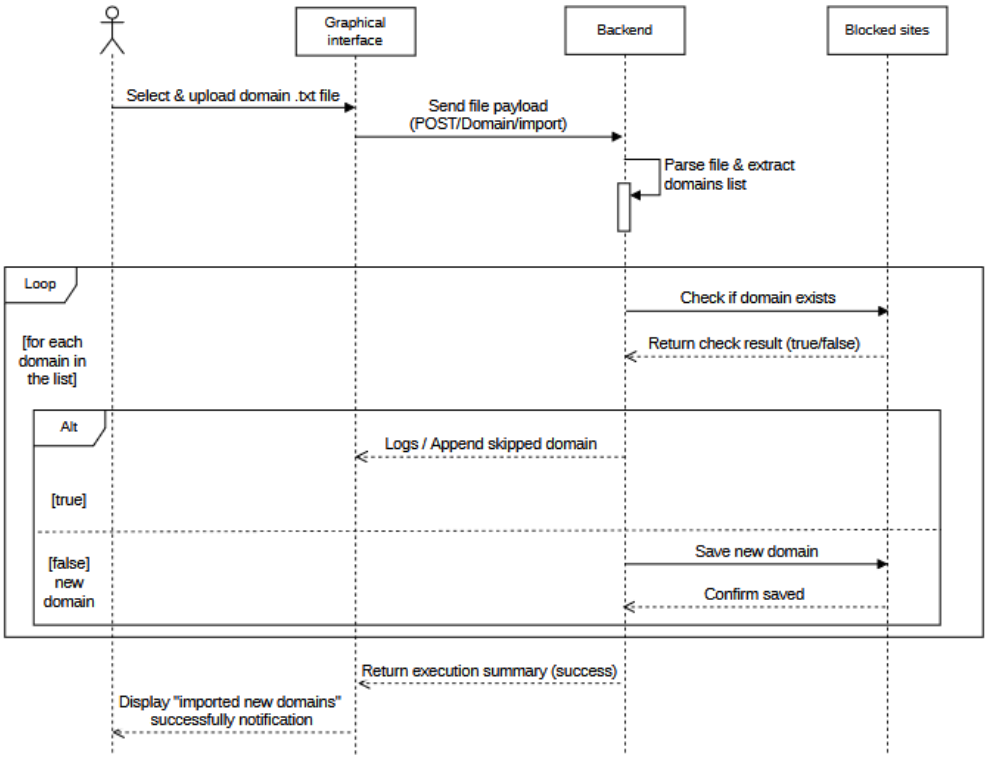


Figure 2.4: Import Domain Sequence Diagram.

**3. Delete Keyword**

The following diagram illustrates the interaction that occurs when the Admin deletes a keyword from the blocked keywords list. The participants are the Admin, the Graphical Interface, the Backend, and the Blocked Keywords data store.

The Admin first requests the keywords dashboard. The Graphical Interface fetches the keywords list from the Backend, which retrieves it from the Blocked Keywords store and returns it for display. The Admin then triggers a delete action with a browser confirmation.

The Graphical Interface sends the delete request to the Backend, which issues a *Delete keyword* command to the Blocked Keywords store. Finally, the store confirms with a *Deleted* acknowledgement, and a *Keyword deleted successfully* message is passed back to the Admin.

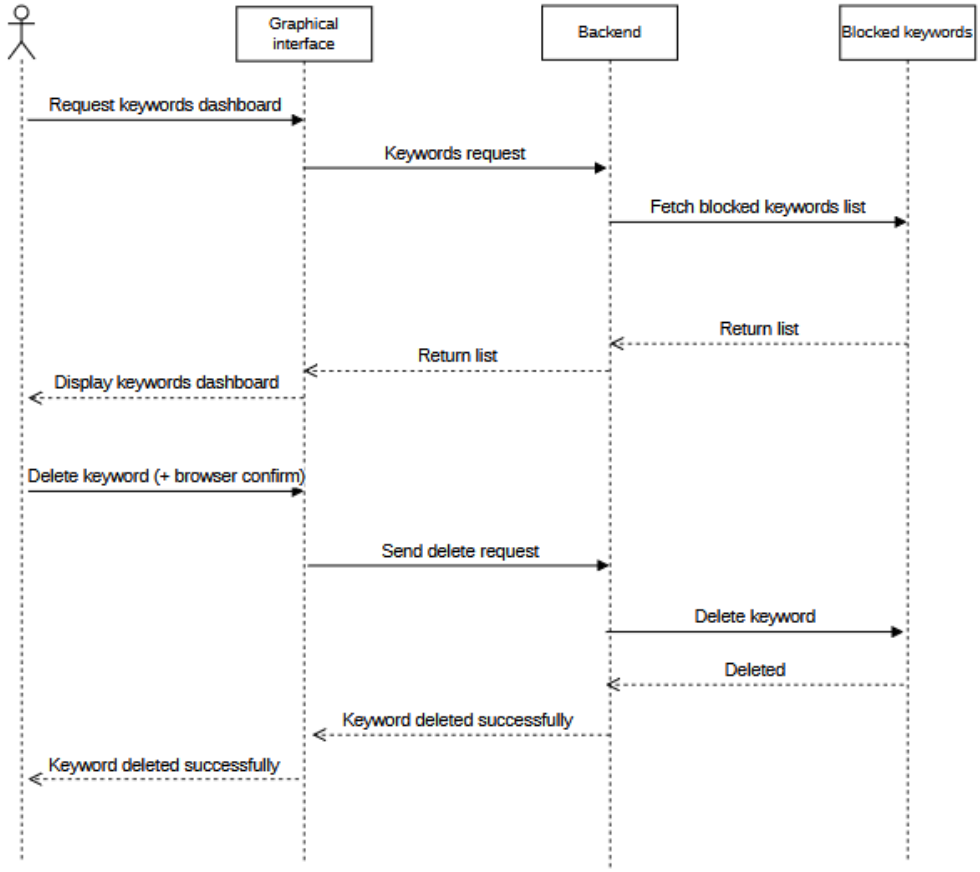


Figure 2.5: Delete Keyword Sequence Diagram.

**4. Test Filter**

The following diagram models the interaction triggered when the Admin uses the filtering simulator to test a URL, domain, or keyword without affecting real traffic. The participants are the Admin, the Graphical Interface, the Backend, and the Blocked List data store.

The Admin requests the test filter page, which the Backend serves back to the Interface. The Admin then submits a test input (URL, domain, or keyword). The Backend sends the payload for security analysis, reads the security policies and lists from the Blocked List store, and matches the input against them. An **Alt** combined fragment governs the result:

- **[Input is clear]:** The Backend returns a *status: ALLOWED* response, and the Interface displays the ALLOWED status to the Admin.
- **[Input matches blocklists]:** The Backend returns block details including the status, triggered policy, and matched keyword or domain. The Interface displays the BLOCKED status along with the triggered policy.

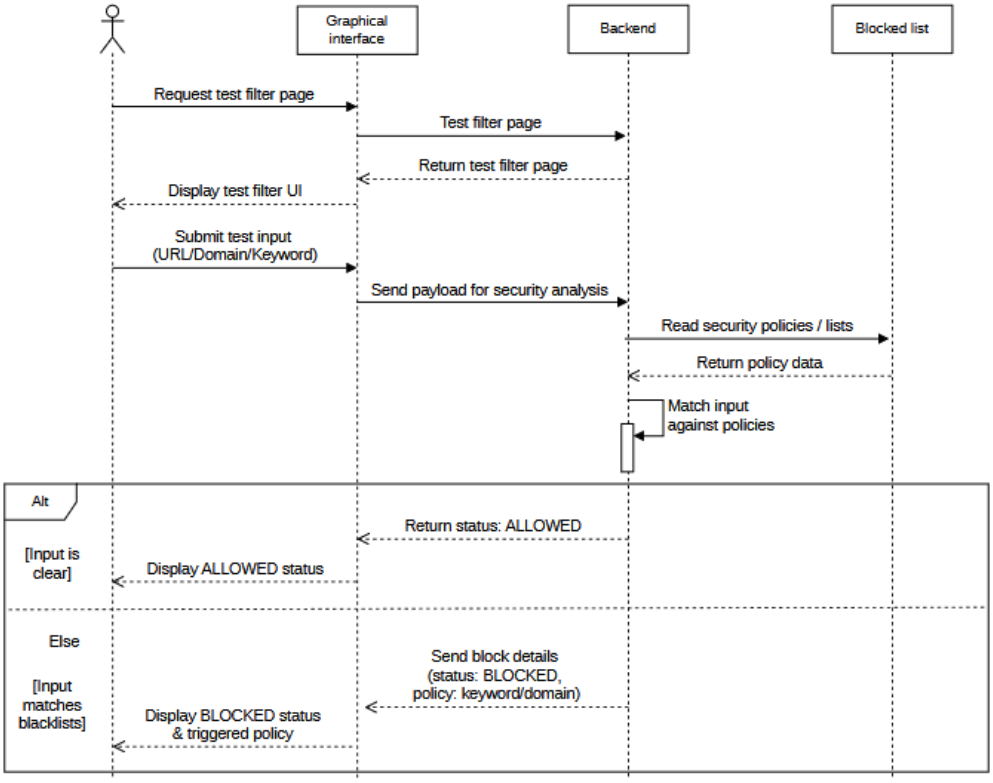


Figure 2.6: Test Filter Sequence Diagram.

**5. Monitor Logs**

The following diagram illustrates the log management interaction available to the Admin. The participants are the Admin, the Graphical Interface, the Backend, and the Logs File data store. This scenario covers three sub-functions: filtering/searching logs, exporting logs, and clearing logs.

