

Ebra Back End Challenge: AI Call Orchestrator Service

📖 1. Context

You're building the backend "orchestrator" for an AI-driven calling platform. Clients enqueue call requests; a pool of workers invokes an external AI-call API, tracks progress, retries on failure, and enforces a maximum of **30 concurrent calls**.

🔍 2. Assumptions & Schema

- **Persistence:** PostgreSQL
- **Queue:** Kafka or alternative durable queue (optional)
- **Locking/State:** Redis (optional)
- **External AI-Call API:**
 - **Endpoint:** `POST /api/v1/calls` returns a `callId` and accepts a `webhookUrl` for callbacks.
 - **Callback:** provider calls back to `/callbacks/call-status` with final status.

```
interface Call {
  id: UUID;                // PK
  payload: {
    to: string;             // e.g. "+966501234567"
    scriptId: string;       // identifier of call script / flow
    metadata?: Record<string, any>;
  };
  status: 'PENDING'        // newly created
    | 'IN_PROGRESS'        // worker picked up
    | 'COMPLETED'         // API eventually succeeded
    | 'FAILED'             // retries exhausted
    | 'EXPIRED';           // optional: aged-out before processing
  attempts: number;        // retry counter
  lastError?: string;      // last failure message
}
```

```
createdAt: timestamp;  
startedAt?: timestamp;  
endedAt?: timestamp;  
}
```

3. Functional Requirements

3.1. HTTP API

1. Create Call

- `POST /calls`
- Body: `{ to: string; scriptId: string; metadata?: {...} }`
- Server sets `status='PENDING'`, `attempts=0`, returns full `Call` record.

2. Fetch Call

- `GET /calls/:id` → returns the `Call`.

3. Update Call

- `PATCH /calls/:id` → update **payload only** if `status==='PENDING'`.

4. List by Status

- `GET /calls?status=PENDING|IN_PROGRESS|FAILED|COMPLETED` → paginated.

5. Metrics Endpoint

- `GET /metrics` → JSON or Prometheus-style counts of calls per status.

3.2. Worker Service

- **Concurrency Limiter:** only **30** calls may be `IN_PROGRESS` at any moment..

Fetch & Lock

```
-- Atomically fetch one PENDING call and mark IN_PROGRESS
UPDATE calls
SET status='IN_PROGRESS', startedAt=NOW()
WHERE id = (
  SELECT id FROM calls
  WHERE status='PENDING'
  ORDER BY createdAt
  LIMIT 1
  FOR UPDATE SKIP LOCKED
)
RETURNING *;
```

-

Invoke AI-Call API

```
POST https://provider.com/api/v1/calls
Content-Type: application/json
```

```
{
  "to": "+966501234567",
  "scriptId": "welcomeFlow",
  "webhookUrl": "https://our-service.com/callbacks/call-status"
}
```

-

- On **202 Accepted**, record returned `callId` in a local mapping if needed.

- **Completion Detection :**

Webhook (Push)

Provider calls your `POST /callbacks/call-status` with:

```
{
  "callId": "8a7f2c1e-1234-4b9b-9d4e-abcdef012345",
  "status": "COMPLETED",          // or "FAILED", "BUSY", "NO_ANSWER"
  "durationSec": 42,
  "completedAt": "2025-07-20T10:00:42Z"
```

```
}
```

-
- You verify authenticity (HMAC/API key), then update `calls` table:
`status=COMPLETED|FAILED, endedAt=completedAt.`
- **Retries on Failure**
 - If network error or 5xx:
 - Increment `attempts`.
 - If `attempts < 3`: re-enqueue with back-off.
 - Else: set `status='FAILED'`, record `lastError`.
- **Release Semaphore Slot**
 - On terminal state (`COMPLETED` or `FAILED`), free up one of the 30 slots so another call can start.
- The system shall allow up to 30 AI calls to run in parallel overall, while ensuring **that no more than one AI call for the same entity** (person or phone) is ever in flight at once.

4. Non-Functional Requirements

- **Tech Stack:**
 - Node.js (v18+), **TypeScript**
 - HTTP framework: Express (or as you like)
 - Queue client: Kafka.js (or as you like)
 - PostgreSQL

- **Containerization (optional):**
 - Provide `docker-compose.yml` to spin up the Node service:
 - All configuration via environment variables (`.env.example`).
-

5. Deliverables

1. **GitHub Repository URL.**
-

6. Evaluation Criteria

Aspect	What We Look For
Correctness	Never double process; respects 30 concurrent limit
Reliability	Retries/back-off work; FAILED vs. COMPLETED clear
Code Quality	Modular, clean, well-typed TypeScript