# NLP Applications (AIMLCZG519)

## Assignment 2 - Sentiment Analysis Application

## Task B - RESTful API Enhancement Plan

**Group #37**

**Team Members**

**Ankit Kumar Agarwal** - 2024aa05560

**Chandrababu Yelamuri** - 2024aa05820

**Preety Gupta** - 2023ac05892

**Srithin Nair** - 2024ab05197

**Deepan KG** - 2024aa05755

## Table of Contents

## Executive Summary

This document presents a comprehensive enhancement plan to expose the functionality of the existing Sentiment Analysis Application through an enterprise-grade RESTful API. The objective of this enhancement is to enable secure, scalable, and standardized access to sentiment analysis capabilities for external applications, including web platforms, mobile clients, and third-party systems.

The current application provides a hybrid NLP-based sentiment analysis engine using VADER and TextBlob, supporting both single-text and batch processing through a web-based graphical user interface. While the system delivers accurate sentiment classification and detailed analysis outputs, it lacks critical enterprise features such as API-level authentication, rate limiting, asynchronous processing, and external system integration. These limitations restrict its usability in real-world, production-scale environments.

To address these gaps, the proposed architecture introduces a layered RESTful API design that separates access control, business logic, NLP processing, and infrastructure services. External clients interact with the system via an API Gateway, which acts as a single entry-point for request routing, load balancing, and secure communication. Authentication and authorization are enforced using JWT-based mechanisms and API keys, ensuring controlled and role-based access to system resources. Rate limiting and quota management are incorporated to prevent abuse, enforce fair usage, and support tier-based access models.

Overall, this enhancement plan transforms the Sentiment Analysis Application into a secure, scalable, and maintainable API platform that adheres to industry best practices.

# 1 Introduction

## 1.1 Purpose

The purpose of this enhancement is to allow external applications to securely consume sentiment analysis capabilities using standardized RESTful APIs.

## 1.2 Scope

- RESTful API design and layered architecture
- Authentication and authorization
- Rate limiting and quota enforcement
- Error handling and response standardization
- Monitoring, analytics, and scalability

# 2 Current State Analysis

## 2.1 Existing Capabilities

The existing system supports hybrid NLP sentiment analysis using VADER and TextBlob, with support for both individual input as well as batch processing. There is a web-based GUI with comprehensive information displayed for the analyzed text data, including the Sentiment, Confidence Score, Processed Tokens, Sentiment Score, etc.

In the batch-processing mode the user can upload a text file with one or more chunks of text, separated by newlines, and the application will do the sentiment analysis for each chunk individually and will also provide all analysis details on a per-chunk basis.

## 2.2 Limitations

- No API-level authentication
- No per-user rate limiting
- No external RESTful API exposure – this means other applications cannot use this application
- No asynchronous webhook-based processing

## 3 Proposed System Architecture

### 3.1 Architecture Overview

Figure 1 illustrates the proposed RESTful API architecture. The design follows a layered approach, separating access control, business logic, NLP processing, and infrastructure services.

The system exposes sentiment analysis capabilities via RESTful APIs. External clients interact with the system through an API Gateway, after which requests pass through authentication, rate limiting, and processing layers before reaching the NLP engine. Both synchronous and asynchronous processing models are supported.