The Admin accesses the logs dashboard. The Graphical Interface fetches the latest traffic logs from the Backend, which queries the last network events from the Logs File and returns the data. The Interface renders the log table in real time. Three **Alt** fragments then model the possible Admin actions:

- **[Admin filters logs (search)]:** The Admin types a query or applies a filter. The Backend filters the log records and returns the matching rows, which are displayed in the updated Interface.
- **[Admin clicks export]:** The Interface requests a text file download. The Backend fetches all log records, streams a text file back to the Interface, and the browser download is triggered.
- **[Admin clicks clear]:** The Interface sends a `POST /LOGS/CLEAR` flush request. The Backend truncates the logs table and returns a success status, after which the Interface refreshes and displays an empty table.

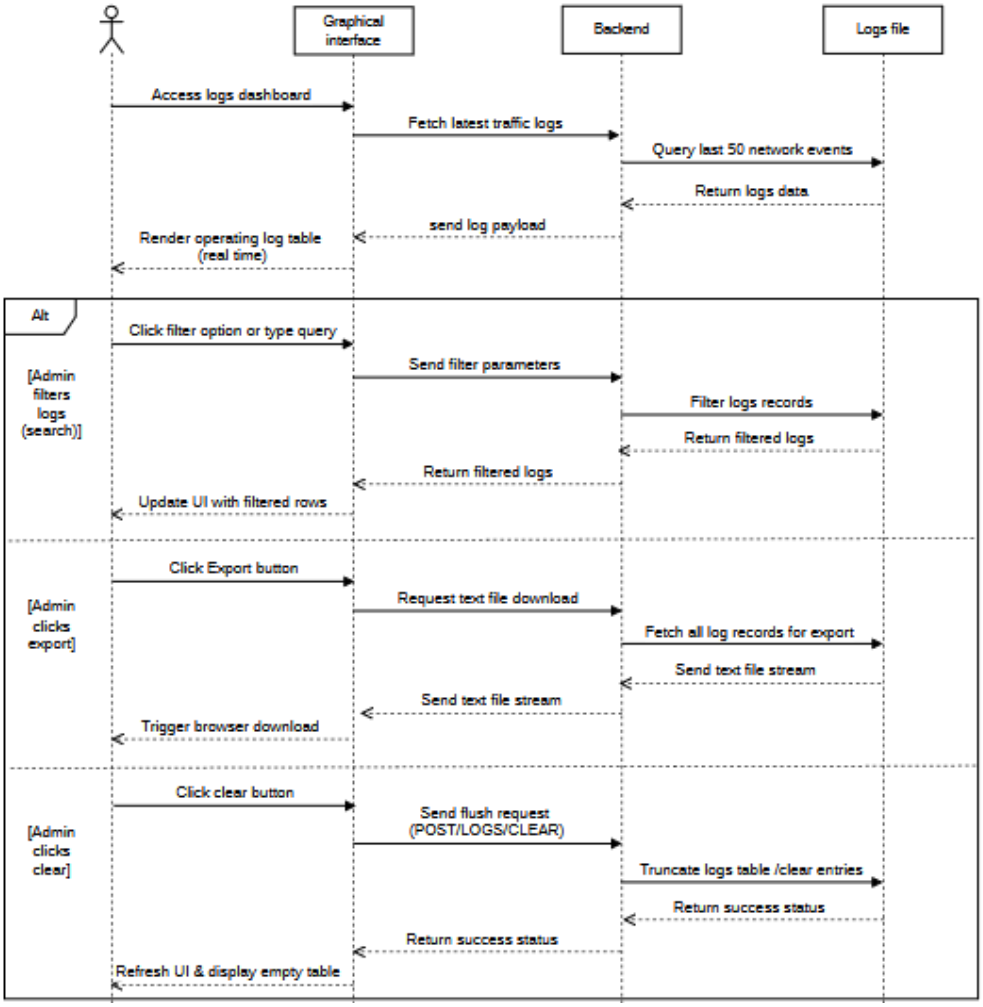


Figure 2.7: Monitor Logs Sequence Diagram.

**b. User Scenarios:**

**Browsing Site**

The following diagram models the complete request lifecycle when a User attempts to access a website through the proxy. The participants are the User, the Browser, the Proxy, and the Internet.

The User types a site URL, which the Browser forwards to the Proxy as a URL request. The Proxy performs a *Check blocked list domain* operation. A first **Alt** fragment governs the domain-level filtering decision:

- **[The site is in the blocked list]:** The Proxy returns a *This site cannot be reached* message to the Browser, which notifies the User. The interaction terminates.
- **[Else]:** The Proxy forwards the request to the Internet, which returns a response page. The Proxy then performs a *Check blocked list keywords* operation on the response content. A nested **Alt** fragment governs the keyword-level decision:

- **[A blocked keyword is found]:** The Proxy returns *This site cannot be reached* to the User.
- **[Else]:** The Proxy delivers *Show website* to the User.

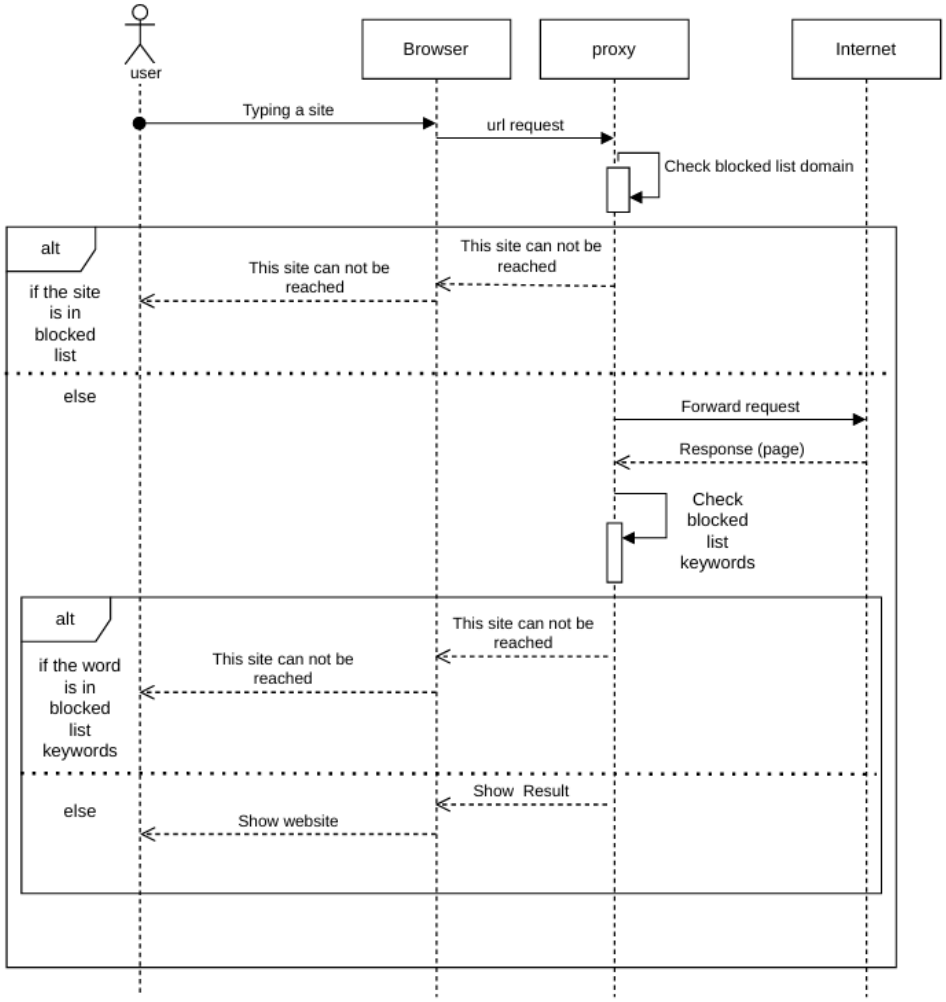


Figure 2.8: Browsing Site Sequence Diagram.

## 2.6 Conclusion

This chapter presented the analysis and design phase of the proposed web filtering proxy system using the UML modelling language. The modelling was structured around three complementary diagram types: the use case diagram, the class diagram, and the sequence diagrams.

The use case diagram defined the system boundaries and identified the interactions between the two actors the Admin and the User and the core system functionalities. The class

diagram described the static structure of the system, revealing the relationships between its main entities. The sequence diagrams detailed the dynamic behaviour of the system by illustrating the step-by-step message exchanges for each functional scenario.

These modelling steps allowed us to clearly define the system's architecture and verify that all functional requirements are addressed. This theoretical foundation guides the implementation phase presented in the following chapter.

# Chapter 3: Implementation

## 3.1 Introduction

Building upon the analysis and conception presented in the previous chapter, this chapter focuses on the practical realization of the proposed web filtering proxy system. It covers the transition from theoretical models to a fully functional application, detailing every step of the development process.

The chapter begins by presenting the development environment, including the hardware and software configurations used throughout the implementation phase. This is followed by an overview of the technologies and tools selected to build the system, with a justification for each choice. The overall system architecture is then described to provide a clear picture of how the different components interact with one another.

The core of this chapter is dedicated to the implementation of the system's main functionalities, including authentication, blocked sites and keywords management, log monitoring, the filtering engine, and the test filter simulator. Each functionality is presented alongside its corresponding user interface.

Finally, the chapter concludes with a testing and validation section, where the system's behavior is evaluated against the defined requirements to ensure correctness, reliability, and consistency.

## 3.2 Development Environment

### 3.2.1 Hardware Environment

The development and testing of the proposed system were carried out on a personal computer with the following hardware specifications:

Table 3.1: Hardware Specifications.

| **Component** | **Specification** |
|---|---|
| Processor | Intel® Core™ i3-6100U CPU @ 2.30 GHz |
| Installed RAM | 8.00 GB (DDR4, 2133 MHz) |
| Graphics Card | Intel® HD Graphics 520 (128 MB) |
| Storage | 112 GB SSD (ADATA SU650) + 466 GB HDD (WDC WD5000LPCX) |

### 3.2.2 Software Environment

The software tools and platforms used throughout the development process are summarized in the following table:

Table 3.2: Software Environment.

| Software | Version | Role |
|---|---|---|
| Windows 10 Pro | 22H2 | Operating system used for development and testing. |
| Python | 3.11.4 | Primary programming language for the backend and proxy logic. |
| Visual Studio Code | 1.120.0 | Main code editor used throughout the development phase. |

## 3.3 Technologies and Tools

This section presents the main technologies and libraries used to develop the proposed web filtering proxy system, along with a justification for each choice.

