SAAQAnalyzer Testing Documentation Index

Overview

Comprehensive testing documentation for SAAQAnalyzer codebase. This index provides quick access to testing resources organized by audience and use case.

For QA Engineers & Test Planners

Start Here: 1. TESTING_PRIORITIES.md - Risk assessment and testing priorities - 3-tier component classification (Critical/High/Medium/Low) - Critical test scenarios and success criteria - Phase-based execution strategy

1. TESTING_SURVEY.md - Comprehensive component analysis
   - 10 DataLayer components (databases, imports, caching, queries)
   - 4 Model components (data structures, progress tracking)
   - 5 UI components (panels, views, displays)
   - 2 Utility components (logging, versioning)
   - Existing test coverage assessment

For Developers Implementing Tests

Quick Reference: - TESTING_PRIORITIES.md - Coverage gaps and test scenarios - What to test by component - Why each component matters - Known pitfalls from architectural rules

Detailed Analysis: - TESTING_SURVEY.md - Full component documentation - Public APIs to test - Complex logic needing validation - Integration points - Test data considerations

---

For Architects & Tech Leads

Strategic Planning: - TESTING_PRIORITIES.md - Phase-based strategy - Phase 1 (Week 1-2): Foundation components - Phase 2 (Week 2-3): Functional components
- Phase 3 (Week 3-4): UI & integration - Estimated 800+ tests across 3 phases

- TESTING_SURVEY.md - Architecture context
  - Integer enumeration optimization strategy
  - Cache management architecture
  - Performance-critical paths
  - Known architectural constraints

Testing Highlights by Risk Level

Tier 1: Critical (Must Test First)

Component	Size	Risk	Tests Needed
OptimizedQueryManager	1.3K	CRITICAL	150 tests (filter conversion, RWI, regularization)
CategoricalEnumManager	787	CRITICAL	80 tests (schema creation, index validation)
FilterCacheManager	892	CRITICAL	100 tests (initialization, guards, data types)
RegularizationManager	1.9K	CRITICAL	120 tests (query translation, coupling)

Total Tier 1 Tests: ~450 (estimated)

Tier 2: High Priority (Should Test)

Component	Size	Risk	Tests Needed
DatabaseManager	7.8K	HIGH	100+ tests (cache invalidation, queries)
CSVImporter	958	HIGH	50+ tests (encoding edge cases)
SchemaManager	441	MEDIUM	60 tests (migration pipeline)
DataModels	2.1K	MEDIUM	40 tests (statistics, validation)

Total Tier 2 Tests: ~250 (estimated)

Tier 3: Supporting (Nice to Have)

Component	Size	Risk	Status
GeographicDataImporter	378	LOW	30 tests (parsing, hierarchy)
ImportProgressManager	258	MEDIUM	40 tests (stage tracking)
FilterPanel	2.7K	MEDIUM	30+ tests (UI state)
ChartView	879	MEDIUM	30+ tests (formatting)

Total Tier 3 Tests: ~100+ (estimated)

Critical Test Scenarios

1. Integer Enumeration Query System (Top Priority)

See: TESTING_SURVEY.md Section 1.4

Key Tests: - Filter string □ ID conversion (parenthesized codes) - RWI calculation with axle distribution - Percentage metric with baseline subquery - Query plan analysis for performance detection - Regularization with Make/Model coupling

Risk if Not Tested: Silent data corruption, 165s performance penalty

---

2. Cache Management System

See: TESTING_SURVEY.md Section 1.3

Key Tests: - Dual-initialization guard (concurrent access prevention) - Cache invalidation pattern (invalidate BEFORE initialize) - Data type selective loading (vehicle vs. license) - Regularization info accuracy - Uncurated pair detection

Risk if Not Tested: Stale data, concurrent initialization hangs

---

3. Character Encoding & Import

See: TESTING_SURVEY.md Section 1.2

Key Tests: - UTF-8 with ISO-Latin-1 fallback - Windows-1252 as final fallback - French diacritics preservation - Batch processing pipeline - Year detection and schema selection

Risk if Not Tested: Encoding errors, corrupted data

---

4. Normalization & Transformation

See: TESTING_SURVEY.md Section 1.1

Key Tests: - normalizeToFirstYear() division logic - applyCumulativeSum() transformation order - Edge cases (zero values, NaN, empty series) - Automatic precision detection - Legend and axis label formatting

Risk if Not Tested: Incorrect trend analysis, wrong precision

---

5. Regularization Make/Model Logic

See: TESTING_SURVEY.md Section 1.6

Key Tests: - Canonical hierarchy generation - Query translation with coupling - Year configuration impact - Cache invalidation on config change - Performance: hierarchy $<$1s (with cache)

