Cardano Analytics Platform

Test Execution Report

December 15th, 2025

Project Catalyst Fund13
Project ID: 1300034

Contributors

Rafael Brandão (rafael@mobr.ai)
Marcio Moreno (mmoreno@mobr.ai)

Cardano Analytics Platform (CAP)

Test Execution Report

Project: CAP (Cardano Analytics Platform)
Repo: `mobr-ai/cap`
Branch / Commit: `main`
Date (local): `Sat Dec 06 09:46:35 AM CET 2025`
Host: `mobr01`
Python: `python 3.12.3`
Poetry: `poetry 1.8.2`
Test directory: `src/tests/`

Objective

Demonstrate, through empirical test execution, that CAP's core subsystems are functional and robust:

- ETL extraction → transformation → loading into RDF/triplestore
- SPARQL query correctness and time-handling
- LLM-assisted NL→SPARQL generation and contextualized answers
- API stability and integration behavior (incl. streaming)
- End-to-end analytical use cases through the user-facing experience

This report provides the milestone evidence that CAP is operational and validated via repeatable tests.

Preconditions / Environment

- CAP and dependencies running via Docker Compose (Postgres, QLever/triplestore, etc.)
- Virtual environment active
- Dev dependencies installed

Relevant checks/commands

```
# from repo root
git status -sb
```

```
docker compose ps
docker compose logs --tail=50
python -V
poetry --version
```

Test Scope

Pytest modules (unit/integration/system)

- `src/tests/test_api.py`
- `src/tests/test_etl_extractors.py`
- `src/tests/test_etl_transformers.py`
- `src/tests/test_etl_loader.py`
- `src/tests/test_etl_service.py`
- `src/tests/test_etl_pipeline.py`
- `src/tests/test_sparql_dates.py`
- (plus shared fixtures: `src/tests/conftest.py`)

Script-based tests

- `src/tests/sparql_tests.py`
- `src/tests/sparql_generation_tests.py`
- `src/tests/oc_tests.py`
- `src/tests/nl_query_tests.py`

Commands

Setup

```
# repo root
source venv/bin/activate
poetry install --with dev

# ensure deps are up
docker compose up -d
docker compose ps
```

Run all pytest + save artifacts

```
mkdir -p test-artifacts

pytest -v \
  --junitxml=test-artifacts/pytest-junit.xml \
  --cov=src/cap \
  --cov-report=term-missing \
  --cov-report=xml:test-artifacts/coverage.xml \
  --cov-report=html:test-artifacts/htmlcov \
  2>&1 | tee test-artifacts/pytest.log
```

Run per-file (useful for targeted reruns)

```
pytest -v src/tests/test_api.py            2>&1 | tee
test-artifacts/test_api.log
pytest -v src/tests/test_etl_pipeline.py   2>&1 | tee
test-artifacts/test_etl_pipeline.log
pytest -v src/tests/test_sparql_dates.py   2>&1 | tee
test-artifacts/test_sparql_dates.log
```

Run a single test function (example)

```
pytest -s -v src/tests/test_etl_service.py::test_etl_service_status \
  2>&1 | tee test-artifacts/test_single.log
```

Run individual script tests

```
# SPARQL execution tests (examples folder)
python src/tests/sparql_tests.py \
  2>&1 | tee test-artifacts/sparql_tests.log

# SPARQL execution tests (specific txt)
python src/tests/sparql_tests.py --txt-folder
documentation/examples/sparql/transactions.txt \
  2>&1 | tee test-artifacts/sparql_tests_transactions.log

# SPARQL generation tests (LLM → SPARQL)
python src/tests/sparql_generation_tests.py \
  2>&1 | tee test-artifacts/sparql_generation_tests.log
```

```
python src/tests/sparql_generation_tests.py --txt-folder
documentation/examples/nl/use_cases.txt \
  2>&1 | tee test-artifacts/sparql_generation_use_cases.log

# “oc” LLM prompt tests
python src/tests/oc_tests.py \
  2>&1 | tee test-artifacts/oc_tests.log

# NL query pipeline integration script
python src/tests/nl_query_tests.py \
  2>&1 | tee test-artifacts/nl_query_tests.log

python src/tests/nl_query_tests.py --txt-folder
documentation/examples/nl/use_cases.txt \
  2>&1 | tee test-artifacts/nl_query_tests_use_cases.log
```

