Backend Assignment April 2025

Backend Assignment: Webhook Delivery Service

Objective:

Your mission is to design and build a robust backend system that functions as a reliable webhook delivery service. This service will ingest incoming webhooks, queue them, and attempt delivery to subscribed target URLs, handling failures with retries and providing visibility into the delivery status.

Core Requirements:

1. **Subscription Management:**
    - Implement API endpoints for CRUD (Create, Read, Update, Delete) operations on webhook subscriptions.
    - A subscription should define at least:
        - A unique identifier for the subscription.
        - The target URL where the webhook payload should be delivered (POST request).
        - *(Optional but recommended)* An optional secret key for payload signature verification (see Bonus Points).
2. **Webhook Ingestion:**
    - Implement an API endpoint (e.g., `/ingest/{subscription_id}`) that accepts incoming webhook payloads via HTTP POST requests (assume JSON body).
    - Upon receiving a payload, the service should quickly acknowledge the request (e.g., return a `202 Accepted`) and queue the delivery task for asynchronous processing.
3. **Asynchronous Delivery Processing:**
    - Implement background workers to process the queued delivery tasks.
    - For each task, the worker should:
        - Retrieve the corresponding subscription details (target URL, secret).
        - *(Optional: Implement signature verification if secret is present)*.
        - Attempt to deliver the original JSON payload via an HTTP POST request to the subscription's target URL. Set a reasonable timeout (e.g., 5-10 seconds).
4. **Retry Mechanism:**
    - If a delivery attempt fails (e.g., non-2xx response from the target URL, network error, timeout), the service must automatically retry the delivery.
    - Implement a retry strategy with exponential backoff (e.g., retry after 10s, 30s, 1m, 5m, 15m).
    - Define a maximum number of retry attempts (e.g., 5 attempts). If all retries fail, mark the delivery as failed.
5. **Delivery Logging:**
    - Log the status of each delivery *attempt* (including retries).
    - Logs should contain:

        - Identifier for the original incoming webhook/delivery task.
        - Subscription identifier.
        - Target URL.
        - Timestamp of the attempt.
        - Attempt number (1 for initial, 2+ for retries).
        - Outcome (Success, Failed Attempt, Failure).
        - HTTP status code received from the target (if applicable).
        - Error details on failure (if applicable).

6. **Log Retention:**
    - Implement a data retention policy for the delivery attempt logs. Keep logs for a defined period (e.g., **72 hours**).
    - Use background tasks to periodically delete logs older than the retention period.
7. **Status/Analytics Endpoint:**
    - Provide an API endpoint to retrieve the status and recent delivery attempt history for a specific original incoming webhook ~~or delivery task ID~~.
    - Provide an API endpoint ~~to list recent~~ delivery attempts (e.g., last 20) for a specific *subscription*.
8. **Caching:**
    - Implement caching to optimize performance, especially during webhook ingestion and processing. Key areas:
        - Cache subscription details (target URL, secret) based on `subscription_id` to avoid frequent database lookups in workers.

Technical Requirements:

- **Language:** Use either **Python** or **Go**. Choose the language you are most comfortable with.
- **Framework:** You are free to choose any web framework suitable for your chosen language (e.g., Flask, Django, FastAPI for Python; Gin, Echo for Go).
- **Database:** Choose a suitable database (SQL or NoSQL). Your choice should reflect considerations for storing subscriptions and a potentially large volume of delivery logs. Explain your choice in the README.
- **Asynchronous Tasks / Queuing:** Implement a robust mechanism for queuing webhook deliveries and handling background tasks. This is critical for ingestion decoupling, retries, and log cleanup.
- **Containerisation:** The entire application (API, background workers, database, Redis, queue if applicable) **must** be containerised using **Docker** and orchestrated using `docker-compose`. The setup should work locally with just Docker installed. Assume no other software will be installed on the machine. Also all testing will be done on a mac device.
- **Deployment:** Deploy the application to a **free tier** of any public cloud provider (e.g., AWS EC2/ECS Fargate, Google Cloud Run/App Engine, Heroku, Render).

Presentation (UI):

- A minimal UI is required to interact with the service (manage subscriptions, view delivery status/logs for a subscription).

- This can be:
  - A simple custom web UI.
  - A generated UI via Swagger/OpenAPI documentation.
- Focus on functionality, not aesthetics.

Deliverables:

- A link to a **private GitHub repository** containing your source code. Add `shobhit@segwise.ai` and `chinmay@segwise.ai` as collaborators. Please do not create an empty repo and and us. Only do it when you are ready to submit.
- A comprehensive **README.md** file in the repository containing:
  - Clear, step-by-step instructions on how to set up and run the application locally using Docker. **Verify these instructions work.**
  - The link to the live, deployed application.
  - Explanation of your architecture choices (framework, database, async task/queueing system, retry strategy, etc.).
  - Discussion on database schema and indexing strategies.
  - Sample `curl` commands or equivalent demonstrating how to use each API endpoint (managing subscriptions, ingesting webhooks, checking status).
  - An estimation of the monthly cost to run the deployed solution on the chosen free tier, assuming continuous operation (24x7) and moderate traffic (e.g., 5000 webhooks ingested/day, average 1.2 delivery attempts per webhook).
  - Any assumptions made.
  - Credits for any significant libraries, external resources, or AI tools used.

Evaluation Criteria (in order of importance):

1. **Completeness & Correctness:**
   - All core requirements implemented (Subscription CRUD, Ingestion, Async Delivery, Retries, Logging, Retention, Status API).
   - Deployed solution is functional.
   - Webhook delivery, retries, logging, and retention work correctly.
   - Caching (Redis) is implemented effectively.
   - Async tasks/queuing are handled reliably.
2. **Documentation & Setup:**
   - README is clear, comprehensive, and accurate.
   - Local setup instructions work flawlessly using Docker.
   - API documentation/examples are provided.
3. **Code Quality & Maintainability:**
   - Code is clean, well-structured, readable, and follows language best practices.
   - Appropriate use of comments.
   - Meaningful variable and function names.
4. **Tests:**

    - Includes unit and/or integration tests covering critical logic (e.g., subscription management, queueing, delivery attempts, retries, status checks).
5. **Performance & Scalability Considerations:**
    - Demonstrates awareness of performance (e.g., indexing, caching, efficient queuing).
    - No obvious performance bottlenecks for the specified requirements.
6. **Cost Effectiveness:**
    - Solution utilizes free-tier resources effectively.
    - Cost estimation is provided and reasonable.
7. **Minimal UI:**
    - UI allows basic interaction and testing of features.

Bonus Points:

- **Payload Signature Verification:** Implement verification of incoming webhooks using a standard method (e.g., checking an `X-Hub-Signature-256` header containing an HMAC-SHA256 hash of the payload, using the stored subscription secret) (+2 points). The ingestion endpoint should reject payloads with invalid signatures.
- **Event Type Filtering:** Allow subscriptions to specify specific event types they are interested in (e.g., `order.created`, `user.updated`). The ingestion endpoint should accept an event type (e.g., in a header or query param), and only deliver webhooks to subscriptions matching that event type (+2 points).

Submission:

- Use the form link in the email to submit. Provide your git repo link, deployment link.
- Please use same email id in all communications.

Note: You are encouraged to use online resources, libraries, and tools. Please provide attribution in your README. Make reasonable assumptions if any requirement is unclear, and document these assumptions.

Good luck, and happy building!