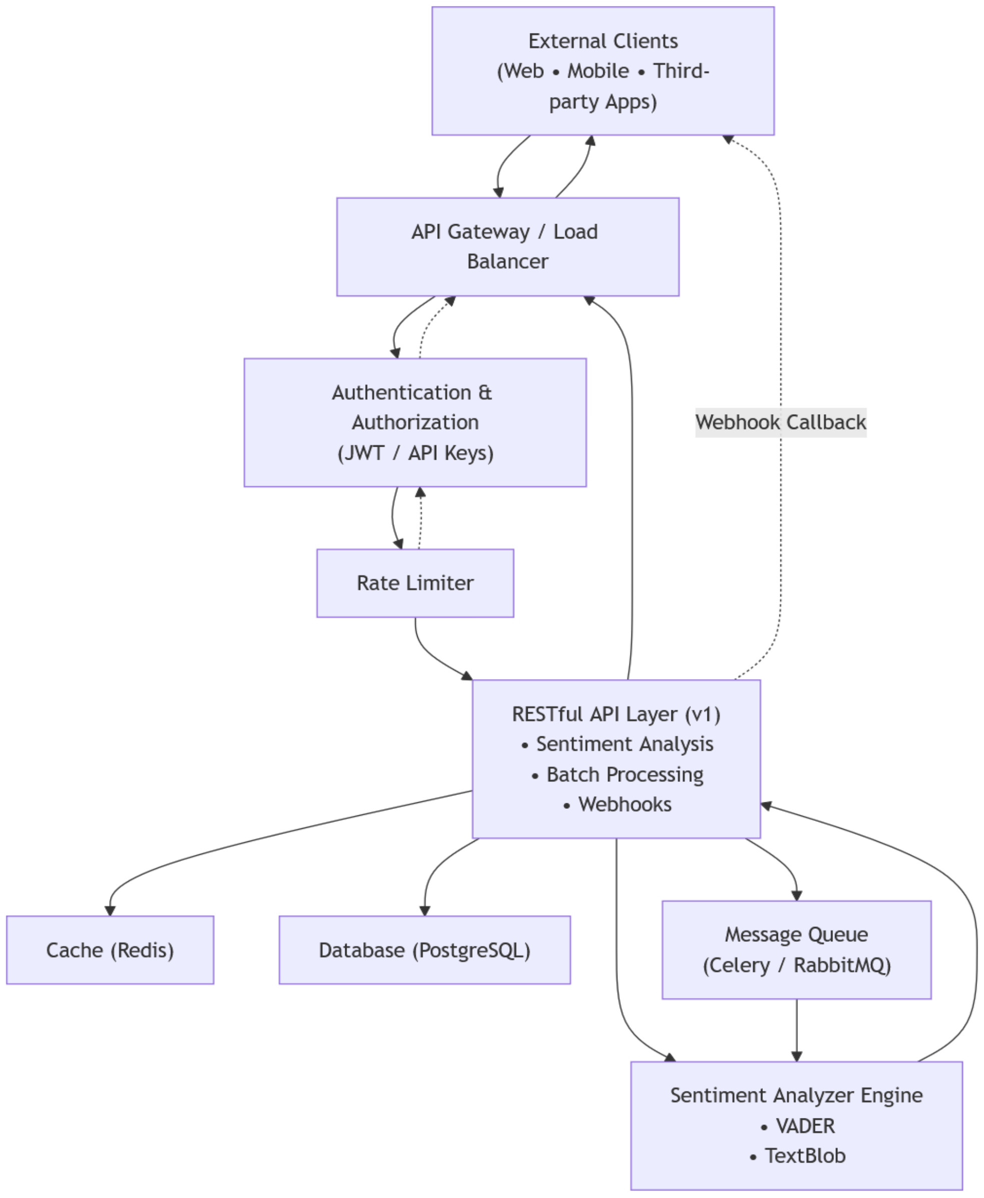
External Clients (Web • Mobile • Third-party Apps)
API Gateway / Load Balancer
Authentication & Authorization (JWT / API Keys)
Rate Limiter
Webhook Callback
RESTful API Layer (v1) • Sentiment Analysis • Batch Processing • Webhooks
Cache (Redis)
Database (PostgreSQL)
Message Queue (Celery / RabbitMQ)
Sentiment Analyzer Engine • VADER • TextBlob

*Figure 1: Proposed RESTful API Architecture*

## 3.2 External Clients

External clients may include web applications, mobile applications, and third-party services. They will interact with the application system using HTTP-based RESTful API calls. There will be provision for both individual and batch requests to be analyzed by the system.

**Responsibilities:**

- Send sentiment analysis requests
- Receive synchronous responses for real-time analysis
- Register webhook endpoints for asynchronous batch processing results

## 3.3 API Gateway / Load Balancer

The API Gateway serves as the single entry point, handling request routing, load balancing, and SSL termination.

