VIRNECT

QAOps-Based Quality Automation & Performance Management

QAOps Implementation & Results

VIRNECT QA Team

Sungtae Kim

I. Project Overview

Purpose

- Establish a test automation pipeline and standardize code quality measurement to systematically ensure overall project quality.
- Enhance collaboration efficiency between development and QA teams by integrating Agile and CI/CD processes.
- Centralize multi-stack environments (Frontend, Backend, Unity C#) into a single SonarQube dashboard for unified quality visualization.
- Strengthen release stability by implementing a continuous regression testing framework and fostering a team-wide quality culture.
- Conduct Postman-based API functional testing to validate backend API behavior and data integrity.
- Perform JMeter-based performance testing and OWASP ZAP-based security testing to proactively verify not only functional quality but also non-functional quality aspects such as performance and security.

Goals

1. Build a Static Analysis Environment for Automated Test Code
• Integrate SonarQube into the Jenkins pipeline to identify code smells, bugs, and security vulnerabilities at an early stage.
2. Secure and Visualize Test Coverage
• Use tools such as Jest, JaCoCo, and Istanbul to measure Statement, Branch, and Line coverage with automated reporting.
3. Enforce CI/CD Quality Gates
• Apply merge-blocking policies on pull requests when quality gate conditions are not met to mitigate risks in advance.
4. Strengthen Development–QA Collaboration
• Share test and quality reports in real time through Jenkins, Slack, and GitHub to facilitate Agile team collaboration.
5. Reduce Code Duplication and Promote Modular Refactoring
• Lower duplication rates and introduce common modules to reduce maintenance costs and improve development productivity.
6. Implement Unified Quality Management Across Multi-Stack Projects
• Standardize overall project quality metrics by managing Java, TypeScript, and C# codebases on a single platform.

I. Project Overview

Goals

7. Establish a Postman-Based API Testing Framework
• Validate request/response behavior, status codes, and data consistency through unit-level and scenario-based API tests.
• Improve regression stability by automating repeated test executions and maintaining structured test reports.
8. Establish a JMeter-Based Performance Testing Framework
• Execute load and concurrency tests on login flows, core APIs, and critical user scenarios to verify TPS, response time, and system stability.
• Manage performance test results as quantitative metrics to define clear release criteria.
9. Conduct OWASP ZAP-Based Security Testing
• Perform vulnerability scans on web and API endpoints to proactively identify security risks based on the OWASP Top 10.
• Document security test results to systematically track remediation actions and maintain security test history.

I. Project Overview

Schedule

No.	Schedule	Category	Key Tasks	Responsible Person		Remarks
1	Monday, April 14, 2025	Configuration & Integration	• Configure advanced quality gates and rules per tech stack (Java, TS, C#). • Implement branch coverage thresholds in CI. • Automate Slack and GitHub PR notifications for quality gate results.	VIRNECT Co., Ltd. | QA Team	Kim Sung-tae, Senior Engineer	
2	Tuesday, May 13, 2025	Test Coverage Expansion	• Expand unit test coverage with Jest, JUnit, and Istanbul. • Identify and address coverage blind spots. • Create coverage heatmap dashboards in SonarQube.	VIRNECT Co., Ltd. | QA Team	Kim Sung-tae, Senior Engineer	Target coverage ≥80% (backend)
3	Monday, June 16, 2025	Automation & Optimization	• Optimize Jenkins pipeline parallelization for faster builds. • Integrate caching mechanisms for test runs. • Refactor repetitive modules to reduce duplication.	VIRNECT Co., Ltd. | QA Team	Kim Sung-tae, Senior Engineer	
4	Monday, July 14, 2025	Security & Vulnerability	• Enable SonarQube security hotspot analysis. • Conduct vulnerability scanning and remediation. • Establish monthly security quality gate reviews.	VIRNECT Co., Ltd. | QA Team	Kim Sung-tae, Senior Engineer	
5	Monday, August 11, 2025	Stabilization & Reporting	• Finalize dashboards for CI/CD & QA metrics. • Automate quarterly test coverage & quality trend reports. • Conduct knowledge-sharing sessions across teams.	VIRNECT Co., Ltd. | QA Team	Kim Sung-tae, Senior Engineer	Quarterly summary planned

