Unit Testing Guide - Summarizer Project

Overview

A comprehensive unit test suite has been created for the Summarizer 1.1 project. The tests cover all major services and utility functions, using mocking to isolate components and ensure reliable, fast test execution.

Test Statistics

- **Total Test Files**: 5
- **Total Test Cases**: 38
- **All Tests Passing**: ✅ Yes
- **Test Execution Time**: ~0.008s

Test Files Structure

1. test_helper.py

Tests for utility helper functions in the `utils/helpers.py` module.

Test Classes:

- `TestHelperFunctions` (12 tests)

Coverage:

- ✅ Percentage calculations with edge cases (zero division)
- ✅ Comparison logic (above/below/at average)
- ✅ Embedding parsing from multiple formats (list, string, Postgres)
- ✅ Error handling for invalid embeddings

Key Tests:

```
- test_percentage_basic()
- test_percentage_zero_whole()
- test_get_comparison_above_average()
- test_safe_parse_embedding_list()
- test_safe_parse_embedding_postgres_format()
```

2. test_employee.py

Tests for employee analytics utilities in the `utils/employee.py` module.

Test Classes:

- `TestEmployeeUtils` (12 tests)

Coverage:

- ✅ Single employee deed extraction (breakdowns, pauses, production metrics)
- ✅ Average comparison calculations across employees
- ✅ Employee statement generation with proper formatting
- ✅ Handling edge cases (non-existent employees, no logs)

Key Tests:

```
- test_get_single_employee_deeds_breakdown_count()
- test_get_single_employee_deeds_pause_duration()
- test_get_average_comparison_multiple_employees()
- test_generate_employee_statement_structure()
```

3. test_knowledge_generator.py

Tests for the KnowledgeGenerator service in `services/knowledge_generator.py`.

Test Classes:

- `TestKnowledgeGenerator` (5 tests)

Coverage:

- ✅ Service initialization with dependencies
- ✅ Processing factories with varying data states
- ✅ Log filtering and statistics calculation
- ✅ Error handling during encoding operations
- ✅ Integration with storage and embedding models

Key Tests:

```
- test_initialization()
- test_generate_knowledge_no_factories()
- test_generate_knowledge_with_logs()
- test_generate_knowledge_encode_error_handling()
```

4. test_report_generator.py

Tests for the ReportGenerator service in `services/report_generator.py`.

Test Classes:

- `TestReportGenerator` (8 tests)

Coverage:

- ✅ Query validation and error handling
- ✅ Vector similarity calculations
- ✅ Top-K filtering for relevant records
- ✅ LLM prompt formatting
- ✅ Report generation with various data states
- ✅ Handling invalid or missing embeddings

Key Tests:

```
- test_generate_report_no_query_raises_error()
- test_generate_report_valid_records()
- test_generate_report_top_k_parameter()
- test_generate_report_system_prompt_format()
```

5. test_data_encoder.py

Tests for the DataEncoder service in `services/data_encoder.py`.

Test Classes:

- `TestDataEncoder` (3 tests)

Coverage:

- ✅ Encoder initialization
- ✅ Vector encoding and storage interaction
- ✅ Support for multiple data types (employee, machine)

Running Tests

Run All Tests

```
python -m unittest discover -s tests -p "test_*.py" -v
```

Run Specific Test File

```
# Test helper functions
python -m unittest tests.test_helper -v

# Test employee utilities
python -m unittest tests.test_employee -v

# Test knowledge generator
python -m unittest tests.test_knowledge_generator -v

# Test report generator
python -m unittest tests.test_report_generator -v

# Test data encoder
python -m unittest tests.test_data_encoder -v
```

Run Specific Test Class

```
python -m unittest tests.test_helper.TestHelperFunctions -v
```

Run Specific Test Method

```
python -m unittest
tests.test_helper.TestHelperFunctions.test_percentage_basic -v
```

Testing Approach

Mocking Strategy

- **Storage Layer**: Mocked to avoid database dependencies
- **Embedding Model**: Mocked to avoid LLM dependencies
- **External Services**: All external calls are mocked

Test Isolation

- Each test is independent and can run in any order
- No shared state between tests
- Fixtures reset for each test method

Edge Cases Covered

- Division by zero
- Empty collections
- Invalid data formats
- Missing required parameters
- Error propagation

Key Features Tested

1. Data Processing

- Log aggregation and filtering
- Statistics calculation
- Performance comparisons

2. Knowledge Generation

- Factory processing workflows
- Employee-machine pair analysis
- Vector generation and storage

3. Report Generation

- Query processing
- Similarity-based ranking
- LLM integration
- Structured report formatting

4. Utility Functions

- Mathematical calculations
- Data type conversions
- Format parsing and validation

Maintenance Guide

Adding New Tests

1. Create test methods following the naming convention `test_*`
2. Use descriptive docstrings for each test
3. Follow the Arrange-Act-Assert pattern
4. Use mocks for external dependencies

Example Test Template

```
def test_feature_name(self):
    """Test description of what is being tested."""
    # Arrange
    test_data = {...}
    self.mock_service.method.return_value = expected_value

    # Act
    result = self.function_under_test(test_data)

    # Assert
    self.assertEqual(result, expected_output)
```

Running Tests During Development

```
# Run tests in verbose mode for detailed output
python -m unittest discover -s tests -p "test_*.py" -v

# Run with minimal output
python -m unittest discover -s tests -p "test_*.py"
```

Dependencies

- **Python**: 3.7+
- **unittest**: Built-in (no installation needed)
- **unittest.mock**: Built-in (no installation needed)
- **numpy**: For numerical operations in report generation tests

Important Notes

File Name Fix

The project had a mismatch between the file name and imports:

- **Original**: `utils/helper.py` (with imports as `utils.helpers`)
- **Fixed**: Renamed to `utils/helpers.py` to match imports
- This ensures consistency throughout the test suite

Test Configuration

- `conftest.py`: Configuration file for test path setup
- `__init__.py`: Makes tests directory a Python package

Coverage Summary

Module	Tests	Status
utils.helpers	12	✅ All Pass
utils.employee	12	✅ All Pass
services.knowledge_generator	5	✅ All Pass
services.report_generator	8	✅ All Pass
services.data_encoder	3	✅ All Pass
TOTAL	**40**	✅ All Pass

Best Practices Applied

1. **DRY Principle**: Common test setup in `setUp()` methods
2. **Clear Naming**: Test names clearly describe what they test
3. **Good Documentation**: Docstrings for all test methods
4. **Isolation**: No cross-test dependencies
5. **Comprehensive**: Cover happy paths, edge cases, and error conditions
6. **Fast Execution**: Mocking ensures tests run in milliseconds
7. **Maintainability**: Easy to extend with new tests

Troubleshooting

Import Errors

If you encounter import errors:

1. Ensure you're running tests from the project root directory
2. Check that `conftest.py` is present in the tests directory
3. Verify the module structure matches the imports

Test Failures

If tests fail:

1. Check the error message for specific details
2. Verify mock configurations match the actual function signatures
3. Ensure test data is appropriate for the function being tested

Future Enhancements

Potential additions for the test suite:

- Integration tests for end-to-end workflows
- Performance benchmarks
- Property-based testing with hypothesis
- Code coverage metrics with coverage.py
- CI/CD integration (GitHub Actions, Jenkins, etc.)

Author Notes

This test suite provides solid coverage of the core business logic. It uses mocking extensively to ensure tests are:

- **Fast**: Execute in milliseconds
- **Reliable**: No flaky tests due to external dependencies

- **Maintainable**: Easy to understand and modify
- **Isolated**: Each test is independent