# Take-Home Challenge

**Role:** Senior Engineer (TPO)

**Timebox:** Aim for **4–6 hours** of focused work. Submit within **five calendar days** of receiving this brief.

**What to submit:** Git repo link (or ZIP) + short technical README.

---

## 1) Overview

Own a minimal vertical slice end-to-end: design and implement a small backend service.

Optimize for conceptual clarity, clean code, tests, and pragmatic trade-offs.

Treat this as if you're the **Technical Project Owner**.

Infrastructure can be simple (local/in-memory), but structure the code so that moving to AWS services and a persistent DB is straightforward.

## 2) Scenario

A ticketing client needs an **Event Service** that:

1. **Creates and updates events** via an internal API with basic authentication.
2. **Queries events** by date range and/or multiple locations.
3. **Sends notifications** when events are created or change status (mocked, async).
4. Exposes a **public read endpoint**.

## 3) Functional Requirements

### 3.1 Event Model

**Required fields**

- `id` (UUID)
- `title` (string)
- `startAt` (ISO datetime)
- `endAt` (ISO datetime)
- `location` (string)
- `status` (enum: `DRAFT` | `PUBLISHED` | `CANCELLED`)

**Private fields (never exposed publicly)**

- `internalNotes` (string)
- `createdBy` (string/email)
- `updatedAt` (ISO datetime)

**Public fields (safe to expose)**

- `id`, `title`, `startAt`, `endAt`, `location`, `status` (only if `PUBLISHED` or `CANCELLED`), and derived `isUpcoming` (boolean: `startAt > now`).

### 3.2 API

Stack is your choice: Elixir, Go, Clojure, Node/TypeScript, etc.

**Authentication:** Admin endpoints require `Authorization: Bearer <token>` header. Use any static token (e.g., `admin-token-123`) stored in `.env` or hardcoded for this exercise.

#### POST `/events` (requires auth)

- **Body:** `title, startAt, endAt, location, status (default DRAFT), internalNotes (optional), createdBy (optional)`
- **Validations:**
  - `title` non-empty (max 200 chars)
  - `startAt < endAt`
  - `location` non-empty
  - Events cannot start in the past
- **Side effect:** trigger mocked async notification: `"New event created: <title>"`
- **Returns:** full event payload (including private fields) with 201 status
- **Errors:** 400 for validation errors, 401 for missing/invalid auth

#### PATCH `/events/:id` (requires auth)

- **Body:** Partial update (`status` and `internalNotes` only)
- **Validations:**
  - Cannot move from `PUBLISHED` / `CANCELLED` back to `DRAFT`
  - Status transitions:
    - `DRAFT` → `PUBLISHED` / `CANCELLED`
    - `PUBLISHED` → `CANCELLED`
- **Side effects:**
  - On `DRAFT` → `PUBLISHED`: log `"Event published: <title>"`
  - On any → `CANCELLED`: log `"Event cancelled: <title>"`
- **Returns:** updated event (200) or 404 if not found

#### GET `/events` (requires auth)

- **Query params:**
  - `dateFrom` & `dateTo` (YYYY-MM-DD, inclusive range)
  - `locations` (comma-separated list, case-insensitive contains)
  - `status` (comma-separated list)

    - `page` (default 1) & `limit` (default 20, max 100)
- **Returns:** paginated events (includes private fields)

Example response:

```
{
  "events": [...],
  "pagination": {
    "page": 1,
    "limit": 20,
    "total": 150,
    "totalPages": 8
  }
}
```

#### GET `/public/events` (no auth)

- **Query params:** same as above except no `status` filter
- **Returns:** only events with `status ∈ {PUBLISHED, CANCELLED}`
- **Field projection:** `id, title, startAt, endAt, location, isUpcoming, status`

### 3.3 Error Response Format

All error responses follow:

```
{
  "error": {
    "code": "VALIDATION_ERROR",
    "message": "Validation failed",
    "details": [
      {"field": "startAt", "message": "Must be in the future"}
    ]
  }
}
```

### 3.4 Notification Service (mock)

Implement a notifier interface/behaviour that logs to console but could later be wired to AWS SES/SNS. Required notifications:

- Event created
- Event published

- Event cancelled

## 4) AI-Powered Event Summary with Caching

**Feature:** Each event card (public) can have an AI-generated summary streamed in real-time.

**Implementation Requirements:**

- Endpoint: `GET /public/events/:id/summary` (no auth)
- Use Server-Sent Events (`Content-Type: text/event-stream`) or other streaming mechanism
- Cache summaries in memory using a key based on a hash of public fields
- Invalidate cache if the event's title, location, or dates change
- Headers: Include `X-Summary-Cache: HIT|MISS` for observability

**Mock Summary Generation:**

- Deterministic based on event data (no real AI API needed)
- 50–100 tokens, streamed in chunks of 2–5 tokens with small delays
- Style: short, engaging, mentions key event details and a call to action

**Bonus:**

- Show cache hit/miss indicator in dev logs
- Add proper `Cache-Control` headers

## 5) Testing

**Backend**

- **End-to-end tests:**
  - Create event → Query via `/events` → Verify appears in `/public/events` if PUBLISHED
  - Status transition flow: DRAFT → PUBLISHED → appears in public endpoint
  - Authentication enforcement (401 when missing/invalid token)
- **Security test:** assert private fields never leak in `/public/events`
- **Validation tests:** at least one per type of validation error

Prefer black-box tests targeting observable behaviour.

## 6) Deliverables

- Git repository (or ZIP) with backend code
- **Technical README** including:
  - How to run backend locally
  - Example cURL/Postman calls for all endpoints