Part 2:

“Provide design documentation to operationalize the collection of billions of URLs using the code developed. Propose the next steps, and how to further optimize for cost, reliability, performance, and scale.

Input: List of billions of URLs send in via a text file and/or in MySQL for a given year month e.g. billions of URLs for amazon.com, walmart.com, bestbuy.com etc for July

Output: Design storage of the metadata and content. Design for unified data schema. Design for configurability, politeness and respect Robots.txt. Define SLOs and SLAs. Elaborate on the key monitoring metrics and tools you would employ to effectively track the system's progress.”

System Architecture Overwview

1. URL ingestion
    - Provide a list of URLs from text files or MySQL database
    - Use Kafka for the stream of input URLs

2. Distributed Crawlers
    - Puppeteer-based crawler wrapped in a microservice architecture (e.g., Node.js with Docker)

3. Metadata Storage
    - Define Unified Schema for metadata for consistency

4. Content Storage
    - Amazon S3 bucket

5. Database
    - DynamoDB

6. Queue Management
    - Kafak RabbitMQ for task scheduling and load balancing

7. Monitoring & Alerting
    - Tool: Prometheus and Grafana for monitoring
    - Metrics: Crawl success rate, average response time, error rates, and resource utilization

Compliance and Configurability

- Respect for Robots.txt:

    - Implement checks to adhere to the rules specified in robots.txt for each domain
    - Use libraries like robots-parser to parse and respect these rules

- Politeness Configuration:
    - Implement configurable delay between requests to the same domain.
    - Use rotating IPs and user agents to avoid detection and blocking

SLOs and SLAs:

1. Service Level Objectives (SLOs):
    - 99.9% uptime.
    - Response time < 200ms for metadata retrieval.
    - 95% success rate in URL crawling

2. Service Level Agreements (SLAs):
    - Detailed agreements with stakeholders on expected service performance and reliability

Key Monitoring Metrics and Tools

1. Metrics:
    - Crawl Success Rate: Percentage of URLs successfully crawled
    - Average Response Time: Time taken to fetch metadata
    - Error Rates: HTTP errors, parsing errors, etc.
    - Resource Utilization: CPU, memory, network usage
    - Throughput: Number of URLs processed per second

2. Tools:
    - Prometheus: For collecting and storing metrics
    - Grafana: For visualizing metrics and setting up alerts
    - ELK Stack (Elasticsearch, Logstash, Kibana): For log aggregation and analysis

Part 3

“Break down on how to proceed with engineering to Proof of Concept. What are the list of potential blockers. What are known and trivial and
what are the estimates arrival time. Implementation schedules. How to have a successful and highly quality release.
Output: Documentations on how to proceed to next steps. Documentations of how to evaluate the proof of concept. Describe your plan for
distributing ownership and responsibilities of this service among team members. Release plan. Resources and time estimations.”

Engineering Steps to PoC

1. Initial Setup:
    - Set up the development environment.
    - Deploy core crawler on a small scale.

2. Scalability Testing:
    - Implement distributed crawling
    - Test on a larger dataset (First with millions of URLs)

3. Integration:
    - Integrate Kafka for URL ingestion
    - Set up metadata storage

4. Compliance and Politeness:
    - Implement respect for robots.txt.
    - Configure polite crawling mechanisms

5. Monitoring:
    - Set up Prometheus and Grafana for monitoring.
    - Implement logging

Potential Blockers and Solutions

1. Blockers:
    - IP blocking by target sites.
    - Rate limiting by target sites.
    - Handling JavaScript-heavy pages.

2. Solutions:
    - Use rotating IPs and proxies.
    - Implement rate limiting on the crawler side.
    - Use headless browsers like Puppeteer for JS-heavy pages.

Implementation Schedule

1. Week 1-2: Environment setup and initial crawler deployment.
2. Week 3-4: Implement distributed crawling and scalability tests.
3. Week 5-6: Integrate URL ingestion and storage systems.
4. Week 7-8: Implement compliance, politeness, and monitoring.
5. Week 9-10: Conduct end-to-end testing and optimization.

Distribution of Responsibilities

- Team Lead: Oversee the project, and ensure milestones are met.
- Software Engineers: Develop and optimize the crawler, and integrate it with storage systems.
- Cloud Engineers: Set up infrastructure, monitoring, and deployment pipelines.
- Data Engineers: Design and implement data schemas, and manage databases.
- QA Engineers: Conduct testing, and ensure compliance with SLAs.

Evaluation of PoC

1. Success Criteria:
    - Ability to crawl and store metadata for millions of URLs
    - System stability and performance under load
    - Compliance with robots.txt and politeness policies
    - Effective monitoring and alerting
2. Evaluation Metrics:
    - Crawl success rate
    - System uptime and response times
    - Error rates and resource utilization

Release Plan

1. Beta Release:
    - Limited rollout to test with a smaller subset of URLs.
    - Gather feedback and make necessary adjustments.
2. Full Release:
    - Rollout to full scale with all intended URLs.
    - Continuous monitoring and optimization.

Resources and Time Estimations

1. Human Resources:
    - 2 Software Engineers
    - 1 Cloud Engineer
    - 1 Data Engineer
    - 1 QA Engineer
    - 1 Project Manager

2. Time Estimations:
    - Development: 6-8 weeks
    - Testing: 2-3 weeks
    - Deployment: 1 week
    - Total: 9-12 weeks

Following this plan, you can scale the web crawler to handle billions of URLs, ensuring efficient and reliable metadata extraction, storage, and monitoring.