Issues API Testing Strategy

Designing and Planning for Effective Automation and Performance Testing

Suhila Ahmed

Agenda

- Introduction
- Objectives
- Strategy
- Design
- Plan
- Execution
- Maintenance
- Conclusion

Introduction

- **Purpose** : This document provides a comprehensive testing strategy for the GitLab API, focusing on the "Issues" endpoints. It aims to ensure the API's functionality, reliability, performance, and CI/CD through systematic testing approaches.

- **Scope**: The testing will cover the "Issues" API endpoint for CRUD operations, which allows for creating, listing, editing, and deleting issues within GitLab project.

Objectives

- Validate the functional aspects of the Issues API.
- Ensure the positive and negative scenarios meets the correct outcomes.
- Ensure the API meets performance benchmarks.
- Automate testing processes for continuous integration and deployment.

Strategy

Test Levels:

- **Integration Testing:** Ensure the API integrates properly with other parts of the GitLab ecosystem.
- **Performance Testing:** Assess the API's response time and scalability.

Strategy

Test Types:

- **Functional Testing:** Verify the API behaves as expected.
- **Negative Testing:** Ensure the API handles bad input or unexpected user behaviour gracefully.
- **Load Testing:** Determine how the API performs under heavy load conditions.
- **Stress Testing:** Identify the API's breaking point or maximum operational capacity**.**
- **Soak Testing:** load testing a system continuously and monitoring for memory leaks and behaviour of the system.

Design

Test Environment:

- Access to a GitLab instance with various projects and permissions set up for testing.

Tools And Technologies:

- Playwright for automated testing of API endpoints
- K6 for performance testing.
- Github CI/CD for integrating testing into the software development lifecycle.

Test Data Management:

- Scripts to generate and clean up test data before and after test runs.

Plan

Test cases

- **Create Issue:** Validate creating a new issue with various parameters.
- **List Issues:** Test listing all issues for a project, with filters applied.
- **Update Issue:** Verify editing existing issues, including state changes.
- **Delete Issue:** Ensure issues can be deleted properly.

Scope of test cases:

- To test positive flows of the Issues API CRUD operation.
- To test different error flows (400, 401, 404 and 500).
- To make sure the api functionality is behaving as described.
- To expect the behaviour of the UI.

Plan

Test Execution Plan for API testing with playwright

Automated Testing:

- Using playwright to build a structured API testing framework.
- Integrate tests into the CI/CD pipeline to run on every commit.

Design Patterns:

- **DRY :** Don’t repeat yourself
- **KISS:** Keep it simple stupid
- **Builder Pattern.**