**Responsibilities:**

- Request routing and load balancing
- SSL/TLS termination
- Forwarding requests to internal services
- Protecting backend services from direct exposure

## 3.4 Authentication & Authorization Service

This service validates client identity using JWT tokens or API keys and enforces role-based and scope-based access control.

**Responsibilities:**

- JWT-based authentication for client requests
- API key validation for server-to-server communication
- Token validation and identity propagation
- Scope-based access control

## 3.5 Rate Limiting & Quota Management

Rate limiting protects the system from abuse by enforcing per-user and per-tier request quotas. There will be costs associated with different tiers and users will need to register based on their specific needs.

**Responsibilities:**

- Enforce per-user and per-tier request limits

- Apply burst and sustained rate limits
- Return quota-related response headers
- Protect system stability under high load

A sample is shown below:

| Tier | Requests/Hour | Requests/Day | Batch Size | Cost |
|---|---|---|---|---|
| Free | 10 | 100 | 5 | Rs 0 |
| Basic | 100 | 1000 | 50 | Rs X/month |
| Professional | 500 | 10000 | 500 | Rs Y/month |
| Enterprise | Unlimited | Unlimited | Unlimited | Custom |

### 3.6 RESTful API Layer

This layer exposes sentiment analysis, batch processing, model access, and webhook management endpoints.

**Responsibilities:**

- Handle sentiment analysis requests
- Manage batch processing jobs
- Expose webhook registration endpoints
- Validate input and standardize responses
- Orchestrate calls to NLP and infrastructure components

### 3.7 Sentiment Analyzer Engine

The sentiment engine performs NLP processing using VADER and TextBlob, including text preprocessing and sentiment scoring.

**Responsibilities:**

- Text preprocessing and normalization
- Sentiment computation using VADER and TextBlob
- Confidence score calculation
- Support for future model extensions (e.g., BERT)

### 3.8 Cache Layer (Redis)

Redis is used for caching frequent requests and maintaining rate-limit counters to improve performance.

**Responsibilities:**

- Cache frequently requested sentiment results
- Store rate-limiting counters
- Reduce database load
- Improve API response latency

### 3.9 Database Layer (PostgreSQL)

PostgreSQL stores persistent data such as user credentials, batch job metadata, and webhook configurations.

**Responsibilities:**

- Store user credentials and API keys
- Maintain batch job metadata
- Store webhook configurations
- Persist audit logs and request metadata

### 3.10 Message Queue & Asynchronous Processing

Asynchronous batch jobs are handled through a message queue and background workers, allowing the API to remain responsive.

**Responsibilities:**

- Queue long-running batch sentiment jobs
- Enable background processing using workers
- Improve system responsiveness
- Decouple request handling from heavy computation

### 3.11 Synchronous vs Asynchronous Communication

The system supports two communication patterns:

- **Synchronous Processing**
  Used for real-time sentiment analysis. Responses are returned over the same HTTP request-response cycle.
- **Asynchronous Processing**
  Used for batch requests. Clients receive a job ID immediately, and results are delivered later via webhook callbacks.

