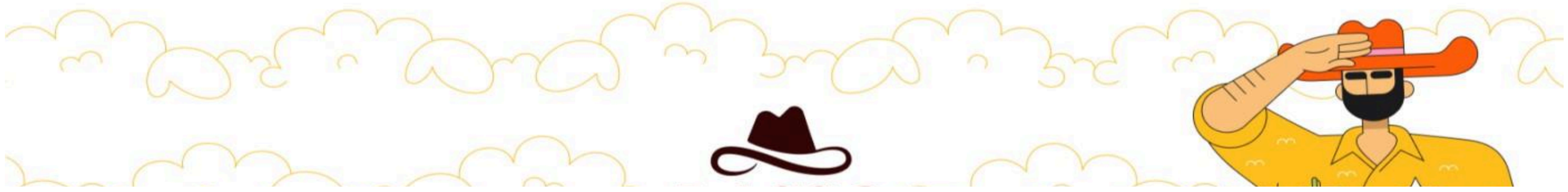


Lasso Security LLM Proxy Service

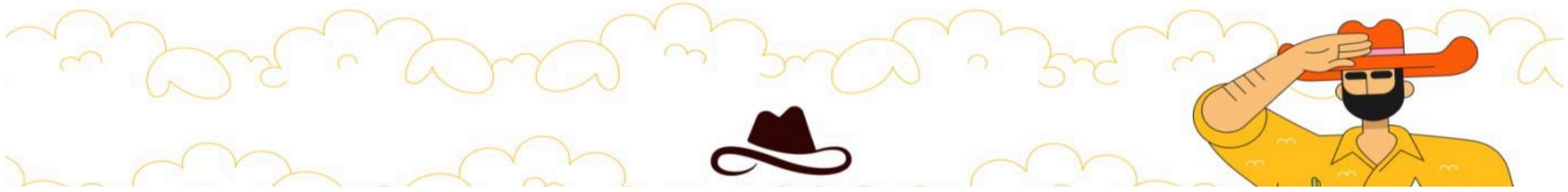
Introduction

This document outlines the design and implementation plan for the Lasso Security LLM Proxy Service. The primary aim is to create a **secure and transparent intermediary** for customer applications. This is achieved by having customer applications point their LLM SDK's baseURL to the proxy's localhost address. The proxy will then intercept, inspect, and modify all outgoing requests before forwarding them to the appropriate official LLM provider API.

The service will specifically focus on two key endpoints for enhanced security: **OpenAI's chat/completions** and **Anthropic's messages**. For all other endpoints, the proxy will simply forward the requests directly to the destination without any modification. This phased approach will gradually introduce core functionalities, ensuring a clear and robust development process.

Technical Requirements & Phased Implementation

The project is broken down into four key phases, each building upon the last to create a comprehensive and feature-rich service.



Phase 1: Basic Endpoint Setup & Request Forwarding

Goal: To establish a foundational NestJS server that can intercept requests from LLM SDKs and forward them to the correct provider API.

- **Endpoints**: The server will have catch-all endpoints for each provider, such as POST /openai/* and POST /anthropic/*.
- **Behavior**: The server will receive requests from client SDKs that have their baseURL configured to point to the proxy. It will then extract the original request path, determine the correct provider, and forward the entire request (including body and headers) to the official provider's API.
- **Verification**: This functionality will be verified by directing an SDK's baseURL to the local server and confirming that the proxy successfully forwards the request to the official provider and returns the provider's response back to the client.

Phase 2: Data Sanitization & Proxying

Goal: To implement core security features by automatically anonymizing sensitive data and classifying content for the specific supported endpoints.

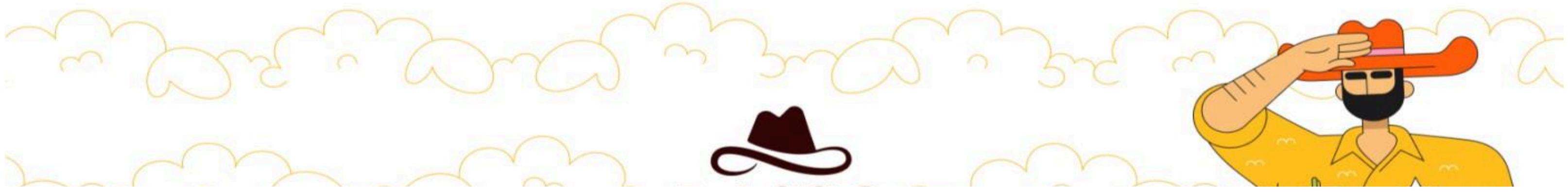
2.a. Email Address Detection

The service will detect and anonymize email addresses in the request payload for /chat/completions and /messages by replacing them with EMAIL_PH.