### 3.3.1 Python

Python 3.11.4 was chosen as the primary programming language for this project. Its simplicity, readability, and extensive ecosystem of libraries make it particularly well-suited for developing network-oriented and data-driven applications. Python also offers strong support for web frameworks, making it an ideal choice for building the backend of the proposed system.

### 3.3.2 Flask

Flask (v3.1.2) is a lightweight and flexible Python web framework used to build the administrative interface of the system. Unlike heavier frameworks, Flask provides only the essential components needed for web development, giving the developer full control over the application structure. It handles routing, request processing, and session management throughout the system.

### 3.3.3 HTML / CSS / JavaScript

The front-end of the administrative dashboard was developed using standard web technologies: HTML for page structure, CSS for styling and layout, and JavaScript for dynamic interactions. These technologies ensure a responsive and user-friendly interface accessible

through any modern web browser. The interface additionally supports a Dark/Light mode toggle, implemented via CSS custom properties (variables) and a JavaScript function that reads and writes the user's preference to localStorage, ensuring the selected theme persists across sessions without requiring any server-side changes.

### 3.3.4 File-based Storage

The system adopts a lightweight file-based storage approach. Blocked sites, blocked keywords, and activity logs are persisted in plain text files: `blocked_sites.txt`, `blocked_keywords.txt`, and `logs.txt`. This design choice eliminates the need for a dedicated database engine, simplifying deployment and reducing system dependencies. This approach is particularly suited for small-to-medium network environments where log volumes remain manageable, and represents a deliberate trade-off between simplicity and scalability.

### 3.3.5 Supporting Libraries

In addition to the core technologies, several Python libraries were used to support specific functionalities of the system:

Table 3.3: Supporting Libraries.

| **Library** | **Version** | **Role** |
|---|---|---|
| Werkzeug | 3.1.5 | Provides WSGI utilities and request handling for Flask. |
| Jinja2 | 3.1.6 | Template engine used to render dynamic HTML pages. |
| Bootstrap | 5.3.2 | CSS framework used for the responsive dashboard UI. |
| Font Awesome | 6.4.2 | Icon library used throughout the interface. |
| Chart.js | latest | JavaScript library used to render the traffic analysis chart on the dashboard. |

## 3.4 System Architecture

The proposed web filtering proxy system is organized into four distinct layers, each responsible for a specific role within the overall application. Figure 3.1 illustrates the interactions between these layers and the data flow across the system.

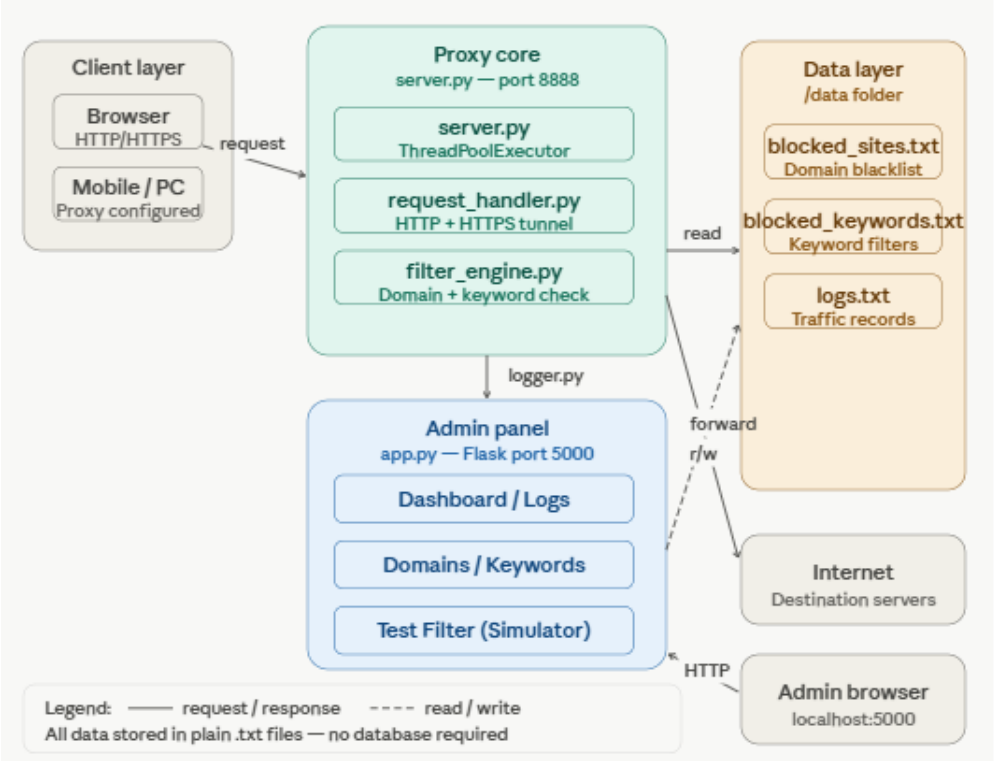


Figure 3.1: System Architecture.

The proposed system is organized into four distinct layers. The **Client layer** consists of devices (browsers, computers, or mobile phones) configured to route their traffic through the proxy. The **Proxy core** handles all incoming connections on port 8888, using a multi-threaded architecture to process HTTP and HTTPS requests simultaneously, applying domain and keyword filtering before forwarding or blocking. The **Admin panel**, served by Flask on port 5000, provides a web-based interface for managing blocked content, monitoring logs, and testing the filter engine. The **Data layer** relies on three plain text files stored in the `/data` directory, eliminating the need for a dedicated database engine.

## 3.5 Implementation of Main Functionalities

This section presents the practical realization of each core functionality of the system. For each module, the implementation logic is described and illustrated with the corresponding user interface.

### 3.5.1 Authentication

The system implements a session-based authentication mechanism using Flask's built-in session management. When the Admin submits their credentials via the login form, the backend hashes the entered password using SHA-256 and compares it against the stored hash. If the

credentials are valid, a session flag `logged_in` is set to `True` and the Admin is redirected to the dashboard. All administrative routes are protected by a `login_required` decorator that checks for this flag on every request, redirecting unauthenticated users to the login page.

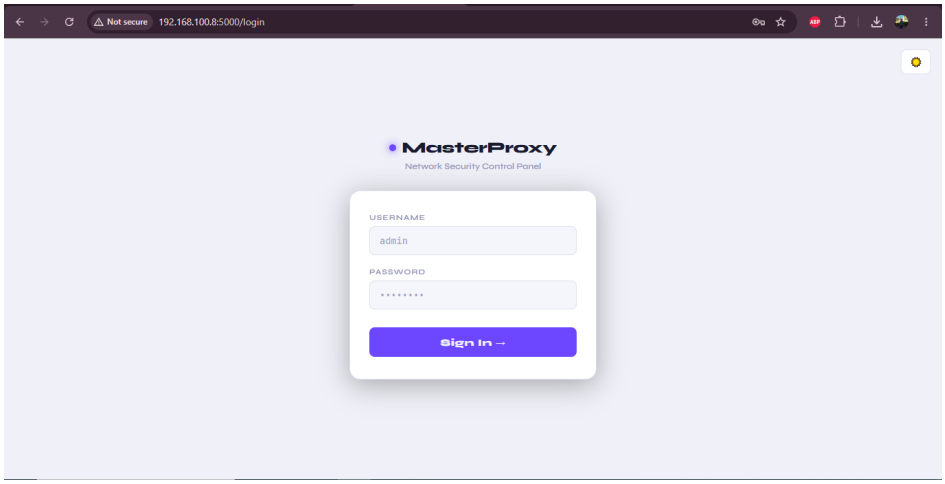


Figure 3.2: Login Interface.

The login page provides a username and password form. In the event of invalid credentials, an error message is displayed inline without redirecting the user.

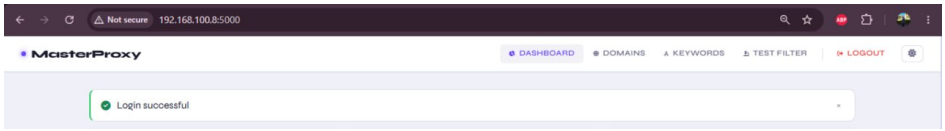


Figure 3.3: Login Success Notification.

A green banner is displayed at the top of the Dashboard immediately after successful authentication, confirming session establishment before any data loads.

### 3.5.2 Manage Blocked Sites

The blocked sites management module allows the Admin to maintain the domain blacklist through four operations. The Add operation accepts a domain name, cleans it using the `clean_domain()` function which strips protocols, `www` prefixes, ports, and query strings, then checks for duplicates before appending it to `blocked_sites.txt`. The Edit operation replaces an existing entry with a new value. The Delete operation filters out the target domain and rewrites the file. The Import operation accepts a `.txt` file, iterates over each line, and adds only non-duplicate entries, returning a count of successfully imported domains.

Figure 3.4 presents the main domain management interface, displaying the current blacklist alongside the add and import forms.