### 3.12 Batch Processing Sequence Diagram

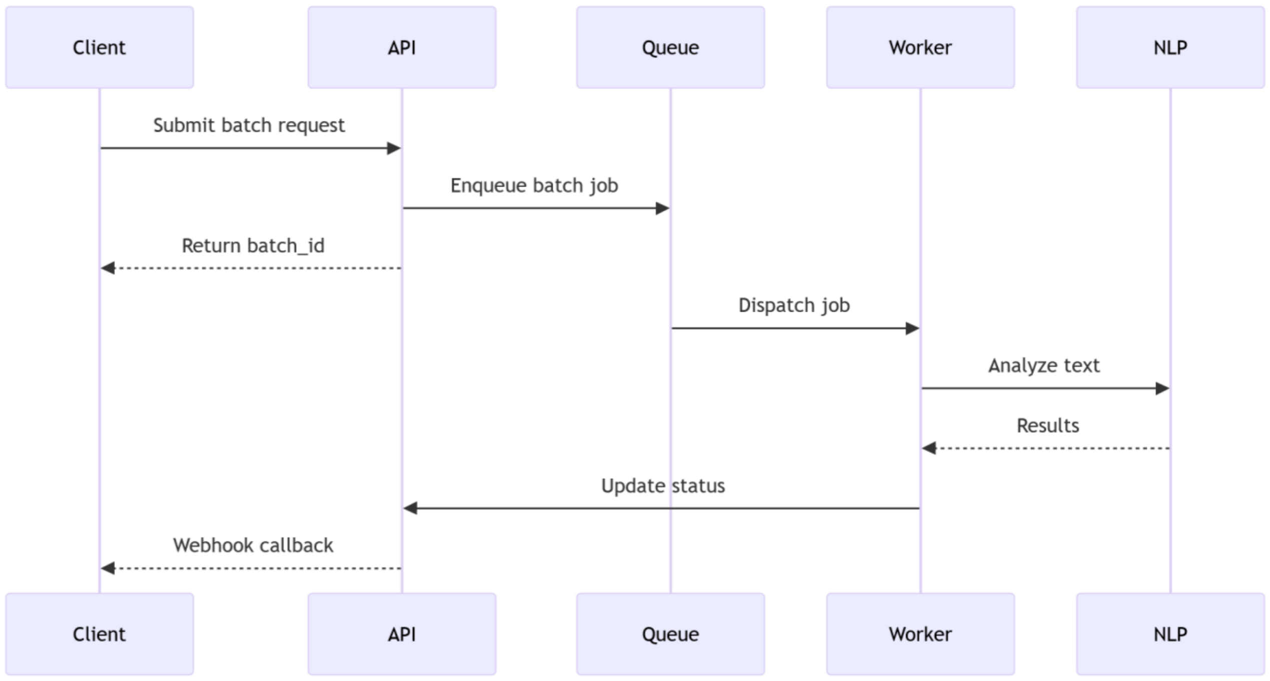

*Figure 2: Sequence Diagram for Batch Sentiment Analysis*

## 4 Monitoring & Analytics

- **Metrics to Track**
  System performance will be tracked against the below metrics that will be monitored on a continuous basis:
  - API response time
  - Request throughput
  - Error rate
  - Cache hit ratio
  - Batch processing queue size
  - User engagement metrics
- **Alerting Rules**
  System will send appropriate alerts when performance metrics indicate degradation. Some typical conditions for such alert could be:
  - Response time > 1000ms
  - Error rate > 5%

    - Queue size > 1000 items
    - Webhook delivery failures

## 5 Success Metrics

- API uptime > 99.9%
- Average response time < 200ms
- Developer adoption (SDK downloads, API calls)

## 6 Phased Implementation Plan

### 6.1 Phase 1: Core NLP & UI Optimization (COMPLETED)

- Implement hybrid sentiment logic (VADER + TextBlob + Keywords).
- Add multi-paragraph batch detection for file uploads.
- Standardize internal JSON response formats.
- Create 3-column interactive responsive dashboard.
- Implement loading indicators and summary stats.

### 6.2 Phase 2: Security & Rate Limiting

- Set up Flask-JWT-Extended for token authentication.
- Implement per-client request throttling with Flask-Limiter.
- Standardize HTTP error codes (401, 403, 429).

### 6.3 Phase 3: Documentation & Testing

- Write comprehensive API documentation
- Write unit tests
- Write integration tests

### 6.4 Phase 4: Deployment & Monitoring

- Set up logging and monitoring
- Deploy for staging and production
- Continuous Performance monitoring

## 7 Conclusion

This enhancement plan transforms the Sentiment Analysis Application into an enterprise-grade API platform. By implementing industry-standard practices for authentication, rate limiting, error handling, and monitoring, the API will be secure, scalable, and maintainable.

The phased approach allows for iterative development and testing while the comprehensive documentation ensures good developer experience.