Testing Policy Documentation

Marito Multilingual Terminology PWA

Team Name: Velox

October 20, 2025
Version 2.0

Contents

1 Introduction

This document outlines the comprehensive testing policy and procedures for the Marito project. It describes our approach to testing, tools used, actual test implementations, continuous integration setup, and provides direct access to all testing artifacts including test suites, CI/CD pipelines, and testing reports.

1.1 Document Purpose

This policy serves as a central hub for all testing-related resources, providing:

- Direct links to testing tools and frameworks
- Access to actual test implementations in the codebase
- CI/CD pipeline configurations and execution reports
- Testing coverage reports and quality metrics
- Testing standards and best practices

2 Testing Framework

2.1 Testing Levels

Unit Testing
: Testing individual components and functions in isolation

Integration Testing
: Testing interactions between components and services

End-to-End Testing
: Testing complete user workflows and scenarios

Performance Testing
: Testing system performance under load conditions

API Testing
: Testing RESTful API endpoints for correctness and reliability

3 Testing Tools

This section provides direct links to the official documentation and repositories of all testing tools used in the Marito project.

3.1 Frontend Testing Tools

Vitest
Primary testing framework for React components
Official Documentation: https://vitest.dev/
GitHub Repository: https://github.com/vitest-dev/vitest
Purpose: Fast unit test runner with native ESM support

React Testing Library
For testing React components in a user-centric way
Official Documentation: https://testing-library.com/react
GitHub Repository: https://github.com/testing-library/react-testing-library
Purpose: Component testing with focus on user behavior

Cypress
For end-to-end testing
Official Documentation: https://www.cypress.io/
GitHub Repository: https://github.com/cypress-io/cypress
Purpose: E2E testing of user workflows

3.2 Backend Testing Tools

Pytest
Primary testing framework for Python services
Official Documentation: https://docs.pytest.org/
GitHub Repository: https://github.com/pytest-dev/pytest
Purpose: Python unit and integration testing

Coverage.py
For measuring code coverage
Official Documentation: https://coverage.readthedocs.io/
GitHub Repository: https://github.com/nedbat/coveragepy
Purpose: Code coverage measurement and reporting

Locust
For load and performance testing
Official Documentation: https://locust.io/
GitHub Repository: https://github.com/locustio/locust
Purpose: Distributed load testing of web services

Pytest-asyncio
For testing async Python code
Official Documentation: https://pytest-asyncio.readthedocs.io/
GitHub Repository: https://github.com/pytest-dev/pytest-asyncio
Purpose: Async/await support for pytest

3.3 Code Quality Tools

Ruff
Fast Python linter

Official Documentation: https://docs.astral.sh/ruff/
GitHub Repository: https://github.com/astral-sh/ruff

Black
Python code formatter
Official Documentation: https://black.readthedocs.io/
GitHub Repository: https://github.com/psf/black

MyPy
Static type checker for Python
Official Documentation: https://mypy.readthedocs.io/
GitHub Repository: https://github.com/python/mypy

ESLint
JavaScript/TypeScript linter
Official Documentation: https://eslint.org/
GitHub Repository: https://github.com/eslint/eslint

Prettier
Code formatter for frontend
Official Documentation: https://prettier.io/
GitHub Repository: https://github.com/prettier/prettier

4 Continuous Integration & Deployment

4.1 GitHub Actions

We use GitHub Actions as our CI/CD platform for the following reasons:

- Native integration with GitHub repositories
- More generous free tier for open-source projects
- Better support for matrix builds and parallel testing
- Faster build times and more concurrent jobs
- Built-in secret management and artifact storage

4.2 CI/CD Pipeline Links

4.2.1 Workflow Configurations

Direct links to workflow configuration files in the repository:

- **Main CI/CD Pipeline:**
https://github.com/COS301-SE-2025/Marito/blob/main/.github/workflows/actions.yml
- **Backend Service Deployment:**
https://github.com/COS301-SE-2025/Marito/blob/main/.github/workflows/DeployServices.yml

- **Frontend Deployment:**
  https://github.com/COS301-SE-2025/Marito/blob/main/.github/workflows/deployFront.yml
- **Database Migrations:**
  https://github.com/COS301-SE-2025/Marito/blob/main/.github/workflows/run-migrations.yml

4.2.2 Pipeline Execution Reports

View live pipeline runs, test results, and build status:

- **All Workflow Runs:**
  https://github.com/COS301-SE-2025/Marito/actions
- **CI/CD Pipeline Runs:**
  https://github.com/COS301-SE-2025/Marito/actions/workflows/actions.yml
- **Deployment Pipeline Runs:**
  https://github.com/COS301-SE-2025/Marito/actions/workflows/DeployServices.yml

5 Testing Procedure

5.1 Local Testing

5.1.1 Frontend Tests

To run frontend tests locally:

```
cd frontend
npm install
npm test                        # Run all tests
npm run test:coverage           # Run with coverage report
npm run test:ui                 # Run with Vitest UI
```

5.1.2 Backend Tests

To run backend tests locally:

```
cd backend

# Run tests for a specific service
cd auth-service
pytest                          # Run all tests
pytest --cov=app                # Run with coverage
pytest -v                       # Verbose output
pytest -k "test_login"          # Run specific test

# Run tests for all services
cd backend
bash run_tests.sh               # Linux/Mac
```

```
bash run_tests_linux.sh     # Linux
```

5.2 Automated Testing

The CI/CD pipeline automatically:

1. Runs on every push to main, master, or develop branches
2. Runs on all pull requests targeting these branches
3. Executes linting checks (Ruff, ESLint, Prettier)
4. Performs type checking (MyPy for backend)
5. Runs all unit and integration tests
6. Generates code coverage reports
7. Fails the build if any tests fail or coverage drops below threshold

6 Test Repository Structure

This section provides direct links to test files and directories in the codebase.