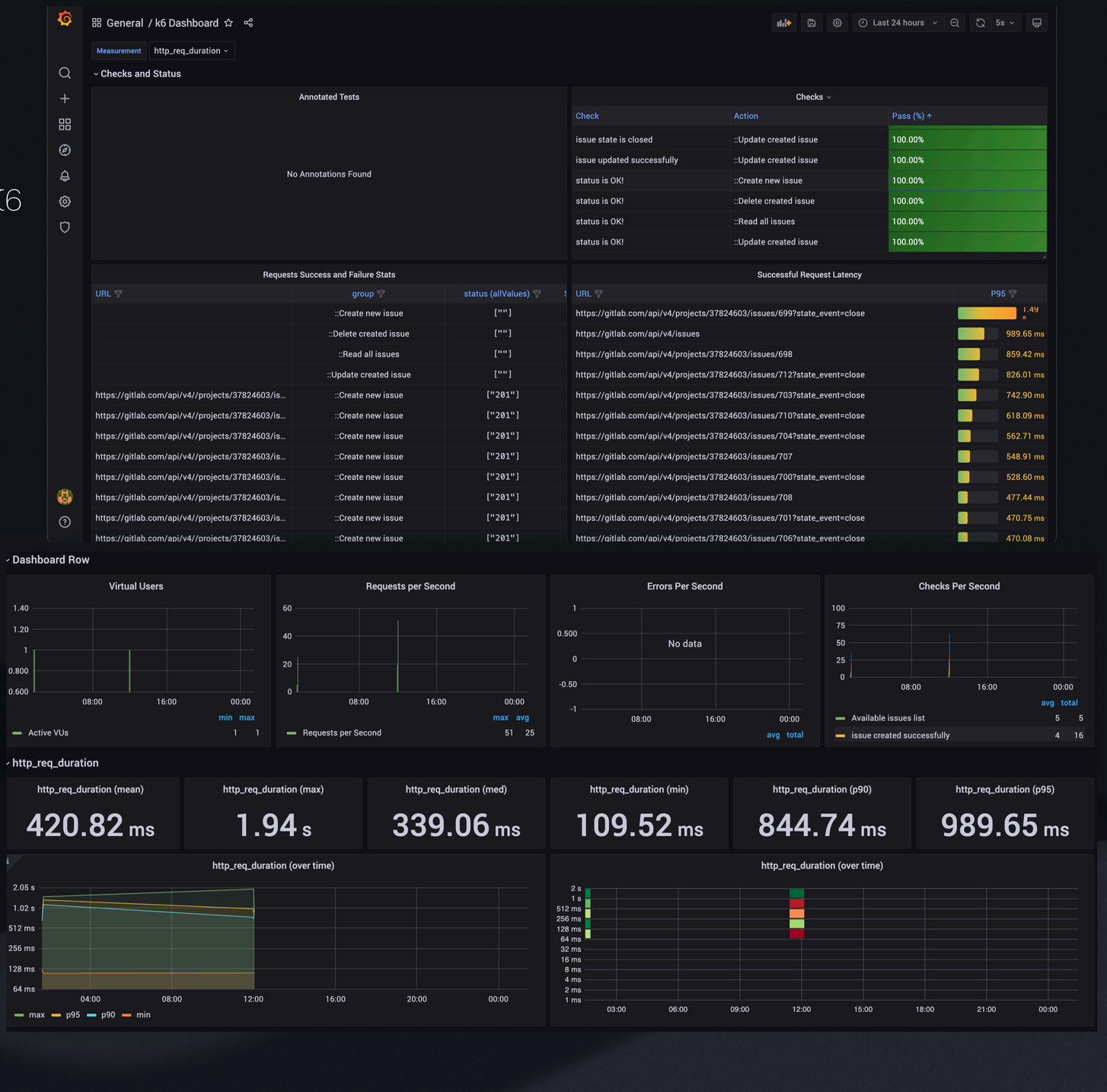
```
/playwright-api-testing
    /tests
        /api
            /specs            # Test files for API tests
                create-issue.spec.ts
                update-issue.spec.ts
                delete-issue.spec.ts
                read-issues.spec.ts
            /helpers          # Utility functions specific to API testing
                assertions.ts
    /models                   # Data models or schemas for API responses
        issues.ts
    /data                     # Test data directory
        userData.json
        productData.json
    /resources                # Wraps Playwright API requests for customization
        /issuesApi.ts

    /utils
        /test-data            # Common utilities for the framework
            issues-data.ts    # Generates test data

    playwright.config.ts      # Playwright configuration
    api.config.ts             # API specific configurations like baseURL
    /allure-reports           # Test reports (e.g., HTML, JSON)
    /node_modules             # Node.js modules
    package.json              # Project metadata and dependencies
    README.md                 # Project documentation
```

Plan

Test Execution Plan for API performance testing with K6

Performance testing:

- Using K6 to build a structured performance testing scripts.
- Integrate scripts into the CI/CD pipeline to run on a specific schedule.
- User InfluxDB and Grafana to view live test results.