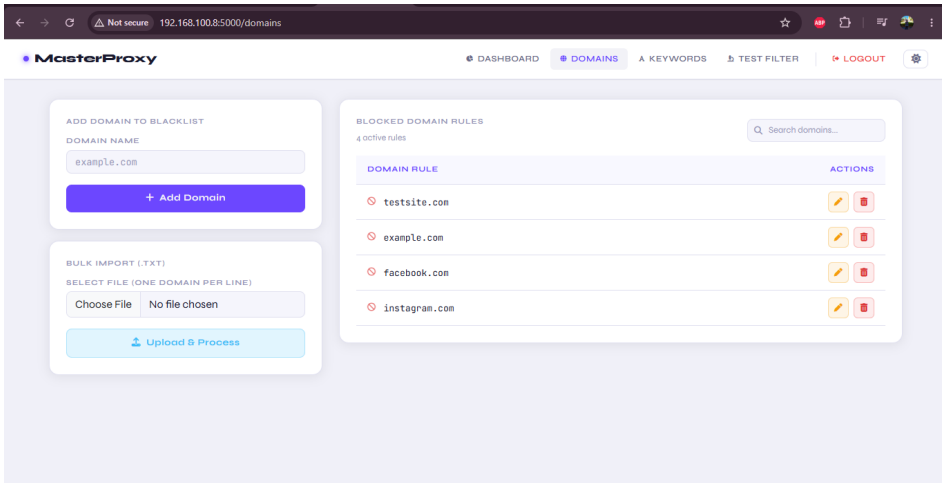


Figure 3.4: Domain Management Interface.

The following figures illustrate the complete Add Domain workflow, from entering the domain name to receiving the success notification and observing the updated blacklist.

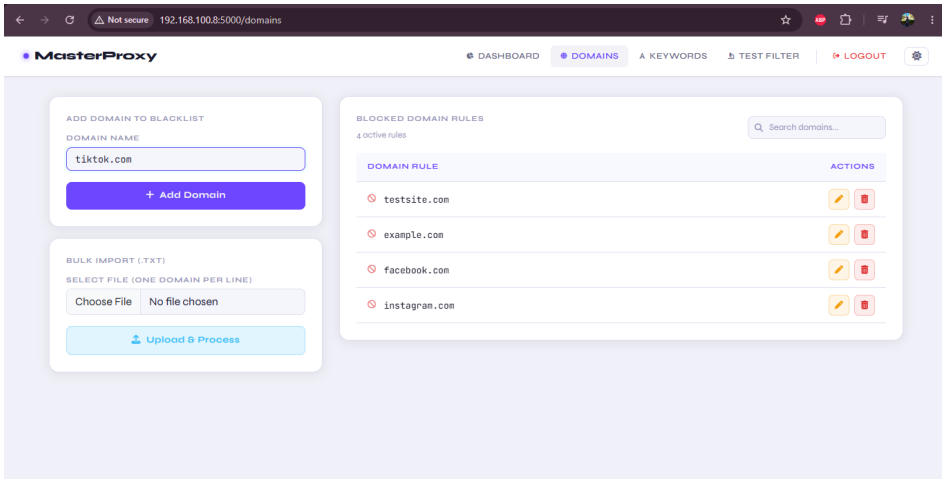


Figure 3.5: Add Domain --- Entering the domain name.

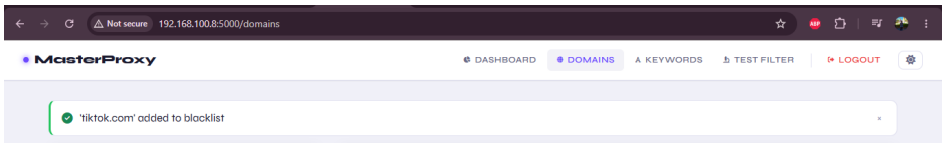


Figure 3.6: Add Domain --- Success notification after submission.

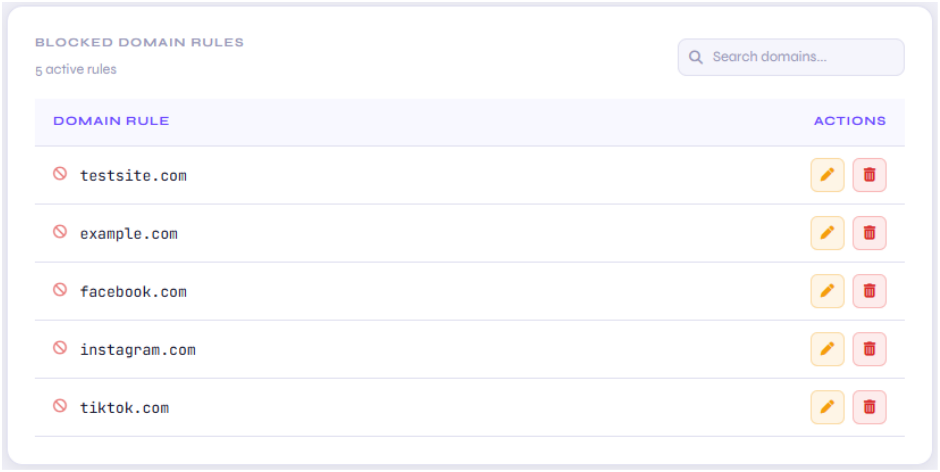


Figure 3.7: Add Domain --- Updated blacklist reflecting the new entry.

The bulk import workflow is illustrated below, showing the file selection step followed by the import confirmation and the updated domain list.

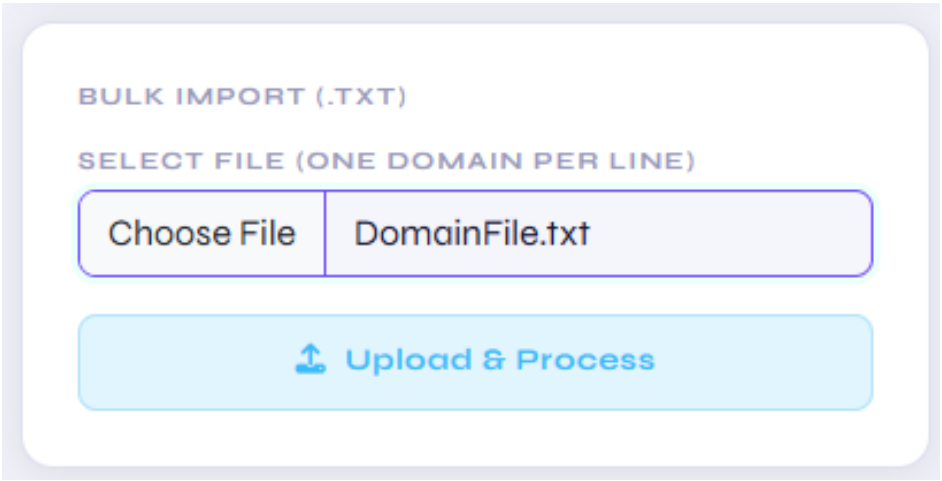


Figure 3.8: Bulk Import --- Selecting the `.txt` domain file.

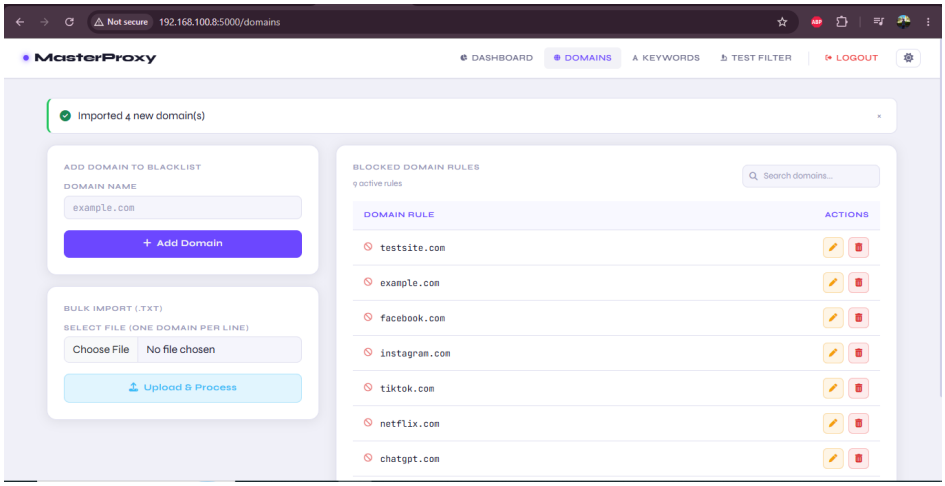


Figure 3.9: Bulk Import --- Confirmation showing the number of imported domains.

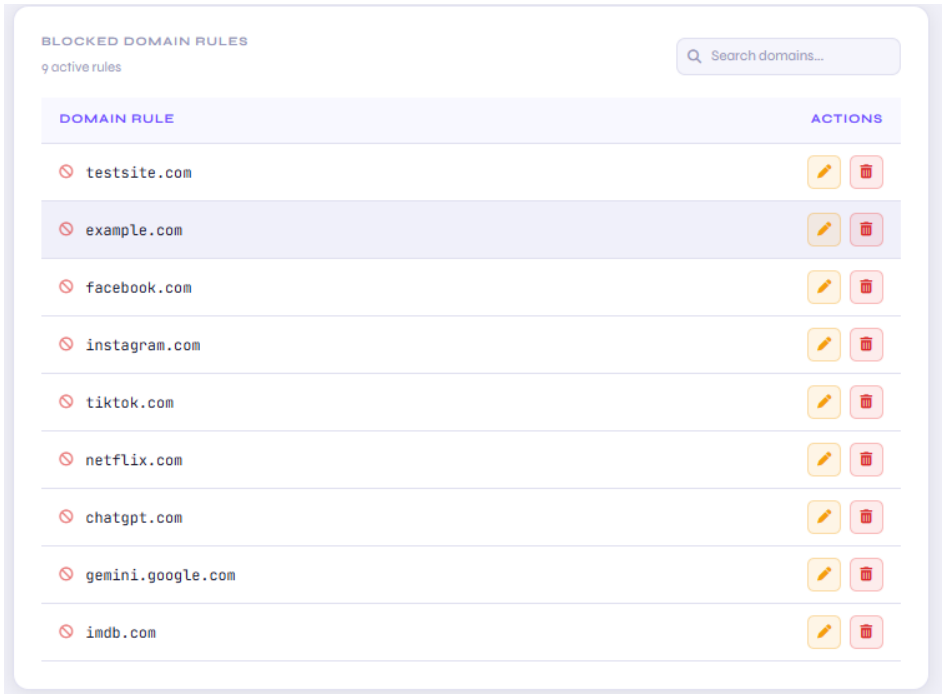


Figure 3.10: Bulk Import --- Final updated domain blacklist after import.

