COMP3011 Technical Report: EventHub API

Module: COMP3011 – Web Services and Web Data
Student: Nathaniel Sebastian (sc232ns)
Date: 5th February 2026
GitHub: github.com/NathS04/comp3011-cw1-api
Live API: comp3011-cw1-api.onrender.com

---

1. Reproducibility

Quickstart (Fresh Clone): 1. `python3 -m venv .venv && source .venv/bin/activate` 2. `pip install -r requirements.txt` 3. `alembic upgrade head` 4. `python -c "import app.main"` (Verifies imports) 5. `pytest -q` (Runs 31 tests) 6. `uvicorn app.main:app --reload`

Expected Results: all 31 tests passed. API available at `http://127.0.0.1:8000/docs`. **Note:** `scripts/import_dataset.py` requires an internet connection to fetch the XML feed.

---

2. Dataset Provenance

Attribute	Details
Source Name	Leeds Temporary Event Notices (TENs)
Provider	Leeds City Council (Data Mill North)
Format	XML (Live Feed)
URL	`https://opendata.leeds.gov.uk/downloads/Licences/temp-event-notice/temp-event-notice.xml`
Last Updated	Daily (as per portal metadata)
Retrieval Date	5th February 2026
Fields Used	`Reference_Number` (ID), `Premises_Name` (Title), `Activities` (Desc), `Event_Start_Date`, `Event_End_Date`
Limitations	No geolocation coordinates; Categories are unstructured text; Times often “00:00”

Why this dataset? It represents a “messy” real-world source requiring non-trivial XML parsing, date normalization (DD/MM/YYYY -> ISO8601), and robust error handling (skipping malformed records).

---

3. Architecture

The system follows a layered architecture to ensure separation of concerns and testability:

1. **Router Layer (`app/api`):** Handles HTTP requests/responses, auth checks, and schema validation.
2. **Service/CRUD Layer (`app/crud.py`):** Contains pure business logic. Decoupled from HTTP.
3. **Data Layer (`app/models.py`):** SQLAlchemy ORM models mapping to database tables.
4. **Database:** SQLite (Development) / PostgreSQL (Production).

Diagram: `Client (React/Curl) <-> FastAPI (Pydantic) <-> Logic (SQLAlchemy) <-> PostgreSQL`

---

4. Engineering Trade-offs & Design Decisions

SQLite vs PostgreSQL

- **Decision:** Use SQLite locally, PostgreSQL on Render.
- **Trade-off:** SQLite is zero-config but lacks advanced concurrent writing. Postgres complicates dev setup but ensures production reliability.
- **Mitigation:** `alembic` handles dialect differences (e.g., SQLite's lack of `ALTER TABLE`).

XML Parsing Strategy (DOM vs Streaming)

- **Decision:** Used `xml.etree.ElementTree` (DOM-style).
- **Trade-off:** Loads entire file into memory. Fast for 500 records (<1MB), but risky for gigabyte-scale files.
- **Why acceptable:** Leeds TEN dataset is small (~50KB). If it grows >100MB, would switch to `iterparse`.

Auth Implementation

- **Decision:** Stateless JWT (HS256) with 30-minute expiry.
- **Trade-off:** Simple scaling (no Redis session store needed) vs inability to revoke tokens immediately.
- **Mitigation:** Documented short expiry as security control.

Recommendation Algorithm

- **Decision:** Simple location-matching heuristic.
- **Trade-off:** Computational speed (O(1) lookup) vs "smartness" (no collaborative filtering).
- **Why acceptable:** Sufficient for cold-start demo; runs in <10ms.

---

5. Dataset Ingestion Pipeline

Script: `scripts/import_dataset.py`

1. **Fetch:** Downloads XML from Leeds Open Data.
2. **Verify:** Computes SHA256 hash of raw content for provenance.
3. **Parse:** Extract `Temporary_Event_Notice` nodes.
4. **Upsert:** Uses `source_record_id` (Reference No) to identify duplicates. Updates existing records instead of creating duplicates.
5. **Log:** Records usage stats to `import_runs` table (duration, rows inserted/updated).

Robustness: Continue-on-error behavior (skips single bad rows, logs error, proceeds).

---

6. Evaluation (Metrics)

Metric	Result	Environment
Import Time	~2.1 seconds	Local M1 Mac (WiFi)
Import Throughput	~240 records/sec	Local M1 Mac
API Latency (Read)	8ms (avg)	Local SQLite
API Latency (Write)	12ms (avg)	Local SQLite
Test Coverage	31 Tests	100% Pass Rate

Test Distribution: * **Auth:** 5 tests (Register, Login, Bad Token) * **CRUD:** 15 tests (Events, RSVPs, Attendees) * **Analytics:** 6 tests (Seasonality, Trending, Recs) * **Import/Admin:** 5 tests (XML parsing, Idempotency)

---

7. GenAI Usage (Critical Reflection)

Tools & Role

- **Gemini 3 Pro:** “Junior Developer” role (Drafting boilerplate, writing tests).
- **Claude Opus:** “Senior Reviewer” role (Refactoring, documentation polish).

Successes

- Gemini successfully suggested the `StaticPool` pattern for in-memory SQLite testing, solving a “thread check” error.
- Claude generated the 9-section report structure which improved document clarity.

Failures & Corrections (Manual Intervention)

1. **Bcrypt Compatibility:** AI suggested an `argon2` hasing library that failed to compile on Render. I manually reverted to `passlib[bcrypt]` for stability.
2. **Duplicate Imports:** Reference code generation created circular imports between `models.py` and `schemas.py`. I manually broke the cycle using string forward references.
3. **Missing Dependency:** AI wrote a script using `requests` but forgot to add it to `requirements.txt`. I caught this via `ModuleNotFoundError` and fixed the build process.

Full logs available in `docs/GENAI_EXPORT_LOGS.pdf`

---

8. Limitations & Future Work

1. **Single Tenant Admin:** No role separation (all users are basic). *Plan: Add `is_admin` column.*
2. **Rate Limiting:** API vulnerable to DoS. *Plan: Add `slowapi` or Nginx limiting.*
3. **Data Freshness:** Imports are manual/triggered. *Plan: Cron job or Celery task.*
4. **Token Refresh:** Users forced to relogin every 30m. *Plan: Implement refresh tokens.*

Word Count: ~880 words
Report for COMP3011 CW1, University of Leeds