Testing Guide - Freya v2.0

Comprehensive testing documentation for Freya v2.0.

Table of Contents

Overview

Freya v2.0 uses **pytest** as the testing framework with comprehensive coverage for:

- **Unit Tests**: Test individual components in isolation
- **Integration Tests**: Test interactions between services
- **Mocking**: Extensive mocking for external dependencies

Testing Stack

- `pytest` - Test framework
- `pytest-asyncio` - Async test support
- `pytest-cov` - Coverage reporting
- `pytest-mock` - Mocking utilities

Test Structure

```
tests/
├── __init__.py
├── conftest.py              # Shared fixtures and configuration
├── unit/                    # Unit tests
│   ├── test_message_bus.py
│   ├── test_config.py
│   ├── test_base_service.py
│   ├── test_stt_service.py
│   ├── test_tts_service.py
│   └── test_audio_manager.py
├── integration/             # Integration tests
│   └── test_audio_pipeline.py
├── mocks/                   # Mock utilities
│   ├── mock_audio.py
│   └── mock_mcp.py
└── fixtures/                # Test data
```

Running Tests

All Tests

```
pytest
```

Unit Tests Only

```
pytest tests/unit/ -v
```

Integration Tests Only

```
pytest tests/integration/ -v
```

With Coverage Report

```
pytest --cov=src --cov-report=html --cov-report=term
```

Specific Test File

```
pytest tests/unit/test_message_bus.py -v
```

Specific Test Function

```
pytest tests/unit/test_message_bus.py::TestMessageBus::test_publish -v
```

Skip Slow Tests

```
pytest -m "not slow"
```

Run Only Integration Tests

```
pytest -m integration
```

Writing Tests

Basic Test Structure

```python
import pytest
from src.services.my_service import MyService

class TestMyService:
    """Test suite for MyService."""

    @pytest.fixture
    async def service(self, mock_message_bus):
        """Create service instance for testing."""
        service = MyService(mock_message_bus)
        await service.initialize()
        yield service
        await service.stop()

    @pytest.mark.unit
    @pytest.mark.asyncio
    async def test_initialization(self, service):
        """Test service initialization."""
        assert service._healthy is True

    @pytest.mark.unit
    @pytest.mark.asyncio
    async def test_start_stop(self, service):
        """Test service lifecycle."""
        await service.start()
        assert service._running is True

        await service.stop()
        assert service._running is False
```

Using Fixtures

Freya provides comprehensive fixtures in `conftest.py`:

```python
@pytest.mark.asyncio
async def test_with_mocks(
    mock_message_bus,
    mock_redis,
    mock_ollama,
    mock_whisper,
    sample_audio_data
):
    # Your test code here
    pass
```

Async Tests

All async tests must use `@pytest.mark.asyncio`:

```python
@pytest.mark.asyncio
async def test_async_function():
    result = await some_async_function()
    assert result is not None
```

Mocking

Use `unittest.mock` for mocking:

```python
from unittest.mock import AsyncMock, patch

@pytest.mark.asyncio
async def test_with_mock():
    with patch('src.services.my_service.external_call') as mock_call:
        mock_call.return_value = "mocked response"
        result = await my_function()
        assert result == "mocked response"
```

Testing Message Bus Communication

```python
@pytest.mark.asyncio
async def test_message_publishing(mock_message_bus):
    # Publish a message
    await mock_message_bus.publish("test.channel", {"data": "test"})

    # Verify it was published
    assert mock_message_bus.publish.called
```

Test Coverage

Coverage Targets

- **New Code**: 70%+ coverage required
- **Existing Code**: 50%+ coverage target
- **Critical Services**: 80%+ coverage recommended

Viewing Coverage Reports

After running tests with coverage:

```
# Terminal report
pytest --cov=src --cov-report=term-missing

# HTML report (opens in browser)
pytest --cov=src --cov-report=html
open htmlcov/index.html

# XML report (for CI/CD)
pytest --cov=src --cov-report=xml
```

Coverage Configuration

Coverage settings are in `pyproject.toml`:

```
[tool.coverage.run]
source = ["src"]
omit = [
    "*/tests/*",
    "*/__pycache__/*",
]

[tool.coverage.report]
precision = 2
show_missing = true
exclude_lines = [
    "pragma: no cover",
    "def __repr__",
    "raise NotImplementedError",
]
```

Best Practices

1. Test Naming

- Use descriptive names: `test_audio_manager_handles_empty_data`
- Follow pattern: `test_<what>_<condition>_<expected>`

2. Test Organization

- One test class per component
- Group related tests in the same file
- Use fixtures for common setup

3. Assertions

- One assertion per test when possible
- Use descriptive assertion messages

```
assert result == expected, f"Expected {expected}, got {result}"
```

4. Mocking External Dependencies

- Always mock external services (APIs, databases, file systems)
- Use provided fixtures from `conftest.py`
- Don’t mock the code you’re testing

5. Async Test Best Practices

- Always use `@pytest.mark.asyncio`
- Clean up resources in fixtures with `yield`
- Test both success and error cases

6. Test Data

- Use fixtures for test data
- Keep test data small and focused
- Use mock generators from `tests/mocks/`

Continuous Integration

Tests run automatically on:

- Every push to feature branches
- Pull requests to master
- Scheduled daily builds

See `.github/workflows/test.yml` for CI configuration.

Troubleshooting

Tests Hang

- Check for missing `await` keywords
- Verify async fixtures are properly cleaned up
- Use `pytest -v --timeout=30` to set timeout

Import Errors

```
# Ensure you're in the project root
python -m pytest tests/

# Or set PYTHONPATH
export PYTHONPATH=$(pwd)
pytest
```

Coverage Not Showing

```
# Ensure coverage is installed
pip install pytest-cov

# Run with explicit coverage
pytest --cov=src --cov-report=term
```

Mock Not Working

- Verify you’re patching the correct import path
- Use `patch.object()` for class methods
- Check mock is called: `assert mock.called`

Resources

- pytest Documentation (https://docs.pytest.org/)
- pytest-asyncio (https://pytest-asyncio.readthedocs.io/)
- unittest.mock (https://docs.python.org/3/library/unittest.mock.html)
- Coverage.py (https://coverage.readthedocs.io/)