Backend Assignment Oct 2025

Your mission if you chose to accept is to build a production-quality, containerized Event Trigger Platform where customers create, manage, and run triggers.

Please read the whole document before you start building. Answers to your questions are most likely already in the document.

WHAT TO BUILD — Requirements

1) Trigger types & lifecycle

- Support **two trigger types**:
  - **Scheduled triggers** — support one-time and recurring schedules; allow both absolute times and interval-based schedules (e.g., at `2025-10-15T14:00:00Z`, or every 30 minutes, or "in 10 minutes").
  - **API triggers** — At time of creating these, users will provide a json schema and an endpoint. API should return a webhook url which when hit with a post call, validates the payload against provided schema and all sends the payload as a post to the provided endpoint.
- Support a **manual/test run** mode for both types that fires **once** and is **not persisted** as a trigger.

2) Trigger management

- CRUD for non-test triggers (Create, Read/List, Update, Delete).
- Deleting a trigger must **not** delete previously fired event logs (until they expire per retention policy).
- Editing affects **future** firings only.

3) Event logging and retention

- Every fired event creates an **Event Log** containing:
  - Unique event id, associated trigger id (or null for pure test runs), trigger type, fired timestamp, payload (if any), source (api/scheduled/manual-test), status (success/failure), and a note if it was a test/manual run.
- **Retention lifecycle**:
  1. **Active**: first 2 hours after fired time (readable in default view).
  2. **Archived**: next 46 hours (still retrievable via an "archived" view).

   3. **Deleted**: permanently removed after 48 hours total.
- The system must implement the archival and deletion behavior reliably (not just soft-delete flags).

4) Scheduling accuracy & concurrency

- The actual execution of scheduled triggers must occur **no later than 10 seconds** after the intended scheduled time.
- The system must tolerate concurrent firings (multiple triggers firing at same time) and remain correct.
- The system must handle restarts and worker failures without losing events and must avoid duplicate event logs across retries (idempotency/dedupe).

5) APIs & interaction

- Provide APIs to:
    - Create, list, get, update, delete triggers.
    - Fire an API trigger (POST to trigger endpoint).
    - Start a manual/test run for scheduled or API triggers (ephemeral).
    - List Event Logs (default: active events — last 2 hours; optional switch to archived).
    - Retrieve event details.
- Document the API (OpenAPI/Swagger and clear README examples). A minimal UI is optional — swagger or openapi ui suffice.

6) Async, queue, and workers

- Use an **asynchronous job mechanism** so that firing is performed by a worker/process separate from the request thread (candidates pick implementation).
- The job system must support scheduling (delayed/scheduled delivery), retries (with backoff), and idempotency for event creation.
- There must be a background mechanism to transition events from active → archived → deleted.

7) Persistence & durability

- Use a durable data store (relational or nosql or any other type of DB) for triggers and event logs such that data survives process restarts.
- Decide on DB and schemas with read-heavy event log access in mind.

8) Observability & health

- Provide health checks for core components (web app, worker(s), queue, DB).
- Provide logs sufficient to debug missed triggers or duplicates.

- Expose simple metrics (counts of pending jobs, processed jobs, events active/archived) in some form (API endpoint or metrics endpoint).

9) Containerization and deployment

- The solution **must be containerized** and runnable locally with a single command (e.g., `docker compose up`).
- Anything else needed should be done as part of docker startup.
- Provide a deployed instance on a **free public cloud** (any free tier of provider of your choice). Add the link in README.
- Do not assume reviewer has preinstalled services beyond Docker. Only docker commands will be run as part of evaluation. No additional software can be installed.

10) Documentation & reproducibility

- Clear README with:
  - How to run locally and run tests.
  - How to access the deployed instance.
  - API usage examples (curl/HTTPie).
  - Design decisions and trade-offs (why you chose your queue, DB, scheduling approach).
  - Any limitations and how you'd improve the system with more time.

Non-functional expectations

- **Durability**: system must not lose events during restarts.
- **Correctness**: events must be reliably logged and obey retention lifecycle.
- **Timing**: scheduled triggers must fire within the 10s window under normal load.
- **Concurrency**: system should handle multiple simultaneous trigger firings without race conditions or double-logging.
- **No hidden assumptions**: repository must include everything needed to run (no private keys, no external paid resources).

Required automated tests

Automated tests must cover key behaviors (framework of your choice). Include tests for:

1. **Create + Fire (API trigger)**:
   - Create API trigger with expected payload structure; POST a valid payload → event log created with payload.
   - POST an invalid payload → validation error and no event log.