### 3.5.3 Manage Keywords

The keywords management module follows the same architecture as the blocked sites module, operating on `blocked_keywords.txt`. All entries are normalized to lowercase before storage to ensure case-insensitive matching during filtering. The module supports single keyword addition, bulk import from `.txt` files, inline editing, and deletion. A real-time search field on the interface allows the Admin to quickly locate specific entries within the list.

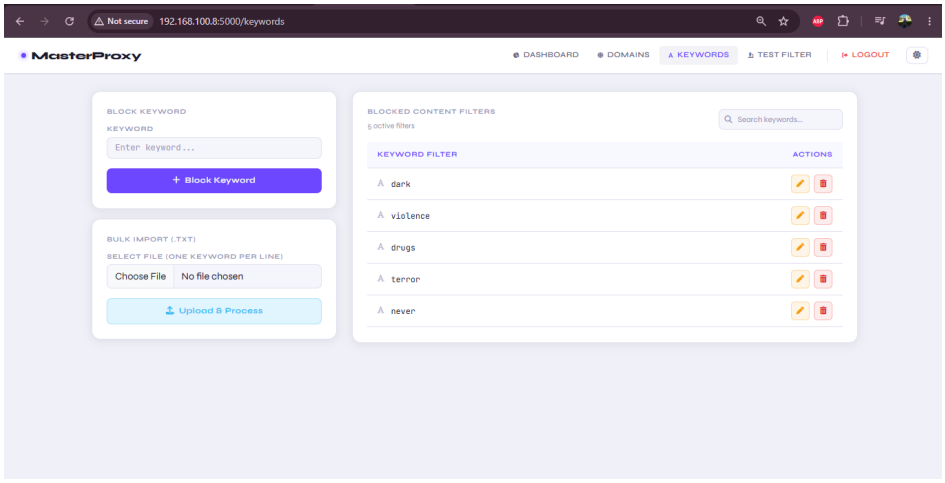


Figure 3.11: Keyword Management Interface.

The following figures illustrate the Delete Keyword workflow, showing the confirmation prompt, the deletion notification, and the updated keyword list.

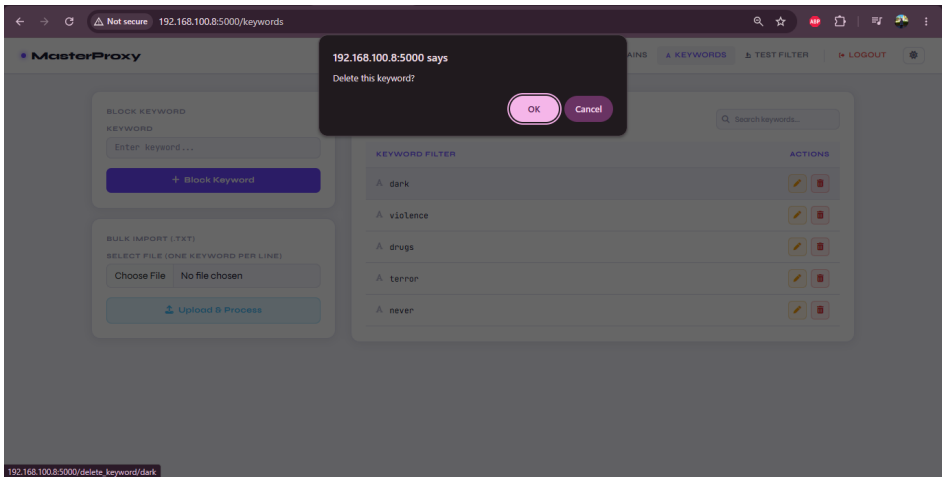


Figure 3.12: Delete Keyword --- Browser Confirmation Dialog.

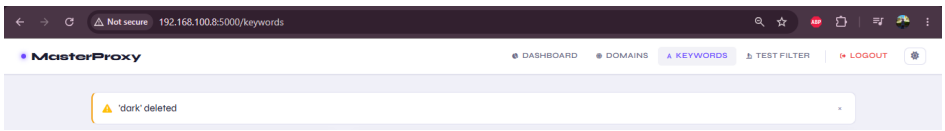


Figure 3.13: Delete Keyword --- Deletion Success Notification.

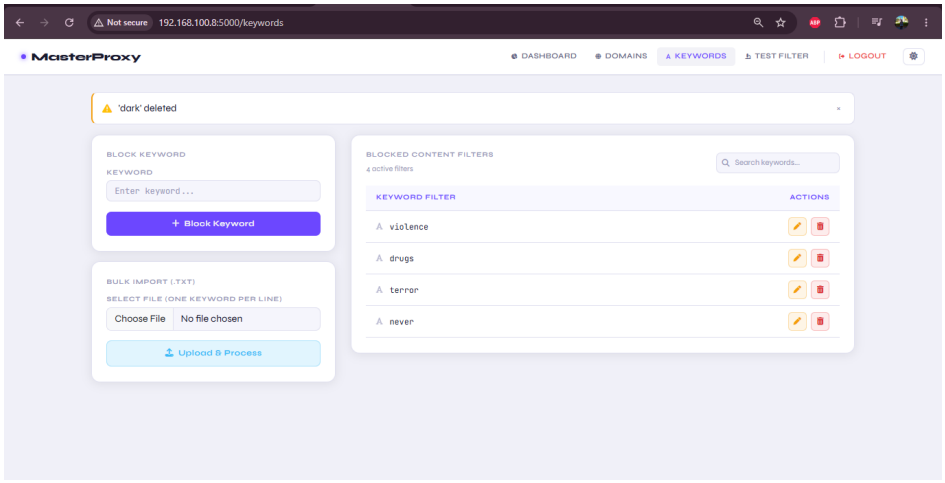


Figure 3.14: Delete Keyword --- Updated Keyword List After Deletion.

### 3.5.4 Monitor Logs

The log monitoring module provides the Admin with full visibility over network activity. On dashboard load, the system reads the last 50 entries from `logs.txt` and renders them in a structured table displaying the timestamp, client IP address, HTTP method, destination hostname, and filtering status (`ALLOWED` / `BLOCKED`). The IP address column additionally serves to identify connected users, as each log entry records the client address that initiated the proxied request. The interface includes a client-side search filter that dynamically hides non-matching rows without reloading the page. Two additional actions are available: Export, which triggers a direct file download of the complete log file, and Clear, which truncates `logs.txt` after an explicit browser confirmation. The dashboard also displays aggregate statistics (total requests, allowed sessions, blocked threats) and a doughnut chart visualizing the traffic distribution.

Figure 3.15 shows the statistics panel at the top of the dashboard, presenting the total, allowed, and blocked request counts.

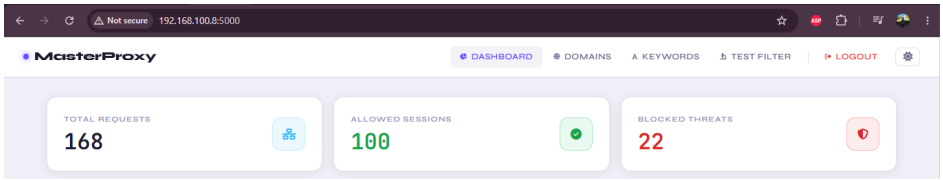


Figure 3.15: Dashboard --- Statistics Panel (Total, Allowed, and Blocked Requests).

Figure 3.16 presents the traffic analysis section, including the doughnut chart and the top blocked destinations table.

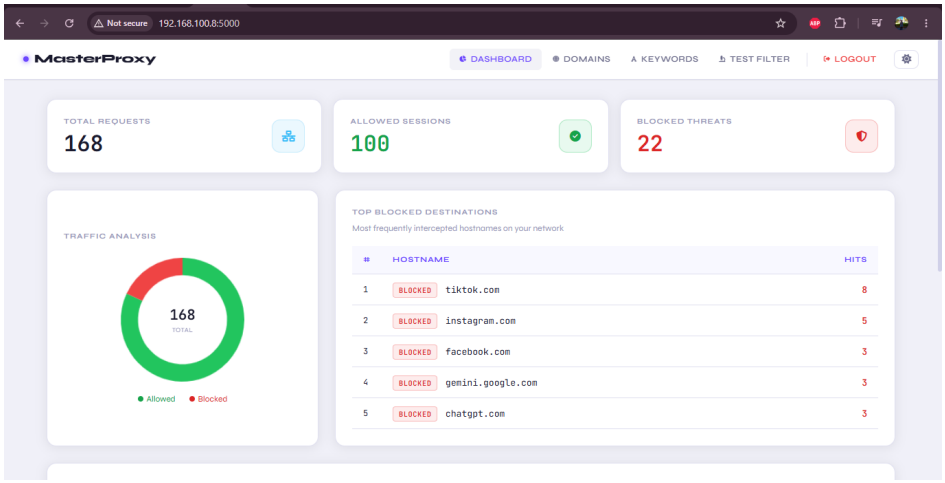


Figure 3.16: Dashboard --- Traffic Analysis Chart and Top Blocked Destinations.

Figure 3.17 shows the operations log table with real-time filtering status for each intercepted request.

