# **Request Engine - Functional Specification**

## **Purpose**

The Request Engine is a Python-based, modular system that enables secure, observable, and automated execution of HTTPS requests. It simulates browser-like behavior via TLS fingerprinting and rotates proxy servers to avoid IP bans or rate limits. It is ideal for scenarios involving network diagnostics, scraping, URL monitoring, and automated endpoint validation.

## **Intended Users**

| **User Role** | **Description** |
| --- | --- |
| Developer | Implements and customizes the system for organization-specific workflows |
| QA Engineer | Uses the tool to validate endpoint availability and behavior |
| Data Analyst | Consumes logs and metrics to evaluate success rates, failures, or anomalies |
| DevOps Engineer | Integrates it into CI/CD pipelines or infrastructure health checks |

## **Use Cases**

| **Use Case** | **Description** |
| --- | --- |
| Endpoint Uptime Monitoring | Send scheduled requests to key URLs and monitor status codes |
| Smart Scraping | Simulate real browsers with rotated proxies and fingerprinted headers |
| API Gateway Testing | Ensure internal APIs are not blocking or rate-limiting test clients |
| Proxy Evaluation | Cycle through proxy lists to test speed, reliability, or geographic routing |
| Threat Detection Simulation | Simulate bot detection triggers (CAPTCHA, block pages) for security testing |
| Metrics-Driven Debugging | Use Prometheus metrics and logs to debug failed requests |

## **Functional Overview**

### **Key Technical Terms**

* **HTTPS request**: A secure web request made to a target URL using the HTTPS protocol.
* **Proxy server**: An intermediary server used to route web traffic to hide the original IP address.
* **TLS fingerprinting**: A method to mimic specific browser characteristics by customizing headers like User-Agent to avoid detection.
* **Prometheus metrics**: A monitoring system that collects and stores time-series data, which can be visualized in tools like Grafana.
* **JSONL (JSON Lines)**: A log format where each line is a separate JSON object for efficient parsing and streaming.
* **YAML (YAML Ain’t Markup Language)**: A human-readable configuration format commonly used in Python applications.
* **CI/CD**: Continuous Integration and Continuous Deployment; used to automate building, testing, and deploying applications.

### **1. Configuration**

* User defines:  
  + Target URLs
  + Proxy list (optional)
  + TLS profiles (user-agent headers)
* Location: config/config.yaml

### **2. Execution**

* Entry point: main.py
* User runs:

python main.py

* The system loads the configuration and initializes the core modules.

### **3. Request Cycle**

For each URL:

1. Select a TLS fingerprint (User-Agent header)
2. Rotate to the next available proxy (if provided)
3. Make an asynchronous HTTPS request
4. Classify the response (Success, CAPTCHA, Blocked, Unknown)
5. Log the request and response
6. Update Prometheus metrics counters

### **4. Logging**

* Format: JSONL
* Location: logs/logs.jsonl
* Events: request\_sent, response\_received

### **5. Metrics**

* Exposed via embedded Prometheus server
* URL: http://localhost:8000/metrics
* Metrics:
  + requests\_success\_total: Count of successful requests
  + requests\_failure\_total: Count of failed requests

## **Summary of Functional Flow**

User runs script

→ ConfigLoader loads YAML

→ RequestManager initializes modules

→ For each URL:

→ Rotate proxy (if any)

→ Select TLS profile

→ Make HTTPS request

→ Classify response

→ Log events

→ Update metrics

## **Functional Boundaries**

| **Boundary** | **Included** | **Not Included** |
| --- | --- | --- |
| TLS Spoofing | Header-based (User-Agent) | Low-level socket-level fingerprinting |
| Proxy Handling | Rotation, static list | Dynamic pool discovery |
| Metrics | Prometheus HTTP server | External exporters or dashboards |
| Logging | Structured logs (JSONL) | Log visualization interfaces |
| Scheduling | Manual or script-based | Built-in job scheduler |
| Retry/Backoff | To be implemented | Not available in current version |

## **Assumptions**

* End user has Python 3.8+ installed
* Network access is available (with or without proxy)
* User understands YAML syntax for configuration

## **Future Enhancements (Optional)**

* Add retry logic with exponential backoff
* Add CAPTCHA solver or CAPTCHA detection confidence
* Integrate scheduling and CRON support
* Add concurrency support for mass URL scans
* Provide CLI or GUI wrapper