

GenAI Conversation Export Logs

Module: COMP3011 – Web Services and Web Data

Student: Nathaniel Sebastian (sc232ns)

Date: 5th February 2026

Tools Used: Google Gemini (Antigravity), Claude (Anthropic)

Summary of AI Usage

Tool	Purpose	Sessions
Google Gemini	Coding, debugging, test generation, architecture exploration	8
Claude	Documentation polish, refactoring review	2

Test Count: 31 tests passing (verified on clean environment)

Session 1: Architecture Planning

Date: 3 February 2026

Tool: Google Gemini

Prompt:

I need to build a REST API for event management as coursework. What's the best architecture approach for FastAPI + SQLAlchemy?

AI Response (Summary):

Suggested a layered architecture: - Routes layer for HTTP handling - CRUD layer for business logic - Models for database entities - Schemas for Pydantic validation

Recommended separating concerns so routes are thin and logic is testable in CRUD.

My Decision:

Adopted this approach. Created `app/api/routes.py`, `app/crud.py`, `app/models.py`, `app/schemas.py`.

Session 2: RSVP Data Model Design (Alternative Exploration)

Date: 3 February 2026

Tool: Google Gemini

Prompt:

Should I store RSVPs as a list embedded in the Event model, or as a separate table?

AI Response (Summary):

Explained trade-offs: - **Embedded:** Simpler queries, no JOINs, but no uniqueness constraint - **Separate table:** Enables UNIQUE(event_id, attendee_id), timestamps, independent queries

Recommended separate table for relational integrity.

My Decision:

Chose separate `rsvps` table with UNIQUE(event_id, attendee_id) constraint. This enabled proper cascade deletes and duplicate prevention—critical for data integrity.

Why this matters (creative reasoning): The embedded approach would have been faster to implement but would have required application-level uniqueness checks, introducing race conditions.

Session 3: Authentication Approach (Alternative Exploration)

Date: 3 February 2026

Tool: Google Gemini

Prompt:

JWT vs server-side sessions for a REST API authentication?

AI Response (Summary):

- **JWT:** Stateless, simpler for pure APIs, no session storage needed
- **Sessions:** Enable immediate revocation, require Redis/DB storage

For coursework scope, JWT simplicity outweighs revocation limitations.

My Decision:

Implemented JWT with 30-minute expiry. Documented revocation as a known limitation.

Trade-off rationale: For a coursework project without Redis infrastructure, JWT's stateless nature aligns with REST principles and simplifies deployment on Render.

Session 4: SQLite Migration Issue

Date: 4 February 2026

Tool: Google Gemini

Prompt:

Alembic migration fails with “Cannot add a NOT NULL column with default value NULL” on SQLite

AI Response:

SQLite doesn't support ALTER TABLE for certain operations. Use Alembic's batch_alter_table context manager which recreates the table.

Code Fix:

```
with op.batch_alter_table('events') as batch_op:  
    batch_op.add_column(sa.Column('source_id', sa.Integer(),  
        nullable=True))
```

Outcome:

Migration succeeded. Applied this pattern to all subsequent migrations.

Session 5: Test Fixture Design

Date: 4 February 2026

Tool: Google Gemini

Prompt:

Tests are interfering with each other. How do I isolate database state?

AI Response:

Use in-memory SQLite with StaticPool and create/drop tables per test function:

```
@pytest.fixture(scope="function")
def db(engine):
    Base.metadata.create_all(bind=engine)
    yield session
    Base.metadata.drop_all(bind=engine)
```

My Enhancement:

Adopted the pattern. All 31 tests now run in <1 second with complete isolation.

Session 6: Analytics Implementation

Date: 4 February 2026

Tool: Google Gemini

Prompt:

| How should I implement a trending score for events?

AI Response:

Suggested formula weighting recent activity higher:

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

Explained that 1.5x multiplier rewards events gaining momentum.

My Implementation:

Adopted the formula. Added `window_days` parameter to control “recent” definition.
Tested in `tests/test_analytics.py`.

Session 7: Timezone Bug Fix

Date: 4 February 2026

Tool: Google Gemini

Prompt:

| Error in analytics.py: NameError: name ‘timezone’ is not defined

AI Response:

Missing import. Add `timezone` to the `datetime` import:

```
from datetime import datetime, timedelta, timezone
```

Outcome:

Fixed. Tests continue to pass.

Session 8: PDF Generation

Date: 4 February 2026

Tool: Google Gemini

Problem:

Coursework requires PDF documentation, but pandoc/weasyprint have missing dependencies.

Solution Found:

Use Puppeteer (Node.js) to convert HTML to PDF:

```
const page = await browser.newPage();
await page.goto(htmlPath);
await page.pdf({path: outputPath, format: 'A4'});
```

Outcome:

Generated docs/API_DOCUMENTATION.pdf and TECHNICAL_REPORT.pdf.

Session 9: Code Quality Audit

Date: 5 February 2026

Tool: Google Gemini

Problem:

Project had hidden issues: - Missing requests in requirements.txt - Unfinished test ending in pass - Duplicate field definitions in models.py - Deprecated FastAPI Query usage

AI Action:

Identified all issues and proposed fixes.

Correction (Human Led):

- Verified failures on clean install before accepting fixes
- Rewrote placeholder test with real assertion logic (location-based recommendation filtering)

- Manually confirmed 31 tests passing
-

Session 10: Documentation Polish

Date: 5 February 2026

Tool: Claude

Purpose:

Review and improve technical report structure for examiner readability.

AI Contribution:

Suggested 11-section structure with compliance checklist, Mermaid diagrams for architecture/ERD, and explicit references.

My Modifications:

- Added measured metrics (import time, latency)
 - Ensured all claims match actual code behavior
 - Cross-checked consistency across all docs
-

Critical Reflection

What AI Did Well

- **Boilerplate generation:** Rapidly scaffolded FastAPI routes and Pydantic schemas
- **Trade-off explanation:** Clearly articulated RSVP storage and auth alternatives
- **Debugging:** Quickly identified missing imports and SQLite migration issues

Where AI Failed (and I Fixed It)

Failure	Impact	My Fix
Omitted requests from requirements.txt	ModuleNotFoundError on clean install	Added dependency manually
Generated placeholder test (pass)	False test coverage	Rewrote with real assertions
Suggested deprecated Query(regex=...)	FastAPI deprecation warning	Changed to Query(pattern=...)
Circular imports (models ↔ schemas)	Import errors	Used string forward references

Conclusion

AI accelerated development of standard patterns by ~3x but introduced integration bugs requiring manual verification. Every AI suggestion was reviewed before commit. The “clean install test” caught the most critical failure (missing dependency).

Appendix: Example Prompts

Prompt 1 (Architecture): > “I need to build a REST API for event management as coursework. What’s the best architecture approach for FastAPI + SQLAlchemy?”

Prompt 2 (Alternative Exploration): > “Should I store RSVPs as a list embedded in the Event model, or as a separate table?”

Prompt 3 (Security Trade-off): > “JWT vs server-side sessions for a REST API authentication?”

Exported for COMP3011 CW1 submission – 5th February 2026