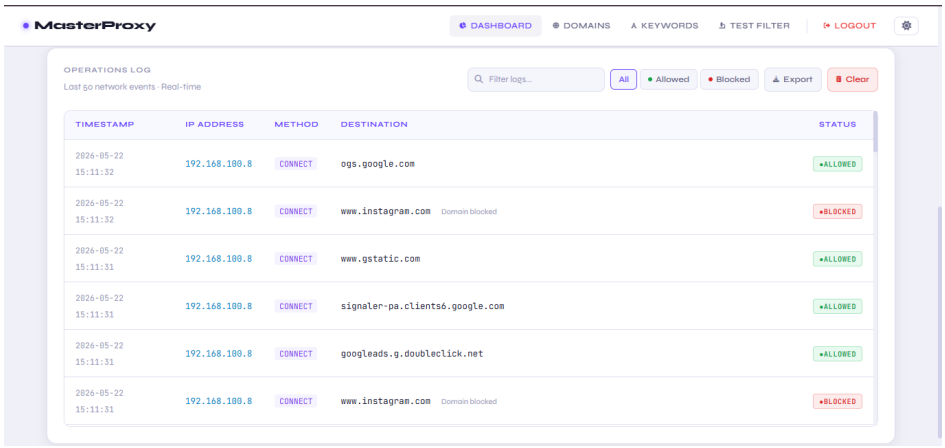


Figure 3.17: Dashboard --- Operations Log Table with ALLOWED and BLOCKED Status.

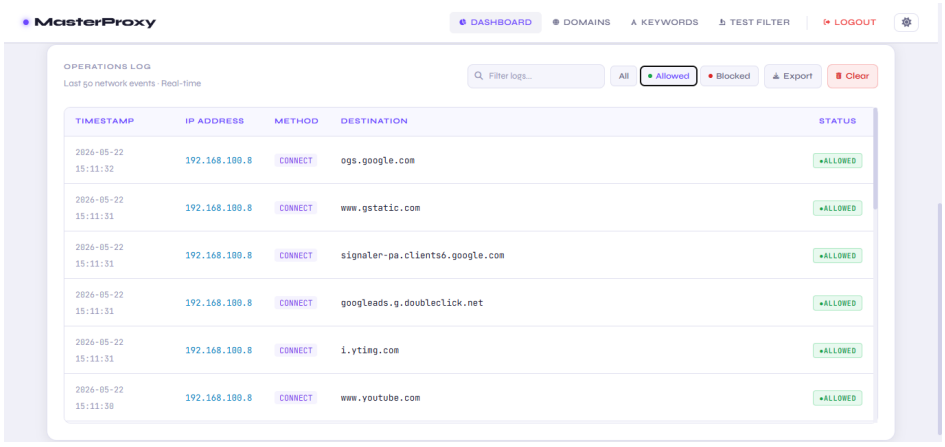


Figure 3.18: Dashboard --- Log Table Showing Blocked Entries with Domain Reason.

Figure 3.19 illustrates the Export Logs action, which triggers a direct download of the complete `logs.txt` file.

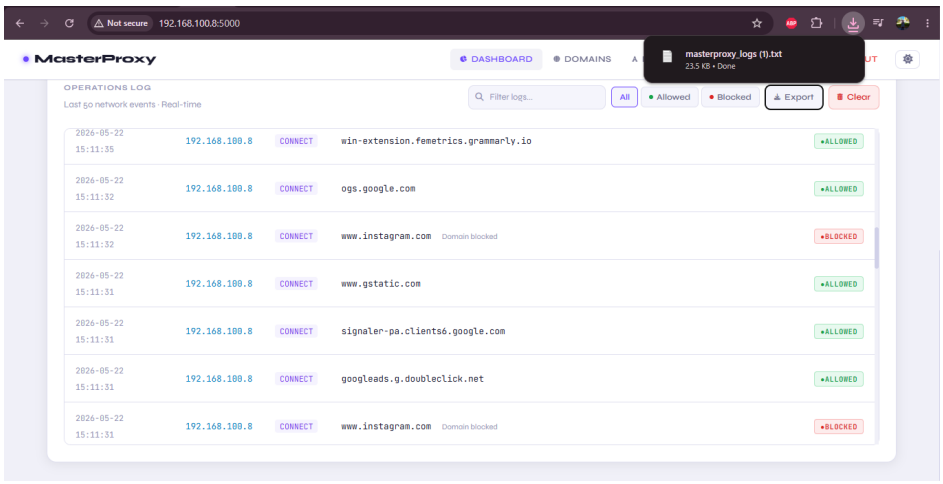


Figure 3.19: Dashboard --- Export Logs File Download.

Figure 3.20 demonstrates the real-time search filter, showing results filtered by a specific keyword.

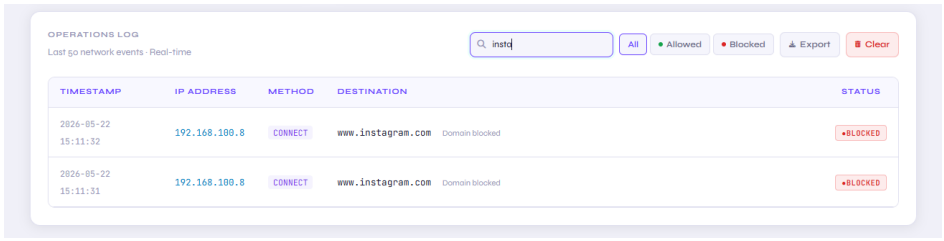


Figure 3.20: Dashboard --- Real-time Log Search Filter in Action.

The following figures illustrate the Clear Logs workflow, from the confirmation dialog to the resulting empty log state.

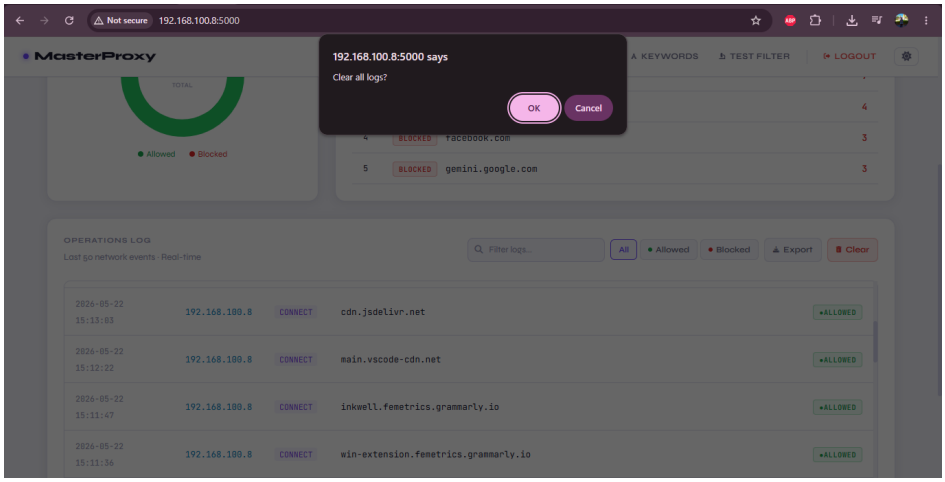


Figure 3.21: Clear Logs --- Browser Confirmation Dialog.

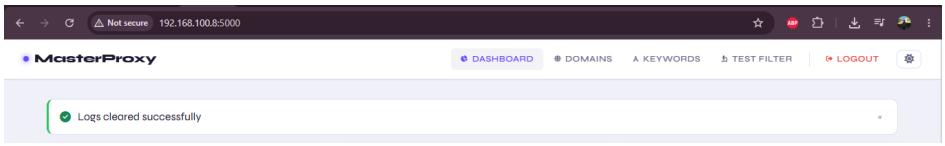


Figure 3.22: Clear Logs --- Success Notification After Clearing.

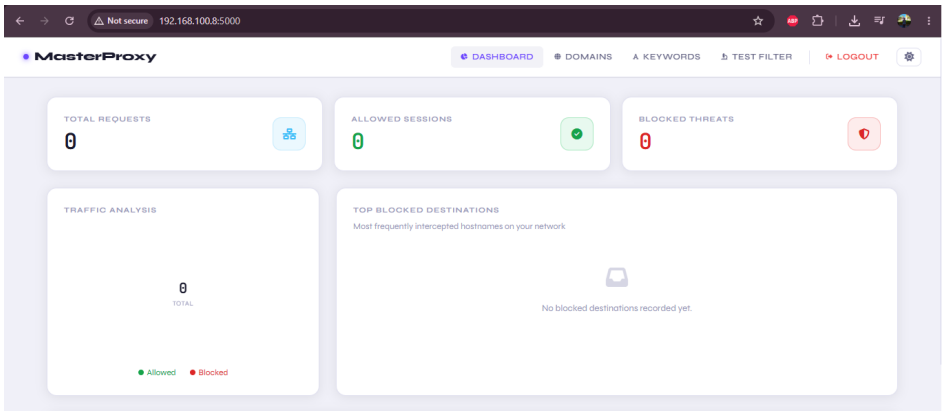


Figure 3.23: Clear Logs --- Dashboard Reset to Zero Statistics.

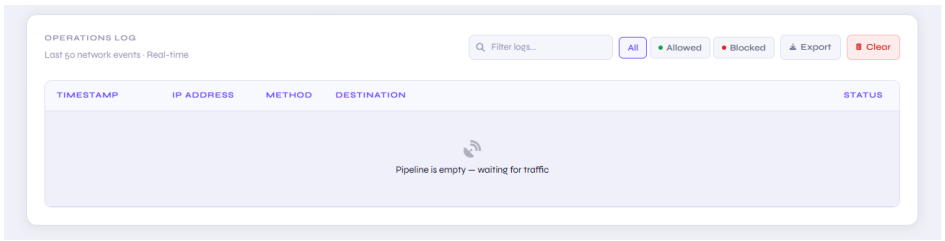


Figure 3.24: Clear Logs --- Empty Log Table Awaiting New Traffic.

### 3.5.5 Filtering Engine

The filtering engine represents the architectural core of the proxy system, implemented programmatically within `filter_engine.py`. This module intercepts every routing payload to evaluate transactional compliance prior to packet forwarding. For domain-level filtering, the engine executes the `is_blocked()` routine, which normalizes the destination host syntax via `normalize_domain()` and compares it directly against the active rules in `blocked_sites.txt`.

