Swift Signals

Testing Policy

Project Testing Policy

This document defines the **standards, tools, and practices** for testing and monitoring within the project. Its objective is to ensure the system is **reliable, maintainable, performant, and observable**, both during development and in production.

Tools Summary

Tool / Framework	**Type**	**Language / Platform**	**Purpose / Coverage**
flake8	Linting	Python	Enforce code standards and detect errors
black	Formatting	Python	Ensure consistent code style
pytest	Unit Testing	Python	Validate correctness of Python code modules
ESLint	Linting	JavaScript / TypeScript	Enforce code standards and detect errors
Prettier	Formatting	JavaScript / TypeScript	Ensure consistent code style
Jest	Unit Testing	JavaScript / TypeScript	Validate correctness of frontend modules
go test	Unit Testing	Go	Validate correctness of Go backend modules
Cypress	End-to-End Testing	JS / Browser	Test full user workflows and integration
Apache JMeter	Load / Performance Testing	Any	Test throughput, latency, and scalability under load
Sentry	Error Monitoring	Multi-platform	Capture runtime errors, crashes, and exceptions
Pingdom	Uptime & Performance Monitor	Multi-platform	Monitor uptime and response times in production

1. Purpose

The purpose of this policy is to:

- Ensure code quality and maintainability.
- Validate functional correctness through automated tests.
- Ensure performance and load handling under realistic conditions.
- Monitor production reliability and detect errors early.

2. Scope

This policy applies to all codebases (frontend, backend, scripts) and covers:

- Code quality (linting and formatting)
- Functional correctness (unit and end-to-end testing)
- Performance and load testing
- Production monitoring and error tracking

3. Testing Standards

3.1 Code Quality

Linting:

- Python: flake8
- JavaScript/TypeScript: ESLint
- Go: golangci-lint

Requirement:
All code must pass linting before merging. CI pipelines enforce linting.

Formatting:

- Python: black
- JavaScript/TypeScript: Prettier

Requirement:
Code must follow standard formatting rules. Automated pre-commit hooks or CI pipelines enforce compliance.

Reasoning:
Consistent and clean code reduces bugs and eases collaboration. Automated linting and formatting ensure **all commits adhere to standards**, regardless of the developer.

3.2 Unit Testing

Tooling:

- Python: pytest
- JavaScript/TypeScript: Jest
- Go: go test

Requirement:

- All new features or bug fixes must include corresponding unit tests.
- CI pipelines must run all unit tests successfully before merging.
- Unit tests should aim for high coverage while remaining maintainable.

Reasoning:
Unit tests form the **foundation of code reliability**, allowing safe refactoring and confidence that modules behave correctly before integration.

3.3 End-to-End Testing

Tooling: Cypress

Requirement:

- Critical user workflows (login, dashboard, forms, checkout) must have automated end-to-end tests.
- Tests must run in CI pipelines on feature branches and before production deployment.

Reasoning:
Cypress ensures the **whole system works as expected from the user perspective**, safeguarding critical workflows.

3.4 Performance and Load Testing

Tooling: Apache JMeter

Requirement:

- New features affecting performance must be evaluated with load tests.
- Baseline metrics (response time, throughput, error rates) must be established for each environment.
- Load testing should be run before major releases.

Reasoning:
Performance testing ensures the system is **responsive and resilient** under expected and peak loads.

3.5 Production Monitoring

Error Tracking: Sentry

- All exceptions and errors must be reported and assigned for investigation.
- Release tracking must be used to correlate issues with deployments.

Uptime Monitoring: Pingdom

- Synthetic checks ensure service availability and response time compliance with SLAs.
- Alerts must be configured for downtime or performance degradation.

Reasoning:
Monitoring ensures **production reliability**, enabling rapid detection and resolution of issues.

4. Testing Workflow

1. **Development Phase:**
    - Code must pass linting (flake8, ESLint) and formatting (black, Prettier`).
    - Unit tests must be written and pass for all new or modified code.
    - Local end-to-end tests may be run for critical workflows.
2. **Pull Request & CI Phase:**
    - CI pipeline runs:
        - Linting and formatting checks
        - Unit tests (pytest, Jest, go test)
        - Cypress end-to-end tests
    - Pull requests **cannot be merged** if any checks fail.
3. **Pre-Release / Staging Phase:**
    - Run performance/load tests (JMeter) if changes affect throughput, concurrency, or critical flows.
    - Validate Sentry error monitoring and ensure alerting is configured.
4. **Production Phase:**
    - Continuous monitoring via Pingdom and Sentry.
    - Critical production incidents trigger review, root cause analysis, and new tests if needed.

5. Policy Enforcement

- Compliance with linting, formatting, and testing standards is **mandatory**.
- CI/CD pipelines will **block merges** for non-compliance.
- Regular audits of test coverage, performance metrics, and monitoring alerts will be performed.
- All team members must follow this policy when contributing code or managing deployments.

6. Policy Review

- This policy will be reviewed **quarterly** or after major tooling or process changes.
- Adjustments will be communicated to all team members to ensure continuous improvement of code quality, reliability, and performance.