Helper commands

One-shot helper

If you want one command to run the whole suite and collect logs:

```
mkdir -p test-artifacts && \
( pytest -v --junitxml=test-artifacts/pytest-junit.xml --cov=src/cap
--cov-report=term-missing --cov-report=xml:test-artifacts/coverage.xml
--cov-report=html:test-artifacts/htmlcov ) 2>&1 | tee
test-artifacts/pytest.log && \
python src/tests/sparql_tests.py 2>&1 | tee
test-artifacts/sparql_tests.log && \
python src/tests/sparql_generation_tests.py 2>&1 | tee
test-artifacts/sparql_generation_tests.log && \
python src/tests/oc_tests.py 2>&1 | tee test-artifacts/oc_tests.log &&
\
python src/tests/nl_query_tests.py 2>&1 | tee
test-artifacts/nl_query_tests.log
```

One-shot CAP Test Runner (Poetry)

```
# From CAP repo root
```

```
mkdir -p test-artifacts && \
poetry install --with dev && \
docker compose up -d && \
(
  echo "=== PYTEST SUITE ===" && \
  poetry run pytest -v \
    --junitxml=test-artifacts/pytest-junit.xml \
    --cov=src/cap \
    --cov-report=term-missing \
    --cov-report=xml:test-artifacts/coverage.xml \
    --cov-report=html:test-artifacts/htmlcov
) 2>&1 | tee test-artifacts/pytest.log && \
(
  echo "=== SPARQL EXECUTION TESTS ===" && \
  poetry run python src/tests/sparql_tests.py
) 2>&1 | tee test-artifacts/sparql_tests.log && \
(
  echo "=== SPARQL GENERATION TESTS ===" && \
  poetry run python src/tests/sparql_generation_tests.py
) 2>&1 | tee test-artifacts/sparql_generation_tests.log && \
(
  echo "=== OC (LLM PROMPT) TESTS ===" && \
  poetry run python src/tests/oc_tests.py
) 2>&1 | tee test-artifacts/oc_tests.log && \
(
  echo "=== NL QUERY INTEGRATION TESTS ===" && \
  poetry run python src/tests/nl_query_tests.py
) 2>&1 | tee test-artifacts/nl_query_tests.log
```

Strict CI-style variant (fail fast)

```
set -euo pipefail

mkdir -p test-artifacts
poetry install --with dev
docker compose up -d

poetry run pytest -v \
  --junitxml=test-artifacts/pytest-junit.xml \
  --cov=src/cap \
  --cov-report=term-missing \
  --cov-report=xml:test-artifacts/coverage.xml \
  --cov-report=html:test-artifacts/htmlcov \
  | tee test-artifacts/pytest.log

poetry run python src/tests/sparql_tests.py | tee
test-artifacts/sparql_tests.log
poetry run python src/tests/sparql_generation_tests.py | tee
test-artifacts/sparql_generation_tests.log
poetry run python src/tests/oc_tests.py | tee
test-artifacts/oc_tests.log
poetry run python src/tests/nl_query_tests.py | tee
test-artifacts/nl_query_tests.log
```

What artifacts this produces

- **JUnit report**
  `test-artifacts/pytest-junit.xml`

- **Coverage**

  - XML: `test-artifacts/coverage.xml`
  - HTML: `test-artifacts/htmlcov/index.html`

- **Logs**

  - `test-artifacts/pytest.log`
  - `test-artifacts/sparql_tests.log`
  - `test-artifacts/sparql_generation_tests.log`
  - `test-artifacts/oc_tests.log`
  - `test-artifacts/nl_query_tests.log`