Risk if Not Tested: Coupling inverted, massive performance regression

---

Test Infrastructure Needs

Database Fixtures

```
 SAAQAnalyzerTests/    Fixtures/     CSV_Files/      Vehicule_2017_Sample_1000.csv # Basic
vehicle data      Vehicule_2023_Encoding_Test.csv # Diacritics anomalies      Permis_2020_
# License data      Vehicule_2015_Fuel_Null.csv # Pre-2017 (no fuel)     Geographic/
    d001_sample.txt # Minimal hierarchy      d001_hierarchy_test.txt # Full validation
    Regularization/     sample_mappings.csv # Make/Model corrections    DatabaseFixture.swi
# Ephemeral test DB    SampleDataGenerator.swift # Synthetic data
```

Coverage Goals

Test Coverage Targets

- **Tier 1 Components**: 80%+ branch coverage (foundation critical)
- **Tier 2 Components**: 60%+ branch coverage (functional)
- **Tier 3 Components**: 40%+ coverage (infrastructure)
- **Overall**: 70% codebase minimum

Performance Validation

- Query execution: <10s (with all indexes)
- Cache initialization: <500ms
- Hierarchy generation: <1s (with cache)
- RWI calculation: <100ms

Reliability Standards

- Zero segmentation faults (concurrency safety)
- Zero silent data corruption
- 100% French diacritics preserved
- No stale cache data

Existing Test Coverage

Current Tests (5 files)

“‘ □ DatabaseManagerTests (80 lines) - Database connection, table checks, basic queries

□ CSVImporterTests (200 lines) - Vehicle/license imports, character encoding

□ FilterCacheTests (150 lines) - Cache separation, persistence, retrieval

□ WorkflowIntegrationTests (100 lines) - End-to-end import □ query workflows

□ No UI Tests - FilterPanel, ChartView, DataInspector “‘

Major Coverage Gaps

By Priority: 1. OptimizedQueryManager - 0% (CRITICAL) 2. CategoricalEnumManager - 0% (CRITICAL) 3. FilterCacheManager - 0% (CRITICAL, new) 4. RegularizationManager - 0% (CRITICAL) 5. DatabaseManager - ~10% (HIGH) 6. CSVImporter - ~15% (HIGH) 7. SchemaManager - 0% (MEDIUM) 8. All UI components - 0% (MEDIUM)

Architectural Rules & Test Validation

From CLAUDE.md - Critical constraints to validate:

Rule	Validation Strategy
Use integer enumeration IDs only	Query analysis: all WHERE clauses on `_id` columns
All enum table ID indexes required	Schema validation: verify 15+ indexes exist
Invalidate BEFORE initialize	Cache refresh cycle test
No `.onChange` for filters	Code review: manual button triggers only
>100ms ops in background	Task isolation tests with timing
Pass DB path, not connection	Concurrent task safety tests
Parent-scope expensive ViewModels	Lifecycle tests for sheet dismissal

---

Next Steps

Immediate (This Week)

1. □ Review TESTING_PRIORITIES.md (risk assessment)
2. Create test database fixture infrastructure
3. Create sample CSV test data files

Short Term (Week 1-2)

1. Implement Tier 1 component tests (450+ tests)

2. Set up CI/CD test pipeline
3. Achieve 80%+ coverage on critical components

Medium Term (Week 2-4)

1. Implement Tier 2 component tests (250+ tests)
2. Add UI component tests (30+ tests)
3. Create performance benchmark suite

Long Term (Ongoing)

1. Maintain 70% overall coverage
2. Performance regression testing
3. Edge case discovery and testing

Document Guide

TESTING_PRIORITIES.md

For: QA leads, project managers, developers **Contains**: Risk assessment, 3-tier prioritization, 8 critical test scenario categories, phase-based execution strategy, success criteria **Length**: ~450 lines **Key Sections**: - Component risk table - Critical test scenarios with checkboxes - Test data fixtures - Coverage gap assessment - Known pitfalls quick reference

TESTING_SURVEY.md

For: Developers, architects, test implementers **Contains**: Detailed component analysis (10 Data-Layer, 4 Models, 5 UI, 2 Utilities), existing test coverage, architectural constraints **Length**: 1,029 lines **Key Sections**: - Component responsibility & size - Public APIs to test - Complex logic needing validation - Integration points - Known pitfalls & architectural rules - Test coverage matrix

This Document (TESTING_INDEX.md)

For: Quick navigation, overview, first-time readers **Contains**: Index, highlights, highlights, quick reference links **Length**: ~300 lines

---

Resources

In Repository: - CLAUDE.md - Critical architectural rules (READ FIRST) - ARCHITECTURAL*GUIDE.md - System design overview - QUICK*REFERENCE.md - 5-minute quick start - REGULARIZATION*TEST*PLAN.
- Detailed regularization scenarios

External: - Swift Testing Framework docs - XCTest documentation - SQLite testing best practices

Questions?

Refer to the relevant detailed documentation: - **"What should I test?"** TESTING*PRIORITIES.md - **"How do I test component X?"** TESTING*SURVEY.md (search by name) - **"Why is rule Y important?"** CLAUDE.md - **"What's the architecture?"** ARCHITECTURAL_GUIDE.md