2. **Scheduled trigger accuracy**:
    - Create one-shot scheduled trigger that fires reasonably soon (tests may use accelerated timings or test configuration). Assert firing occurred within 10s of schedule.
3. **Manual/test runs**:
    - Trigger a manual/test scheduled run and assert it fires once and is not persisted.
4. **Retention lifecycle**:
    - Simulate or accelerate time to verify active → archived → deleted transitions (or provide test configuration to shorten retention and prove transitions work).
5. **Idempotency / duplicate protection**:
    - Simulate worker retry or duplicate delivery and assert at-most-once event log (or clearly documented dedupe behavior).
6. **Concurrent firing**:
    - Fire multiple triggers simultaneously and assert all events logged correctly and within timing SLAs.

Tests do not need to simulate high production load but must demonstrate correctness of logic.

Deliverables (must)

- Public Git repository with source code.
- Docker configuration (`Dockerfile` + `docker-compose.yml`) enabling local run.
- README with setup, API docs, deployment link, design notes.
- Automated tests and instructions to run them.
- (Optional) lightweight UI or documented curl examples for manual testing.

Evaluation rubric (high-level, for reviewers)

Total: **40 points**

1. **Functional completeness — 12 pts**
    - CRUD + trigger creation and API firing: 4
    - Manual/test runs implement correctly: 2
    - Event logs + retention lifecycle: 6
2. **Correctness & timing — 8 pts**
    - Scheduled events fire within 10s SLA in tests: 4
    - Idempotency/duplicate handling, worker retries: 2
    - Concurrent firing correctness: 2
3. **Architecture & implementation quality — 8 pts**
    - Clean separation of web/worker, clear job flow: 4
    - Persistence and data modeling, indexing choices: 2
    - Observability & health endpoints: 2

4. **Code quality & tests — 6 pts**
    - Readable, modular, documented code: 3
    - Automated tests that exercise key behavior: 3
5. **Deployment & reproducibility — 4 pts**
    - Containerized + runs locally via single command: 2
    - Deployed to free public cloud + link in README: 2

Bonus (up to 6 pts)

- Simple authentication for management APIs (API keys or equivalent): 2 pts
- Aggregated event logs by trigger (counts in last 48 hours as toggle) and access to individual events: 2 pts
- Metrics endpoint (Showing counts and latencies for triggers) + basic dashboard or clear json schema response: 2 pts

---

Constraints & guidance (do not over-specify)

- **Do not** hardcode vendor-specific services requiring paid accounts in CI or deployment.
- You may use open-source libraries or services, but document why and how they are used.
- Focus on correctness and reliability over adding many peripheral features.
- You can use any online resource at your disposal like google, chatgpt, open libraries for completing the assignment . But please give credits in your README on whatever tools/help you used to complete the assignment. We like people who are resourceful but also honest.
- Make sure you understand what you are doing as we would ask you to explain your choices in next rounds.
- If anything is not clear in the assignment, make a reasonable assumption(mention it in the README as assumptions) and move forward.
- Do not use any hosted versions of DB or redis. Everything should be locally setup.

Submission

- **Language for the code has to be python or golang**. This is what you will be using primarily at Segwise, hence we want to see how comfortable you are in these languages. Choice of web framework is yours.
- API design, what fields to take etc is your choice.
- Which DB to use is left for you to decide.
- How/If to cache things to speed up your solution is your decision.
- Submit **link to the github project**. No zip files in google drive or email would be accepted. Please make sure repository is private. **Add shobhit@segwise.ai as collaborator** to your

github project. Any documentation or instructions you want to provide should be in a README in the repo itself. Please add us only after your solution is ready.

- Also submit the **link to the deployed solution** in README itself. Please use a free tier or a free service for deployment. There is no reimbursements for any cloud cost incurred.
- Please also submit sample api request/response for the apis in the README file.
- **README should also contain information on how to setup and run the submission locally. Please test this before submitting that your instructions work. The instructions will be followed exactly as per your documentation.** Due to the volume of submissions we will not be able to reach out to fix any issues later on. So please make sure the instructions work.
- Submission needs to be made within 48 hours of being given the assignment or as communicated in the message. We will not be able to accept in delayed assignments. So if you submit after deadline, we might not be able to provide a detailed feedback.
- Github profile names might not match in some cases to the email you used to submit your application. To avoid any confusion, please fill the google form provided as well: https://forms.gle/EtefJnvpVYqHTDp66

Happy building!!