Result summary

Test Suite Scope (Executed)

Pytest suite (unit/integration/system)

Validated across modules in `src/tests/`, including:

- ETL: `test_etl_extractors.py`, `test_etl_transformers.py`, `test_etl_loader.py`, `test_etl_service.py`, `test_etl_pipeline.py`
- API: `test_api.py`
- SPARQL utilities & dates: `test_sparql_dates.py`
- NL pipeline supporting tests: `nl_cache_tests.py`, `nl_normalization_tests.py`
- Shared fixtures: `conftest.py`

Scripted SPARQL & NL pipeline validations

- SPARQL execution/regression checks: `sparql_tests.py`
- SPARQL generation (LLM-assisted): `sparql_generation_tests.py`
- LLM response behavior checks: `oc_tests.py`
- Full NL query integration harness: `nl_query_tests.py`

How Tests Were Run

With CAP dependencies running (`docker compose up -d`), from project root:

```
source venv/bin/activate
poetry install --with dev
```

Pytest:

```
pytest -v
pytest -v src/tests/test_api.py
pytest -s src/tests/test_etl.py::test_etl_service_status
pytest --cov=src/cap
```

SPARQL tests:

```
python src/tests/sparql_tests.py

python src/tests/sparql_tests.py --txt-folder
documentation/examples/sparql/transactions.txt
```

SPARQL generation:

```
python src/tests/sparql_generation_tests.py

python src/tests/sparql_generation_tests.py --txt-folder
documentation/examples/nl/use_cases.txt
```

Natural Language tests:

```
python src/tests/oc_tests.py
python nl_query_tests.py

python nl_query_tests.py --txt-folder
documentation/examples/nl/use_cases.txt
```

Execution Environment (mobr01)

OS / Kernel

- Linux Mint 22.2 (Ubuntu 24.04 base), Kernel `6.8.0-90-generic`

CPU / Memory

- Intel Core Ultra 9 285K (24-core), up to ~5.4 GHz observed
- 64 GiB RAM

GPU

- NVIDIA GeForce RTX 5080 (driver `580.95.05`)
- Intel integrated graphics present (i915)

Storage

- Total ~9.14 TiB across NVMe + HDD
- Multiple NVMe drives including WD Black SN850X 4TB

Network

- Realtek RTL8125 2.5GbE (active), plus MEDIATEK Wi-Fi device present

---

Results

Automated Tests (Pytest)

- **Status:** ✅ PASS
- **Result:** `20 passed`
- **Measured runtime (from pytest summary): 43.72 seconds**

SPARQL Generation + Simple Validations

- **Status:** ✅ PASS
- **Result:** completion message indicates **all checks passed** (no failures)

LLM / NL Pipeline (Ollama integration)

- **Status:** ✅ PASS
- **Result:** successful generation calls and pipeline completion; no error states observed

End-to-End NL Use Cases (batch)

- **Status:** ✅ PASS
- **Result:** all test queries completed successfully; no failures
- **Measured performance (from run summary):**
  - **Total: 399.24 s**
  - **Average: 39.92 s**
  - **Min: 17.71 s**
  - **Max: 151.22 s**

User-Facing Use Case Validation

- **Status:** ✅ PASS (functional verification)
- **Result:** governance voting analytics query returns a valid numeric answer and displays expected supporting details (including transaction reference) in the user experience.

Defects / Blockers

- **Critical defects:** None observed
- **Blockers:** None
- **Outcome:** Test execution provides empirical evidence of platform correctness and end-to-end operational readiness for the validated scenarios.

Acceptance Criteria Assessment

Acceptance Criterion	Status
ETL pipeline integrity (extract/transform/load + orchestration)	✅ Met
SPARQL correctness + date handling	✅ Met
NL→SPARQL generation + execution	✅ Met
LLM response contextualization pipeline	✅ Met
End-to-end analytics use cases complete successfully	✅ Met
Performance metrics captured for E2E batch	✅ Met