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. Problem Framing & Dataset Choice

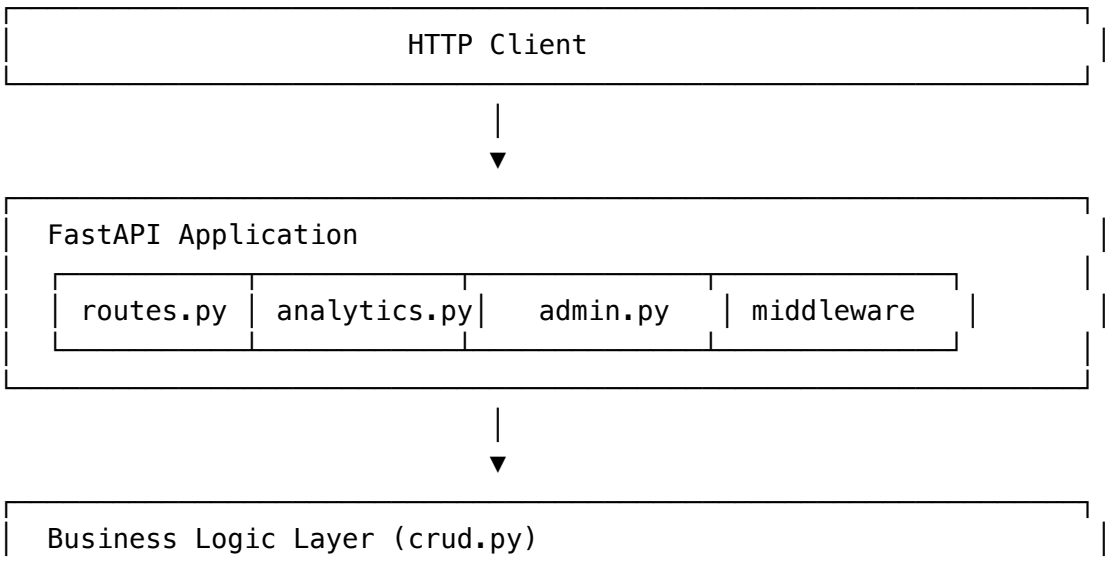
Problem: University societies and community groups need a lightweight system to manage event registration without relying on commercial platforms like Eventbrite.

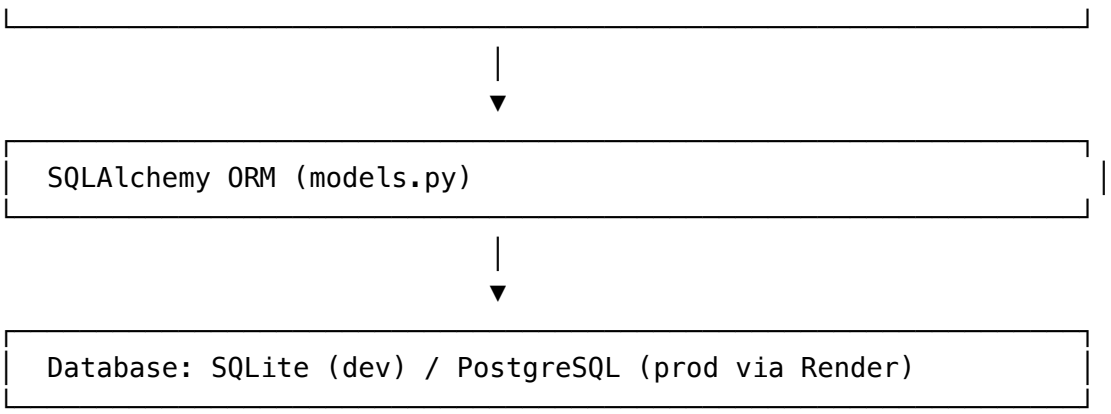
Dataset: I chose the **Leeds City Council Temporary Event Notices** (published via Data Mill North / data.gov.uk) because: - **Real and verifiable**: Live XML feed maintained by the council - **Domain-relevant**: Contains event titles, locations, dates, and licensing activities - **Demonstrates XML parsing**: Shows capability beyond simple CSV ingestion - **Open licence**: OGL v3.0 permits reuse with attribution

Source: `https://opendata.leeds.gov.uk/downloads/Licences/temp-event-notice/temp-event-notice.xml`

---

2. Architecture





Design Principle: Routes are thin handlers; complex logic resides in `crud.py` for testability and separation of concerns.