6.1 Frontend Tests

6.1.1 Test Directory Structure

Tests are located in the `frontend/Tests` directory:

```
frontend/
  Tests/
    components/           # Component unit tests
    pages/               # Page component tests
    utils/               # Utility function tests
    integration/         # Integration tests
    e2e/                 # End-to-end tests
```

6.1.2 Frontend Test Files

Direct links to frontend test implementations:

- **Admin Page Tests:**
  https://github.com/COS301-SE-2025/Marito/blob/main/frontend/Tests/AdminPage.test.tsx
- **Feedback Hub Tests:**
  https://github.com/COS301-SE-2025/Marito/blob/main/frontend/Tests/FeedbackHub.test.tsx

- **Feedback Page Tests:**
  https://github.com/COS301-SE-2025/Marito/blob/main/frontend/Tests/FeedbackPage.test.tsx
- **Landing Page Tests:**
  https://github.com/COS301-SE-2025/Marito/blob/main/frontend/Tests/LandingPage.test.tsx
- **New Glossary Tests:**
  https://github.com/COS301-SE-2025/Marito/blob/main/frontend/Tests/NewGlossary.test.tsx
- **Search Page Tests:**
  https://github.com/COS301-SE-2025/Marito/blob/main/frontend/Tests/SearchPage.test.tsx
- **User Profile Page Tests:**
  https://github.com/COS301-SE-2025/Marito/blob/main/frontend/Tests/UserProfilePage.test.tsx
- **Workspace Page Tests:**
  https://github.com/COS301-SE-2025/Marito/blob/main/frontend/Tests/WorkspacePage.test.tsx

6.1.3 Frontend Integration Tests

- **New Glossary API Integration:**
  https://github.com/COS301-SE-2025/Marito/blob/main/frontend/Tests/integration/NewGlossary.api.test.tsx
- **Workspace API Integration:**
  https://github.com/COS301-SE-2025/Marito/blob/main/frontend/Tests/integration/WorkspacePage.api.test.tsx

6.2 Backend Tests

6.2.1 Test Directory Structure

Each microservice has its own test directory:

```
backend/
  service-name/
    app/
      tests/
        test_*.py           # Test files
        conftest.py         # Pytest configuration
        __init__.py
```

6.2.2 Backend Test Files by Service

Authentication Service Tests

- **Auth Endpoints Tests:**
  https://github.com/COS301-SE-2025/Marito/blob/main/backend/auth-service/app/tests/test_auth.py
- **Admin Functions Tests:**
  https://github.com/COS301-SE-2025/Marito/blob/main/backend/auth-service/app/tests/test_admin.py

Search Service Tests

- **Search Functionality Tests:**
  https://github.com/COS301-SE-2025/Marito/blob/main/backend/search-service/app/tests/test_search.py
- **Autocomplete/Suggest Tests:**
  https://github.com/COS301-SE-2025/Marito/blob/main/backend/search-service/app/tests/test_suggest.py
- **CRUD Operations Tests:**
  https://github.com/COS301-SE-2025/Marito/blob/main/backend/search-service/app/tests/test_crud_search.py

Analytics Service Tests

- **Analytics Unit Tests:**
  https://github.com/COS301-SE-2025/Marito/blob/main/backend/analytics-service/app/tests/test_analytics_unit.py
- **Analytics CI Tests:**
  https://github.com/COS301-SE-2025/Marito/blob/main/backend/analytics-service/app/tests/test_analytics_ci.py

Feedback Service Tests

- **Feedback API Tests:**
  https://github.com/COS301-SE-2025/Marito/blob/main/backend/feedback-service/app/tests/test_feedback_api.py
- **Feedback CRUD Tests:**
  https://github.com/COS301-SE-2025/Marito/blob/main/backend/feedback-service/app/tests/test_feedback_crud.py
- **RBAC Tests:**
  https://github.com/COS301-SE-2025/Marito/blob/main/backend/feedback-service/app/tests/test_rbac.py

Glossary Service Tests

- **Glossary API Tests:**
  https://github.com/COS301-SE-2025/Marito/blob/main/backend/glossary-service/app/tests/test_glossary.py
- **Unit Functions Tests:**
  https://github.com/COS301-SE-2025/Marito/blob/main/backend/glossary-service/app/tests/test_unit_functions.py
- **Glossary Integration Tests:**
  https://github.com/COS301-SE-2025/Marito/blob/main/backend/glossary-service/app/tests/test_glossary_integration.py

