

Suite di Test - Sistema Catasto Storico

Panoramica

La suite di test fornisce una copertura completa del sistema Catasto Storico, includendo test unitari, di integrazione, GUI e end-to-end. La struttura è progettata per essere modulare, manutenibile ed estensibile.

Struttura dei Test

```
tests/
├── conftest.py           # Configurazione e fixtures condivise
├── test_database_manager.py # Test per operazioni database
├── test_gui_widgets.py   # Test per componenti GUI
├── test_integration.py   # Test di integrazione
├── run_tests.py          # Script runner principale
├── unit/                 # Test unitari aggiuntivi
├── integration/          # Test integrazione complessi
├── fixtures/             # Dati di test
└── reports/              # Report di test e coverage
```

Setup Iniziale

1. Installazione Dipendenze

```
bash

# Crea ambiente virtuale
python -m venv venv-test
source venv-test/bin/activate # Linux/Mac
# oppure
venv-test\Scripts\activate # Windows

# Installa dipendenze
pip install -r requirements-test.txt
```

2. Configurazione Database di Test

Il database di test viene creato automaticamente dalla fixture `test_db_setup`. Assicurarsi che PostgreSQL sia in esecuzione e configurato correttamente.

```
bash

# Verifica connessione PostgreSQL
psql -U postgres -c "SELECT version();"
```

3. Variabili d'Ambiente (Opzionale)

```
bash

export TEST_DB_HOST=localhost
export TEST_DB_PORT=5432
export TEST_DB_USER=postgres
export TEST_DB_PASSWORD=postgres
```

Esecuzione dei Test

Test Completi

```
bash

python tests/run_tests.py all
```

Test per Categoria

```
bash

# Solo test unitari
python tests/run_tests.py unit

# Solo test di integrazione
python tests/run_tests.py integration

# Solo test GUI
python tests/run_tests.py gui

# Test veloci (esclude test lenti)
python tests/run_tests.py fast
```

Test con Coverage

```
bash

python tests/run_tests.py coverage
# Il report HTML sarà in htmlcov/index.html
```

Test in Parallelo


```
bash

python tests/run_tests.py parallel --workers 4
```

Test Specifico

```
bash

python tests/run_tests.py --test tests/test_database_manager.py::TestComuneOperations::test_agg
```



Fixtures Principali

test_db_setup

- **Scope:** Session
- **Descrizione:** Crea e configura il database di test
- **Cleanup:** Automatico alla fine della sessione

db_manager

- **Scope:** Function
- **Descrizione:** Istanza di CatastoDBManager connessa al DB test
- **Dipendenze:** test_db_setup

clean_db

- **Scope:** Function
- **Descrizione:** Database pulito per ogni test
- **Uso:** Test che richiedono stato iniziale pulito

sample_data

- **Scope:** Function
- **Descrizione:** Database con dati di esempio pre-popolati
- **Contenuto:** Comune, possessori, partite, località

Convenzioni di Test

Naming

- File: test_<modulo>.py
- Classi: Test<Componente>
- Metodi: test_<funzionalità>_<scenario>

Struttura Test

python

```
def test_nome_descrittivo(self, fixtures):  
    """Docstring che descrive cosa testa."""  
    # Arrange - Setup dei dati  
  
    # Act - Esecuzione azione  
  
    # Assert - Verifica risultati
```

Marcatori

python

```
@pytest.mark.slow # Test che richiede > 1 secondo  
@pytest.mark.integration # Test di integrazione  
@pytest.mark.gui # Test che richiede GUI  
@pytest.mark.unit # Test unitario puro
```

Test Database

Operazioni CRUD

- Test creazione, lettura, aggiornamento, eliminazione
- Validazione constraint e integrità referenziale
- Gestione errori e eccezioni personalizzate

Transazioni

- Test commit e rollback
- Isolamento transazioni
- Operazioni concorrenti

Performance

- Test con grandi volumi di dati
- Ottimizzazione query
- Gestione pool connessioni

Test GUI

Widget Testing

- Inizializzazione componenti
- Interazioni utente (click, input)

- Aggiornamento stato e visualizzazione

Signal/Slot

- Emissione e ricezione segnali
- Connessioni tra componenti
- Event propagation

Validazione Form

- Input validation
- Error feedback
- Form submission

Test di Integrazione

Workflow Completi

- Creazione entità → Modifica → Eliminazione
- Import/Export dati
- Ricerche complesse

Scenari End-to-End

- Trasferimento proprietà
- Gestione variazioni
- Report generation

Best Practices

1. Isolamento

- Ogni test deve essere indipendente
- Usare fixtures per setup/teardown
- Non dipendere dall'ordine di esecuzione

2. Velocità

- Minimizzare accessi al database quando possibile
- Usare mock per dipendenze esterne
- Marcare test lenti appropriatamente

3. Affidabilità

- Evitare test flaky

- Gestire timing in test asincroni
- Verificare cleanup appropriato

4. Manutenibilità

- Test semplici e focalizzati
- Nomi descrittivi
- Riutilizzare fixtures comuni

Debugging Test

Esecuzione Verbosa

```
bash
```

```
pytest -vv tests/test_database_manager.py
```

Con Debugger

```
bash
```

```
pytest --pdb tests/test_specific.py
```

Print Statements

```
bash
```

```
pytest -s tests/ # Non cattura output
```

Continuous Integration

GitHub Actions Example

yaml

name: Tests

on: [push, pull_request]

jobs:

test:

runs-on: ubuntu-latest

services:

postgres:

image: postgres:15

env:

POSTGRES_PASSWORD: postgres

options: >-

--health-cmd pg_isready

--health-interval 10s

--health-timeout 5s

--health-retries 5

steps:

- uses: actions/checkout@v3

- uses: actions/setup-python@v4

with:

python-version: '3.9'

- name: Install dependencies

run: |

pip install -r requirements-test.txt

- name: Run tests

run: |

pytest --cov=. --cov-report=xml

- name: Upload coverage

uses: codecov/codecov-action@v3

Troubleshooting

Database Connection Issues

FATAL: password authentication failed

Soluzione: Verificare credenziali in `TEST_DB_CONFIG`

GUI Test Failures

`QXcbConnection: Could not connect to display`

Soluzione: Usare display virtuale (xvfb) in CI/headless

Import Errors

`ModuleNotFoundError: No module named 'catasto_db_manager'`

Soluzione: Verificare PYTHONPATH includa directory progetto

Estensione della Suite

Aggiungere Nuovi Test

1. Creare file in `tests/` seguendo naming convention
2. Importare fixtures necessarie da `conftest.py`
3. Implementare test seguendo pattern esistenti
4. Aggiungere marcatori appropriati

Aggiungere Nuove Fixtures

1. Definire in `conftest.py` o file test specifico
2. Documentare scope e cleanup
3. Considerare riutilizzabilità

Metriche di Qualità

Coverage Target

- Minimo: 70% coverage totale
- Ideale: 85%+ per moduli critici
- Focus su business logic, non boilerplate

Performance Benchmark

- Test unitari: < 100ms
- Test integrazione: < 1s
- Test E2E: < 5s
- Suite completa: < 5 minuti

Risorse Utili

- [Pytest Documentation](#)
- [PyQt Testing Guide](#)
- [PostgreSQL Testing Best Practices](#)
- [Python Testing 101](#)