# Testing Policy Documentation
## Marito Multilingual Terminology PWA

Team Name: Velox

September 7, 2025

## Contents


**1 Introduction** **2**

**2 Testing Framework** **2**
2.1 Testing Levels . . . . . 2

**3 Testing Tools** **2**
3.1 Frontend Testing . . . . . 2
3.2 Backend Testing . . . . . 2

**4 Continuous Integration** **2**
4.1 GitHub Actions . . . . . 2

**5 Testing Procedure** **3**
5.1 Local Testing . . . . . 3
5.2 Automated Testing . . . . . 3

**6 Test Repository Structure** **3**
6.1 Frontend Tests . . . . . 3
6.2 Backend Tests . . . . . 3

**7 Test Reports** **3**

**8 Testing Standards** **4**
8.1 Coverage Requirements . . . . . 4
8.2 Testing Best Practices . . . . . 4

**9 Git Repository** **4**

# 1. Introduction

This document outlines the testing policy and procedures for the Marito project. It describes our approach to testing, tools used, and the processes we follow to ensure high-quality software delivery.

# 2. Testing Framework

## 2.1. Testing Levels

- **Unit Testing**: Testing individual components and functions
- **Integration Testing**: Testing interactions between components
- **End-to-End Testing**: Testing complete user workflows
- **Performance Testing**: Testing system performance under load

# 3. Testing Tools

## 3.1. Frontend Testing

- **Jest**: Primary testing framework for React components
- **React Testing Library**: For testing React components in a user-centric way
- **Cypress**: For end-to-end testing

## 3.2. Backend Testing

- **Pytest**: Primary testing framework for Python services
- **Coverage.py**: For measuring code coverage
- **Locust**: For load testing

# 4. Continuous Integration

## 4.1. GitHub Actions

We use GitHub Actions as our CI/CD platform instead of Travis CI 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

# 5. Testing Procedure

## 5.1. Local Testing

1. Developers must write tests for new features
2. All tests must pass locally before committing
3. Run `npm test` for frontend tests
4. Run `pytest` for backend tests

## 5.2. Automated Testing

1. Tests run automatically on pull requests
2. Code coverage reports are generated
3. Performance tests run nightly
4. Security scanning is performed on dependencies

# 6. Test Repository Structure

## 6.1. Frontend Tests

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

```
frontend/
  Tests/
    components/
    pages/
    utils/
    e2e/
```

## 6.2. Backend Tests

Each service has its own tests directory:

```
backend/
  service-name/
    tests/
      unit/
      integration/
      e2e/
```

# 7. Test Reports

Test reports are automatically generated and stored in the following locations:

- Frontend coverage: `frontend/coverage/`
- Backend coverage: `backend/*/htmlcov/`

- E2E test videos: `frontend/cypress/videos/`
- Performance test reports: `backend/performance-reports/`

## 8. Testing Standards

### 8.1. Coverage Requirements

- Minimum 80% code coverage for new features
- Critical paths must have 100% coverage
- Integration tests for all API endpoints
- E2E tests for main user workflows

### 8.2. Testing Best Practices

- Write tests before implementation (TDD)
- Keep tests focused and atomic
- Use meaningful test descriptions
- Mock external dependencies
- Follow AAA pattern (Arrange-Act-Assert)

## 9. Git Repository

All test cases and reports can be found in our GitHub repository:

- Repository: `https://github.com/COS301-SE-2025/Marito`
- Test Documentation: `/Documentation/Testing Policy/`
- CI/CD Workflows: `/.github/workflows/`