Vote Service Tests

- **Vote Endpoint Tests:**
  https://github.com/COS301-SE-2025/Marito/blob/main/backend/vote-service/app/tests/test_vote_endpoint.py

Workspace Service Tests

- **Main Workspace Tests:**
  https://github.com/COS301-SE-2025/Marito/blob/main/backend/workspace-service/app/tests/test_main.py
- **Bookmarks Tests:**
  https://github.com/COS301-SE-2025/Marito/blob/main/backend/workspace-service/app/tests/test_bookmarks.py
- **Groups Tests:**
  https://github.com/COS301-SE-2025/Marito/blob/main/backend/workspace-service/app/tests/test_groups.py
- **Notes Tests:**
  https://github.com/COS301-SE-2025/Marito/blob/main/backend/workspace-service/app/tests/test_notes.py
- **Workspace Integration Tests:**
  https://github.com/COS301-SE-2025/Marito/blob/main/backend/workspace-service/app/tests/test_workspace_integration.py

7 Testing Reports & Coverage

7.1 Generating Coverage Reports

7.1.1 Frontend Coverage

Generate frontend test coverage reports:

```
cd frontend
npm run test:coverage
# Coverage report will be generated in frontend/coverage/
# Open frontend/coverage/index.html in a browser
```

7.1.2 Backend Coverage

Generate backend test coverage reports:

```
cd backend/auth-service
pytest --cov=app --cov-report=html
# Coverage report will be generated in htmlcov/
# Open htmlcov/index.html in a browser

# For all services combined
cd backend
pytest --cov=. --cov-report=html --cov-report=term
```

7.2 Coverage Report Storage

Coverage reports are stored in the following locations:

- **Frontend Coverage:** `frontend/coverage/`
- **Backend Coverage (per service):** `backend/[service-name]/htmlcov/`
- **CI/CD Artifacts:** Available in GitHub Actions workflow run artifacts

Note: Coverage HTML reports are gitignored and must be generated locally or downloaded from CI/CD artifacts.

7.3 Accessing CI/CD Test Reports

To view test results from CI/CD pipeline runs:

1. Navigate to https://github.com/COS301-SE-2025/Marito/actions
2. Click on any workflow run
3. Expand the test execution steps to view:
    - Test pass/fail status
    - Test execution logs
    - Coverage percentages
    - Linting and type checking results
4. Download artifacts (if any) for detailed HTML coverage reports

7.4 Test Report Archive

Historical test reports and quality metrics are maintained in:

- **Documentation Reports Directory:**
  https://github.com/COS301-SE-2025/Marito/tree/main/Documentation/reports
- **GitHub Actions History:**
  Accessible through the Actions tab for the last 90 days of runs

8 Testing Standards

8.1 Coverage Requirements

- **Minimum Code Coverage:** 80% for new features
- **Critical Paths:** 100% coverage required
- **API Endpoints:** Integration tests for all endpoints
- **User Workflows:** E2E tests for main user workflows
- **Edge Cases:** Explicit tests for error handling and edge cases

8.2 Testing Best Practices

8.2.1 General Principles

- Write tests before or alongside implementation (TDD/BDD)
- Keep tests focused, atomic, and independent
- Use meaningful and descriptive test names
- Mock external dependencies and services
- Follow AAA pattern (Arrange-Act-Assert)
- Ensure tests are deterministic and repeatable
- Clean up test data and resources after execution

8.2.2 Frontend Testing Best Practices

- Test user interactions, not implementation details
- Use accessibility queries (getByRole, getByLabelText)
- Avoid testing internal component state directly
- Test error states and loading states
- Mock API calls consistently

8.2.3 Backend Testing Best Practices

- Use fixtures for common test setup
- Test both success and failure scenarios
- Validate request/response schemas
- Test authentication and authorization
- Test database transactions and rollbacks
- Use async test support for async endpoints

8.3 Test Naming Conventions

Frontend
`ComponentName.test.tsx` or `functionName.test.ts`

Backend
`test_feature_name.py`

Integration
`test_feature_integration.py`

E2E
`test_workflow_e2e.py`

8.4 Pull Request Requirements

Before merging any pull request:

1. All tests must pass in CI/CD pipeline
2. Code coverage must meet minimum thresholds
3. No linting or type checking errors
4. All new features must include tests
5. Test descriptions must be clear and meaningful
6. Complex logic must have unit tests

9 Testing Tool Configuration Files

Direct links to testing configuration files:

- **Frontend Test Config (Jest):**
  https://github.com/COS301-SE-2025/Marito/blob/main/frontend/jest.config.js

- **Frontend Vite Config:**
  https://github.com/COS301-SE-2025/Marito/blob/main/frontend/vite.config.ts
- **Backend Pytest Config:**
  Configuration in `pyproject.toml` files per service
- **MyPy Configuration:**
  https://github.com/COS301-SE-2025/Marito/blob/main/backend/mypy.ini
- **ESLint Configuration:**
  https://github.com/COS301-SE-2025/Marito/blob/main/frontend/eslint.config.js