When an HTTP or HTTPS request targets a blacklisted domain, the socket connection is immediately severed, and the proxy injects a dynamic, localized `403 Forbidden` administrative page back into the client runtime browser environment, as demonstrated in Figure 3.25. For unencrypted HTTP traffic, the engine performs deep packet inspection via `contains_blocked_keyword()`, scanning response bodies for elements persisted in `blocked_keywords.txt`. Secure HTTPS connections are handled through an isolated `CONNECT` tunnel mechanism in `https_handler.py`, applying structural domain-level checking before establishing the encrypted tunnel interface.

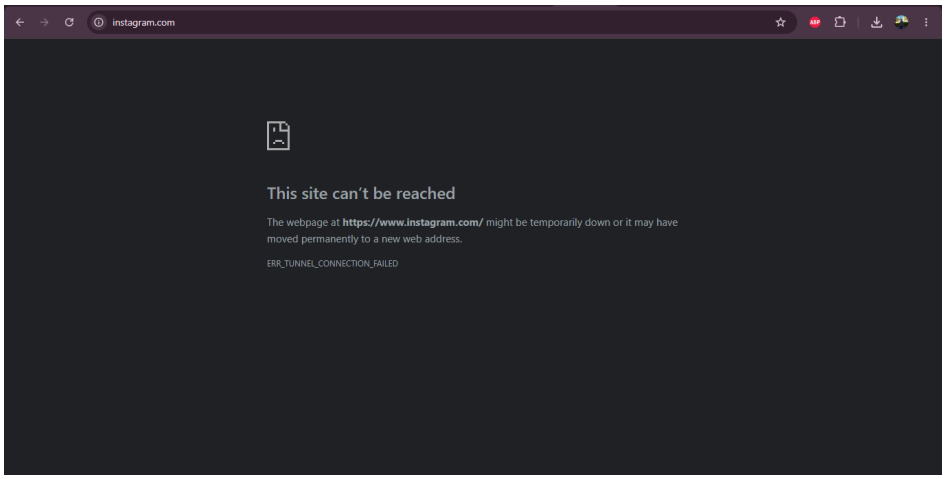


Figure 3.25: Browser Block Page Returned When Accessing a Blocked Domain (Instagram).

To guarantee systematic visibility and data persistence, every interception event handled by the core socket pipeline is logged asynchronously via `logger.py` into the underlying flat-file repository (`logs.txt`). This file employs a pipe-separated matrix structure capturing strict metadata parameters: `timestamp | IP | method | host | status | reason`.

Finally, to bridge this low-level file data with the administrative layer, the Flask backend extracts these raw transactional lines and parses them into a structural view. As illustrated in Figure 3.26, the management dashboard serializes the raw logs into a readable, reactive data grid equipped with client-side dynamic search and status color tokens.

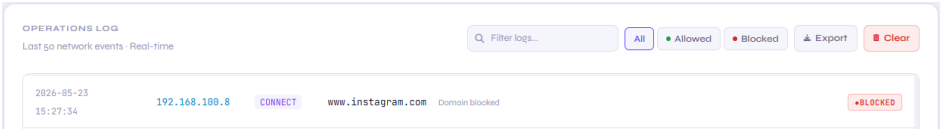


Figure 3.26: Administrative Flask Dashboard Interface Displaying the Processed Event Log Data Table.

### 3.5.6 Test Filter Simulator

The test filter simulator, accessible at `/check`, allows the Admin to evaluate the filtering engine against any input without generating real network traffic. The Admin submits a URL, domain, or keyword via the interface. The backend applies the same `clean_domain()` normalization, then checks the input sequentially against the domain blacklist and the keyword list. The result is displayed immediately on the page with a clear `ALLOWED` or `BLOCKED` status badge, along with the triggered policy type (`Domain Blacklist Policy` or `Keyword Content Policy`). Each test is also recorded in `logs.txt` with the source IP `127.0.0.1` to distinguish simulator entries from real proxy traffic.

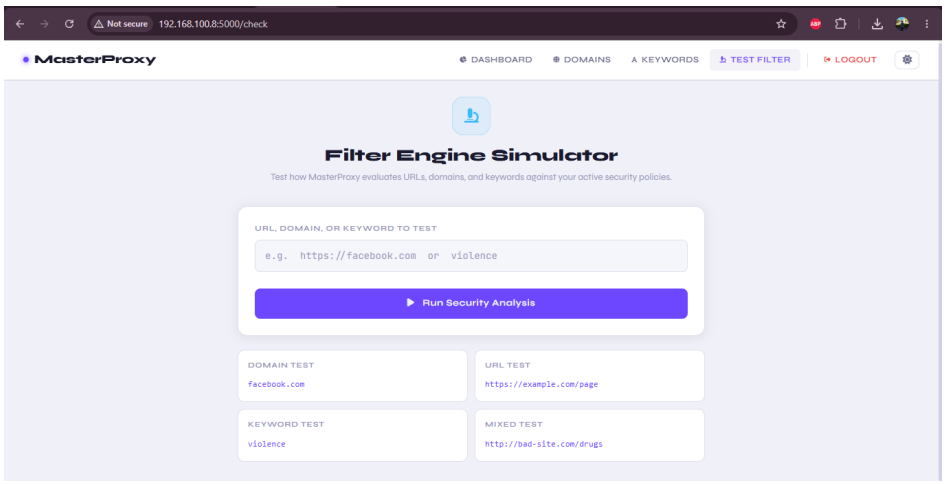


Figure 3.27: Test Filter Simulator Interface.

Figure 3.28 shows the result returned when the submitted input matches a domain blacklist entry.

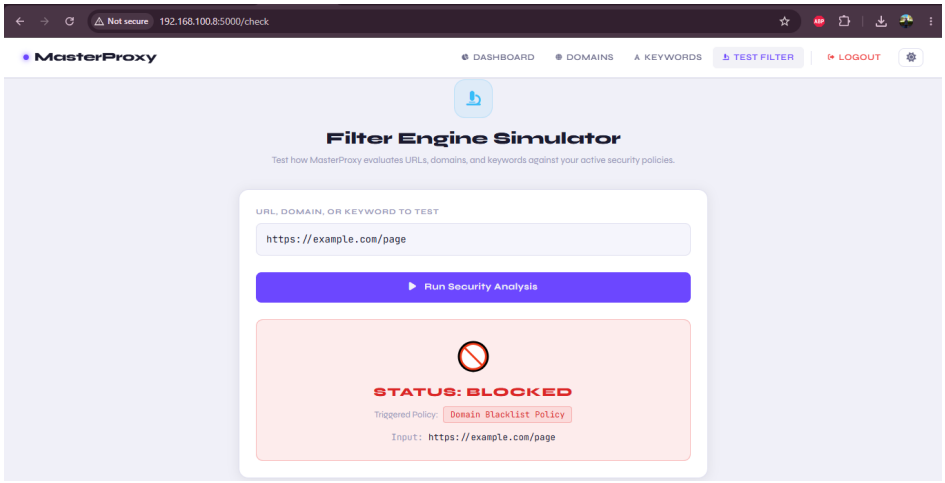


Figure 3.28: Test Filter --- BLOCKED Result with Triggered Policy.

Figure 3.29 shows the result returned when the submitted input does not match any active security policy.

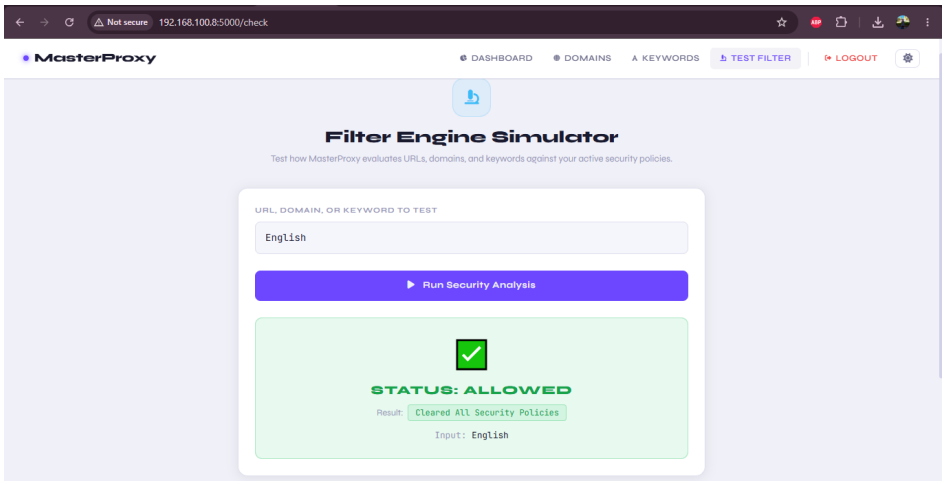


Figure 3.29: Test Filter --- ALLOWED Result After Clearing All Security Policies.

## 3.6 Testing and Validation

### 3.6.1 Testing Approach

The system was validated through functional testing, where each feature was tested against a defined set of inputs to verify that the output matched the expected behavior. Testing was performed on a local network using a mobile hotspot to simulate a real deployment environment, with client devices configured to route traffic through the proxy at port `8888` and the admin panel accessed at port `5000`.

