Question 5:

Scenario: A free finance news API allows 10 calls per minute. Each API call would take in a financial keyword (ie. inflation) and would take 5 seconds to process its sentiment analysis to be sent to the end-user. As the number of end-users increases, the frequency of calls increases greatly.

Task: Suggest one or more implementations that would allow the end-user to still receive the requested info with lesser disturbances or delay.

=>To optimize the utilization of the free finance news API while minimizing disturbances or delays for end-users, we can implement the following strategies:

Rate Limiting and Queueing:

1. Utilize Flask as the web framework.
2. Implement rate limiting using the Flask-Limiter extension.
3. Employ Celery with RabbitMQ or Redis as the backend to queue incoming requests, ensuring compliance with the API's rate limit of 10 calls per minute.

Caching:

1. Employ the cachetools library for in-memory caching or redis-py for caching with Redis.
2. Decorate API endpoints with caching mechanisms provided by these libraries to store sentiment analysis results for commonly queried financial keywords, reducing the number of API calls.

Asynchronous Processing:

1. Utilize Celery for asynchronous task execution.
2. Define tasks for processing API calls asynchronously, allowing multiple requests to be handled concurrently without being blocked by the API's rate limit.

Throttling and Prioritization:

1. Implement custom middleware or decorators to throttle requests based on user attributes or request characteristics, ensuring essential information is delivered promptly.
2. Prioritize requests within the application logic based on predefined criteria, such as user importance or urgency.

Usage Quotas for End-users:

1. Track and enforce usage quotas using a database like MongoDB or Redis, preventing any single user from overwhelming the system with excessive requests.
2. Implement middleware or decorators to enforce per-user usage quotas, ensuring fair usage of the API resources.

Load Balancing and Scaling:

1. Utilize Docker for containerization and Kubernetes for orchestration.
2. Deploy the application across multiple containers and use Kubernetes to manage scaling and load balancing, ensuring efficient handling of increasing traffic.

Error Handling and Retry Mechanism:

1. Implement retry logic using the retrying library or built-in retry mechanisms in libraries like requests, ensuring robust error handling in case of API rate limit exceedance or unavailability.
2. Utilize exception handling to gracefully handle API errors and retries, ensuring uninterrupted service for end-users.

By implementing these strategies, we can efficiently manage the usage of the finance news API, reduce disturbances for end-users, and ensure timely delivery of requested information, even as the number of end-users increases.