Non-functional testing & Monitoring

Non-Functional Testing and Monitoring

This page outlines the **non-functional testing, code quality, and monitoring strategies** used in the project. These tools ensure the system is **maintainable, reliable, performant, and observable** during production.

1. Unit Testing

Tooling:

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

Purpose:

- Verifies **individual components or functions** in isolation.
- Ensures logic is correct and behaves as expected.
- Catches regressions early during development or CI.

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

2. Linting

Tooling:

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

Purpose:

- Enforces **coding standards and best practices**.
- Detects **syntax errors, unused variables, and anti-patterns**.
- Improves code readability and maintainability.

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

3. Formatting

Tooling:

- Python: black
- JavaScript/TypeScript: prettier

Purpose:

- Ensures **consistent code style** across the codebase.
- Automatically reformats code to a standard style, removing debates over style.

Reasoning:
Consistent formatting makes code **easier to read and maintain**, allowing developers to focus on functionality rather than style.

4. Sentry (Error Monitoring & Observability)

Tooling: Sentry integrated in both backend and frontend.

Purpose:

- Tracks **runtime errors and exceptions** in real-time.
- Provides **stack traces, user context, and environment details**.
- Supports **release tracking**, so errors can be traced to specific deployments.

Reasoning:
Sentry ensures **production reliability**, allowing developers to triage, fix, and prevent recurrence of errors quickly.

5. Cypress (End-to-End Testing)

Tooling: Cypress automated testing framework.

Purpose:

- Tests **complete user workflows** in the browser.
- Validates integration between frontend and backend.
- Detects regressions in real-world user interactions.

Reasoning:
Cypress ensures the **whole system works as expected from the user perspective**, safeguarding critical flows such as forms, logins, or dashboards.

6. Pingdom (Uptime & Performance Monitoring)

Tooling: Pingdom synthetic monitoring.

Purpose:

- Monitors **service uptime and availability** from external locations.
- Tracks **response times and latency metrics**.
- Sends alerts for outages or performance issues.

Reasoning:
Pingdom validates **system reliability from a user perspective**, ensuring SLAs are met and downtime is detected immediately.

7. JMeter (Load & Performance Testing)

Tooling: Apache JMeter

Purpose:

- Simulates **high traffic and concurrent users**.
- Measures **system performance, throughput, and response times** under load.
- Identifies bottlenecks and ensures the system can **handle peak demand**.

Reasoning:
Performance testing ensures the system is **responsive and resilient** under expected and peak loads, complementing functional and unit tests.

8. Integration & CI/CD Strategy

- **Linting and formatting** (flake8, black, ESLint, Prettier) prevent style drift and enforce best practices.
- **Unit tests** (pytest, Jest, go test) validate correctness of code modules.
- **Cypress E2E tests** ensure workflows function correctly.
- **Sentry and Pingdom** provide production observability.
- **JMeter** validates performance and load handling.

Overall Goal:
Create a system that is **maintainable, correct, safe, performant, and reliable** across development, staging, and production environments.