Bonus Suggestions (Optional):

- IP Address (IPv4) Detection: Consider detecting and replacing IP addresses with IP_ADDRESS_PH.
- IBAN Detection: Consider detecting and replacing IBAN numbers with IBAN_PH.
- Try to think of a reusable parsing method that will support easy matching for new keywords in the future.

- **Proxying**: The modified, sanitized request will be forwarded to the official provider's API. For all other endpoints not on the approved list, the request will be forwarded directly without modification.



Phase 3: Conditional Blocking & Secure Caching

Goal: To implement a dynamic blocking mechanism and a security-aware caching layer.

- **Conditional Blocking**: The service will inspect the current time's seconds value. If the seconds digit is 1, 2, 7, or 8, the request will be blocked, and an error response will be returned. For all other seconds values, the request will be allowed to proceed.

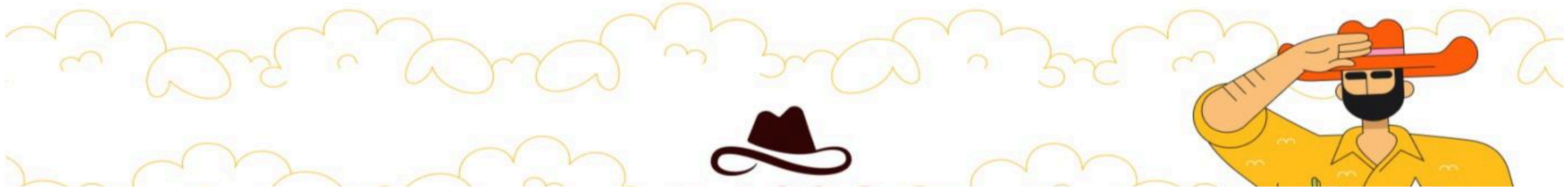
Phase 4: Persistence & Logging

Goal: To provide a robust auditing and visibility layer by storing request data in a database.

- **Database**: PostgreSQL, which can be run locally using Docker Compose.
- **Logging Data**: For each request, the service will log:
  - A timestamp.
  - The provider name (e.g., openai, anthropic).
  - The **anonymized** request payload.
  - The action taken, which can be one of the following:
    - **proxied** (sent to the LLM provider).
    - **blocked_time** (stopped due to the Phase 3 rule).
    - **blocked_financial** (stopped due to the LLM-based policy).
    - **served from cache** (returned a cached result).
- **Performance**: Logging will be handled **asynchronously** to ensure it does not introduce significant latency to the request-response cycle.

Design Considerations

- **Scalability**: The architecture will be designed to easily accommodate new LLM providers without extensive code changes.
- **Modularity**: Provider-specific logic will be encapsulated to allow for easy swapping or modification without affecting the core proxying functionality.
- **Low Latency**: Processing within the proxy will be minimized to ensure the lowest possible latency for client requests.



Definition of Done

Upon completion, the project will be delivered as a GitHub repository, including:

- **Server Artifacts**: A well-structured codebase and a README.md with clear instructions on how to install and run the server locally.
- **Functionality Checklist**:
    - **Phase 1**: Endpoints for /openai and /anthropic are active, and all requests are correctly forwarded to the destination.
    - **Phase 2**: Requests to **chat/completions** and **messages** are successfully anonymized via regex and perform LLM-based financial topic classification before proxying. Other endpoints are forwarded unmodified.
    - **Phase 3**: Requests are blocked based on the time-based rule, and identical, sanitized requests within the configured time window are correctly served from the cache.
    - **Phase 4**: All requests are logged in the PostgreSQL database with the correct timestamp, provider, anonymized payload, and action taken.
    - **Phase 5 (Bonus)** LLM-Based Policy Enforcement**:** For the supported endpoints, a secondary, lightweight LLM call will classify the request's content. A prompt will be used to determine if the user's input relates to a 'FINANCIAL' topic. If so, the request will be blocked; otherwise, it will proceed.