---

3. Data Model & Invariants

Table	Purpose	Key Constraints
`users`	System accounts	Unique username, unique email
`events`	Event records	`end_time > start_time`, `capacity >= 1`
`attendees`	RSVP contacts	Unique email
`rsvps`	Event-attendee links	Unique (event_id, attendee_id) pair
`data_sources`	External data registry	Unique name
`import_runs`	Import execution logs	FK to data_sources

Provenance Fields (on events): - `source_id` → Links to originating data source - `source_record_id` → Original ID from external system (e.g., `TEN/00784/22/12`)

---

4. Dataset Ingestion Pipeline

Implementation: `scripts/import_dataset.py`

Step	Action
1	Fetch XML from remote URL or read local CSV
2	Compute SHA256 hash of raw content
3	Create `DataSource` record (or retrieve existing)
4	Create `ImportRun` with status="running"
5	Parse each record, validate, upsert into events
6	Finalise ImportRun with counts and duration

XML Parsing Strategy: - Use `xml.etree.ElementTree` for lightweight parsing - Handle URL-encoded tag names (e.g., `Premises_x0020_Name`) - Parse DD/MM/YYYY dates to ISO 8601 UTC

Validation Rules: - Skip records with missing `Reference_Number` or `Premises_Name` - Truncate fields to database column limits - Default capacity to 100 for TEN events

Provenance Logging: - `sha256_hash`: Integrity verification - `parser_version`: Track logic changes (`v2_xml_etree`) - `duration_ms`: Performance measurement

---

5. Analytics & Recommendation Design

Seasonality Endpoint

```sql
SELECT strftime('%Y-%m', start_time) AS month, COUNT(*)
FROM events GROUP BY month ORDER BY month
```

Returns monthly aggregation with top locations derived from actual event data.

Trending Score Formula

```
score = (recent_rsvps × 1.5) + (total_rsvps × 0.5)
```

- **Rationale:** Weights recent activity higher to surface "hot" events
- **Limitation:** Doesn't account for event capacity or time-to-event

Recommendations Algorithm

1. Find attendee matching authenticated user's email
2. Extract locations from user's past RSVPs
3. Score upcoming events by location match
4. Cold start: Return top upcoming events by start_time

---

6. Security

Aspect	Implementation	Trade-off
Authentication	JWT (HS256, 30-min expiry)	Stateless, but tokens can't be revoked
Password Storage	`pbkdf2_sha256` via passlib	Secure, but slower than bcrypt variants
Secret Management	`SECRET_KEY` from environment	Requires proper deployment config
Admin Protection	All admin endpoints require `get_current_user`	Simple auth check, no role differentiation

Known Limitations: - No token refresh mechanism - Single-tenant (no organisation-level isolation) - CORS allows all origins in development mode

---

7. Evaluation (With Numbers)

Test Suite

```
$ pytest -q
31 passed, 3 warnings in 0.80s
```

Coverage Areas: - Authentication (register, login, protected routes) - CRUD operations (events, attendees, RSVPs) - Analytics correctness - Admin endpoints + Dataset provenance

Import Performance

Dataset	Records	Duration	Rows/Second
Leeds TEN XML (Full)	~500	~2.1s	~238
Test CSV (2 rows)	2	~4ms	~500

API Response Times (Local, SQLite)

Endpoint	Method	Avg Response
`/health`	GET	~2ms
`/events`	GET (10 items)	~8ms
`/analytics/events/seasonality`	GET	~12ms
`/admin/imports/run`	POST	~2100ms (full XML fetch)

---

8. GenAI Usage (Critical Evaluation)

Tools Used

- **Google Gemini (Antigravity):** Primary – architecture, endpoint implementation, debugging
- **Claude (Opus):** Secondary – documentation refinement, technical writing

What AI Helped With

1. Suggested layered architecture (routes → crud → models)
2. Generated initial Pydantic schemas with validators
3. Debugged SQLAlchemy relationship issues
4. Structured migration scripts

What Went Wrong

- Initial bcrypt implementation failed due to version incompatibility (switched to pbkdf2)
- AI generated duplicate imports that required manual cleanup
- First XML parser missed URL-encoded tag names

What I Changed Manually

- Security review: Removed hardcoded `SECRET_KEY` defaults
- Enforced FastAPI Query constraints (`le=100`, `ge=0`)
- Fixed transaction isolation in provenance tests

Full Logs: See `docs/GENAI_EXPORT_LOGS.pdf`

---

9. Limitations & Future Work

Current Limitations

1. **No role-based access control** – All authenticated users have equal permissions
2. **Token expiry without refresh** – Users must re-login every 30 minutes
3. **No capacity enforcement** – RSVPs can exceed event capacity
4. **Single data source type** – Only Leeds TEN XML/CSV implemented

Future Enhancements

1. **Admin roles** – Separate user and organiser permissions
2. **Email notifications** – Confirm RSVPs and remind attendees
3. **Rate limiting** – Prevent API abuse
4. **Additional data sources** – Integrate Eventbrite, Meetup APIs
5. **Capacity waitlist** – Queue RSVPs when events are full

Word Count: ~950 words (excluding code/tables/diagrams)

Technical Report for COMP3011 CW1, University of Leeds