### Functional Test Cases

Table 3.4: Functional Test Cases.

| Test | Feature | Input | Expected Result | Result |
|---|---|---|---|---|
| T01 | Login | Valid credentials (admin / admin123) | Redirect to dashboard | Pass |
| T02 | Login | Invalid credentials | Error message displayed | Pass |
| T03 | Add domain | facebook.com | Added to blacklist | Pass |
| T04 | Add domain | facebook.com (duplicate) | Duplicate error message | Pass |
| T05 | Import domains | .txt file with 5 domains (2 duplicates) | 3 new domains imported | Pass |
| T06 | Delete domain | facebook.com | Removed from blacklist | Pass |
| T07 | Add keyword | violence | Added to keyword list | Pass |
| T08 | Proxy filtering | Request to facebook.com | 403 block page returned | Pass |
| T09 | Proxy filtering | Request to google.com (not blocked) | Page loaded normally | Pass |
| T10 | Keyword filtering | HTTP response containing ``violence'' | 403 block page returned | Pass |
| T11 | Test Filter | facebook.com | BLOCKED --- Domain Blacklist Policy | Pass |
| T12 | Test Filter | google.com | ALLOWED --- Cleared All Policies | Pass |
| T13 | Test Filter | violence | BLOCKED --- Keyword Content Policy | Pass |
| T14 | Export logs | Click Export button | .txt file downloaded | Pass |
| T15 | Clear logs | Click Clear + confirm | Logs table emptied | Pass |
| T16 | Logout | Click Logout | Session cleared, redirect to login | Pass |

### Validation Summary

All defined test cases passed successfully. The proxy correctly intercepts and filters both HTTP and HTTPS traffic at the domain level, and applies keyword-based filtering on HTTP response content. The administrative interface accurately reflects all changes in real time, and the logging system maintains a complete and consistent record of all network events. The test filter simulator produces results identical to those of the live proxy engine, confirming

the consistency between the simulation and the actual filtering logic.

## 3.7 Limitations

Although the proposed system fulfils its core objectives and passed all defined test cases, several limitations were identified during development and testing:

- **HTTPS keyword filtering is not supported.** Because HTTPS traffic is encrypted end-to-end, the filtering engine can only apply domain-level blocking for HTTPS connections. Keyword inspection is restricted to unencrypted HTTP responses, which represent a diminishing share of modern web traffic.
- **File-based storage does not scale to high-traffic environments.** Storing blocked lists and logs in plain text files is practical for small networks, but concurrent write operations under heavy load could introduce race conditions or slow down the proxy response time.
- **Single-administrator model.** The system supports only one administrator account. Multi-user access control with role-based permissions is not available in the current version.
- **No SSL/TLS inspection.** The proxy does not perform Man-in-the-Middle (MITM) decryption of HTTPS traffic. This means that content filtering cannot be applied to the body of encrypted responses, limiting the depth of inspection for secure connections.
- **Testing was conducted in a controlled local environment.** All validation was performed on a private network using a mobile hotspot. Behaviour under real-world conditions, such as high concurrency or very large blocklists, was not evaluated.

## 3.8 Future Work

Based on the limitations identified above and the feedback gathered during testing, the following extensions are proposed for future development:

- **SSL/TLS inspection (HTTPS keyword filtering).** Integrating a MITM certificate authority would allow the proxy to decrypt, inspect, and re-encrypt HTTPS traffic, enabling keyword-level filtering for encrypted connections and significantly extending the system's coverage.
- **Database-backed storage.** Migrating from plain text files to a lightweight relational database such as SQLite would improve concurrency handling, enable more efficient querying of large log files, and support atomic write operations under heavy load.

- **Site alternative redirection.** A planned feature would allow the administrator to associate a blocked domain with a suggested alternative URL, which would be presented to the user on the block page instead of a plain error message, providing a more informative user experience.
- **Multi-administrator support.** Adding role-based access control would allow multiple administrators with different privilege levels to manage the system independently, improving usability in institutional environments.
- **Real-time dashboard updates.** Replacing the current static log table with WebSocket-based live updates would give administrators an instant view of network activity without requiring manual page refreshes.
- **Category-based filtering.** Integrating a domain classification API would allow the system to block entire categories of websites (e.g., social media, gambling, adult content) rather than requiring manual entry of individual domains.

## 3.9 Conclusion

This chapter presented the complete implementation of the proposed web filtering proxy system, covering all phases from the development environment to functional testing and validation.

The development environment and the selected technologies were described, justifying the choice of Python, Flask, and a file-based storage approach as appropriate solutions for the targeted deployment context. The system architecture was then detailed, illustrating the separation between the proxy core, the administrative panel, and the data layer.

The implementation of each core functionality was presented alongside its corresponding user interface, demonstrating the practical realization of the use cases defined in the previous chapter. The filtering engine was shown to correctly intercept and evaluate both HTTP and HTTPS traffic, applying domain-level and keyword-level policies as designed. The test filter simulator was confirmed to produce results consistent with the live proxy engine, validating the coherence of the system.

All sixteen functional test cases passed successfully, confirming that the system behaves as expected under normal operating conditions. The identified limitations and the proposed future extensions provide a clear roadmap for improving and scaling the system beyond its current scope.

# General Conclusion

Throughout the lifecycle of this Master's research, we have successfully conceptualized, designed, and developed a centralized network auditing and security environment focused on a custom Python-based web filtering proxy platform integrated with an intuitive Flask management dashboard.

The implemented prototype demonstrates the technical feasibility of combining low-level multi-threaded socket programming with lightweight web-driven configuration layers. In terms of runtime evaluation, the custom proxy engine successfully performed real-time connection interception, extracted hostname targets from application-layer packet sequences, and enforced dynamic rule-based blocklists efficiently. Furthermore, the administrative dashboard successfully mitigated manual configuration errors by providing structural CRUD facilities, modular flat-file database updates, and responsive activity log visualization.

Our core practical and technical contributions can be summarized as follows:

- Developing an independent, socket-based web filtering proxy architecture that handles baseline application-layer protocol routing without relying on heavy commercial network tools.
- Designing an integrated, single-pane-of-glass administrative layout using the Flask framework to facilitate real-time log analysis and domain-level database interactions.
- Delivering a structured validation test bench that empowers system administrators to inspect, filter, and audit network traffic patterns dynamically.

To advance the capabilities, security posture, and overall throughput performance of the proposed proxy management ecosystem, we suggest the following future directions:

- **Implementation of Full SSL/TLS Inspection:** Integrating an internal dynamic Certificate Authority (CA) into the proxy infrastructure to perform full Man-in-the-Middle (MitM) decryption, enabling deep packet inspection and keyword filtering across secure HTTPS body layers.
- **Migration to a Robust DBMS Environment:** Replacing the local file-based storage architecture with an enterprise-grade database engine (such as SQLite or PostgreSQL) to enhance data transactional velocity and optimize log searching speeds.
- **Integration of Asynchronous Optimization:** Re-engineering the core connection handlers utilizing Python's asynchronous libraries (`asyncio`) to minimize network latency and handle a higher volume of concurrent client requests under heavy traffic loads.

We believe these empirical findings lay down a reliable architectural framework for lightweight, secure, and centralized private corporate boundary defense deployment.

# References


[1] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 8th ed. Pearson Education, 2021.

[2] A. S. Tanenbaum and D. J. Wetherall, *Computer Networks*, 6th ed. Pearson Education, 2021.

[3] D. Wessels, *Squid: The Definitive Guide*. O'Reilly Media, 2004.

[4] NGINX Inc., *Nginx documentation: Reverse proxy*, https://nginx.org/en/docs/.

[5] HAProxy Technologies, *Haproxy documentation*, https://www.haproxy.org/documentation.html.

[6] Envoy Project, *Envoy proxy documentation*, https://www.envoyproxy.io/docs/.

[7] Cloudflare, *The state of application security in 2023*, 2023.

[8] Akamai Technologies, *State of the internet report: Security and performance*, 2023.

[9] J.-L. Gailly and M. Adler, *Proxy auto-configuration (pac) file format*, https://findproxyforurl.com/pac-functions/.

[10] L. Gao et al., ``The effectiveness of web prefetching," in *ACM SIGMETRICS*, 2002, pp. 243–254.

[11] S. Podlipnig and L. Böszörmenyi, ``A survey of web cache replacement strategies," *ACM Computing Surveys*, vol. 35, no. 4, pp. 374–398, 2003.

[12] R. Fielding et al., ``Hypertext transfer protocol (http/1.1): Semantics and content," Internet Engineering Task Force (IETF), Tech. Rep. RFC 9110, 2022.

[13] Apache Software Foundation, *Apache http server documentation: Mod_proxy*, https://httpd.apache.org/docs/.

[14] V. Paxson, ``Bro: A system for detecting network intruders in real-time," *Computer Networks*, vol. 31, no. 23-24, pp. 2435–2463, 1999.

[15] OWASP Foundation, *Owasp top ten web application security risks*, https://owasp.org/www-project-top-ten/.

[16] L. Richardson and S. Ruby, *RESTful Web Services*. O'Reilly Media, 2007.

[17] M. Leech et al., ``Socks protocol version 5," Network Working Group, Tech. Rep. RFC 1928, 1996.

[18] Gartner, *Market guide for secure web gateways*, 2024.

[19] Electronic Frontier Foundation, *Privacy and security guidelines for proxy services*, https://www.eff.org/.

[20] NIST, ``Zero trust architecture,'' National Institute of Standards and Technology, Tech. Rep. SP 800-207, 2020.

[21] R. Dingledine, N. Mathewson, and P. Syverson, ``Tor: The second-generation onion router,'' in *USENIX Security Symposium*, 2004, pp. 303–320.

[22] Istio Project, *Istio service mesh documentation*, https://istio.io/latest/docs/.

[23] M. Bishop, ``Http/3,'' Internet Engineering Task Force (IETF), Tech. Rep. RFC 9114, 2022.

[24] J. Iyengar and M. Thomson, ``Quic: A udp-based multiplexed and secure transport,'' Internet Engineering Task Force (IETF), Tech. Rep. RFC 9000, 2021.

[25] S. Newman, *Building Microservices: Designing Fine-Grained Systems*, 2nd ed. O'Reilly Media, 2021.

[26] E. Rescorla et al., *Tls encrypted client hello*, Internet-Draft, IETF, 2023.

[27] Forrester Research, *The forrester wave: Zero trust extended ecosystem platform providers*, 2023.

[28] Linkerd Project, *Service mesh design patterns*, https://linkerd.io/.

[29] Cloud Native Computing Foundation, *Cncf annual survey: Cloud native trends*, 2024.

[30] W3C, *Webassembly specification*, https://www.w3.org/TR/wasm-core/, 2024.