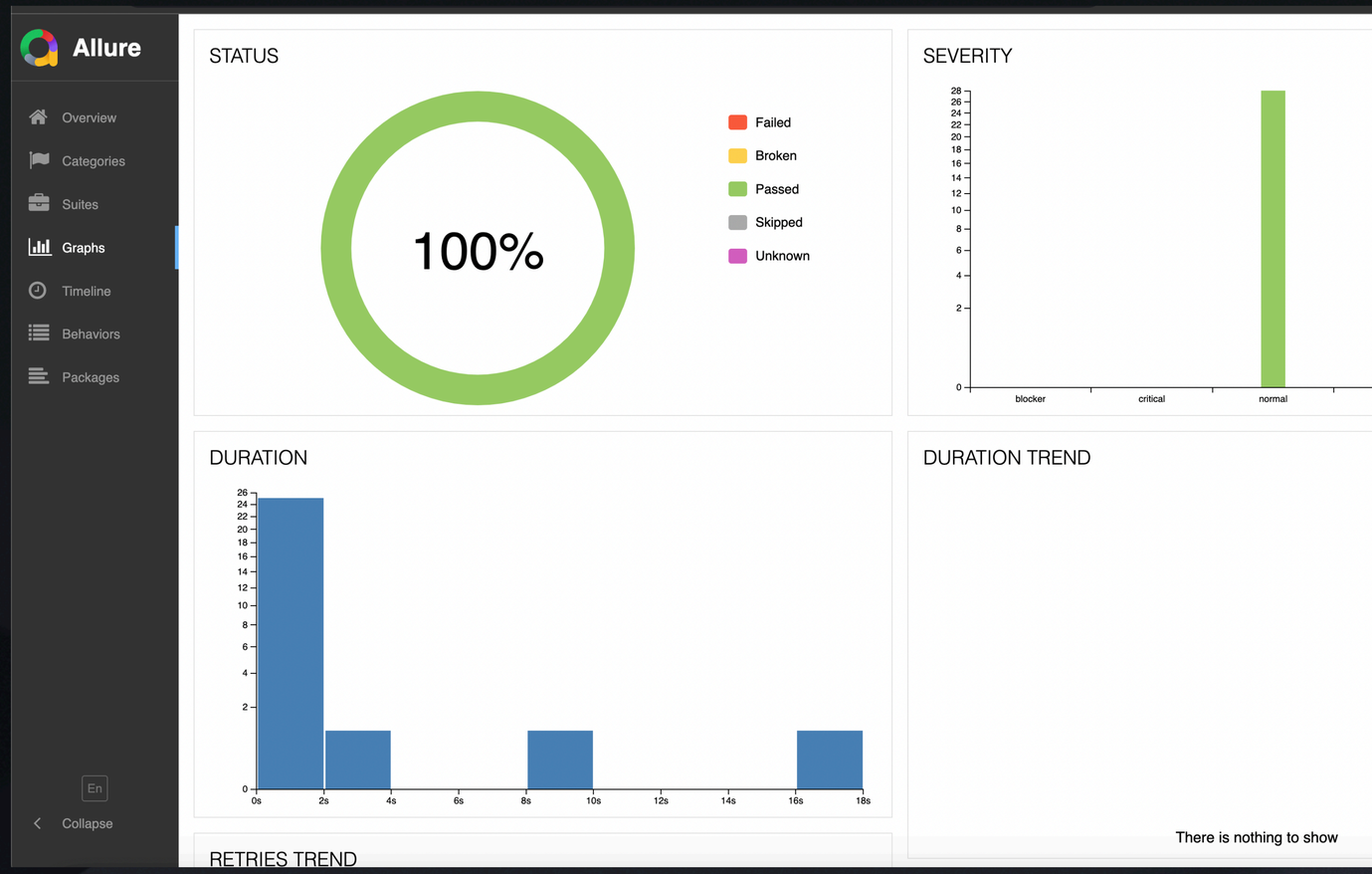
Execution

Continuous Integration

- Use Github CI/CD pipelines to automate the execution of test suites.
- Configure pipelines to trigger tests based on specific events, such as push or merge requests.

Monitoring and Reporting

- Real-time monitoring of test executions through Github CI/CD dashboards.
- Detailed test reports generated after each test run, including pass/fail status, performance metrics, and security vulnerabilities.
  - Allure Reports for Playwright
  - InfluxDB and Grafana for K6



Maintenance

- Regularly review and update test cases to reflect changes in the API.
- Refactor tests to improve efficiency and coverage.
- Maintain comprehensive documentation of test cases, test data, and testing procedures.
- Document all findings and fixes for future reference.

Conclusion

The design and planning phases are critical for setting a solid foundation for the testing process. A well-formulated testing strategy ensures that testing efforts are aligned with project objectives, resources are effectively utilized, and risks are managed proactively. By following these steps, teams can create a comprehensive and effective testing strategy that contributes to the delivery of high-quality software.

Questions