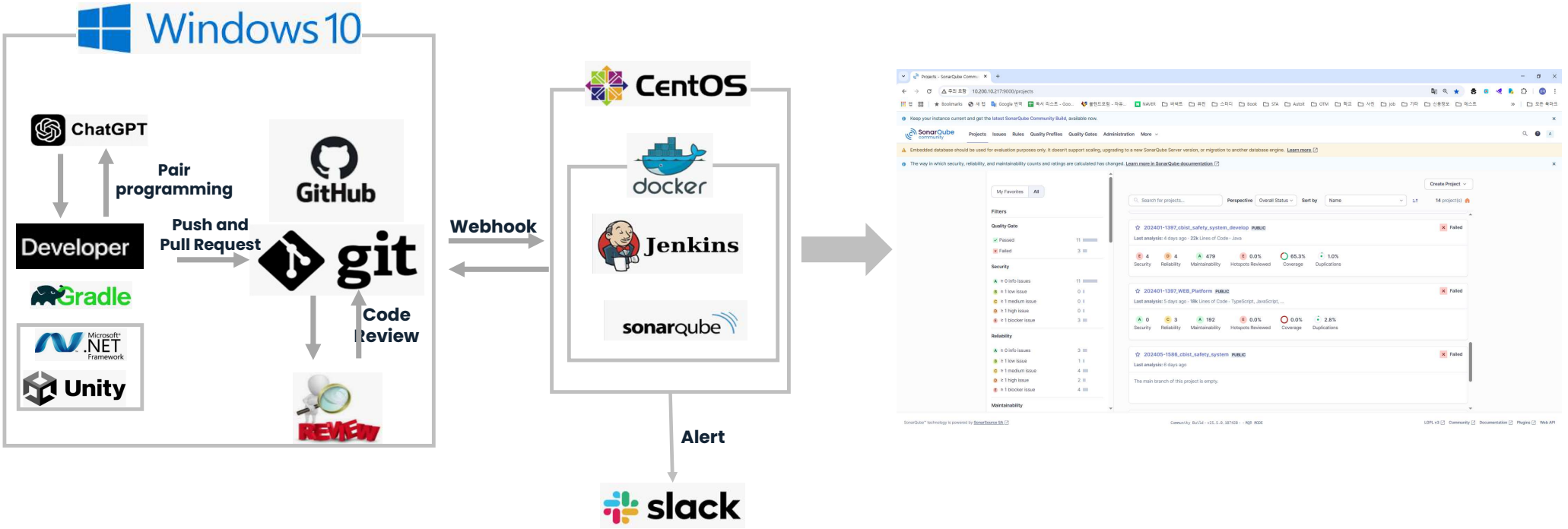
I. Project Overview

Schedule

No.	Schedule	Category	Key Tasks	Responsible Person		Remarks
6	August 26, 2025 (Tuesday)	Performance Test Planning	• Design JMeter-based performance test scenarios. • Establish load and concurrency test plans for key APIs and critical user flows. • Define performance metrics such as TPS, response time, and error rate.	VIRNECT Co., Ltd. | QA Team	Kim Sung-tae, Senior Engineer	
7	August 29, 2025 (Friday)	Performance Testing (Internal)	• Execute internal performance tests using JMeter. • Analyze test results and identify system bottlenecks. • Document performance test results and derive improvement actions.	VIRNECT Co., Ltd. | QA Team	Kim Sung-tae, Senior Engineer	
8	December 1, 2025 (Monday)	Final Performance Testing	• Execute final performance tests on the system with applied improvements. • Verify whether target performance metrics (TPS, response time, stability) are satisfied. • Prepare the final performance test report and assess release readiness.	VIRNECT Co., Ltd. | QA Team	Kim Sung-tae, Senior Engineer	
9	December 8, 2025 (Monday) - December 12, 2025 (Friday)	API Integration Testing	• Execute API integration tests for Platform and Admin APIs. • Validate authentication, request/response, and integration flows. • Identify defects and document test results for release readiness.	VIRNECT Co., Ltd. | QA Team	Kim Sung-tae, Senior Engineer	
10						

I. Project Overview

Building an Integrated Quality Management Pipeline (Java + TS + C#)



‹Manage all stacks on a single SonarQube dashboard›

I. Project Overview

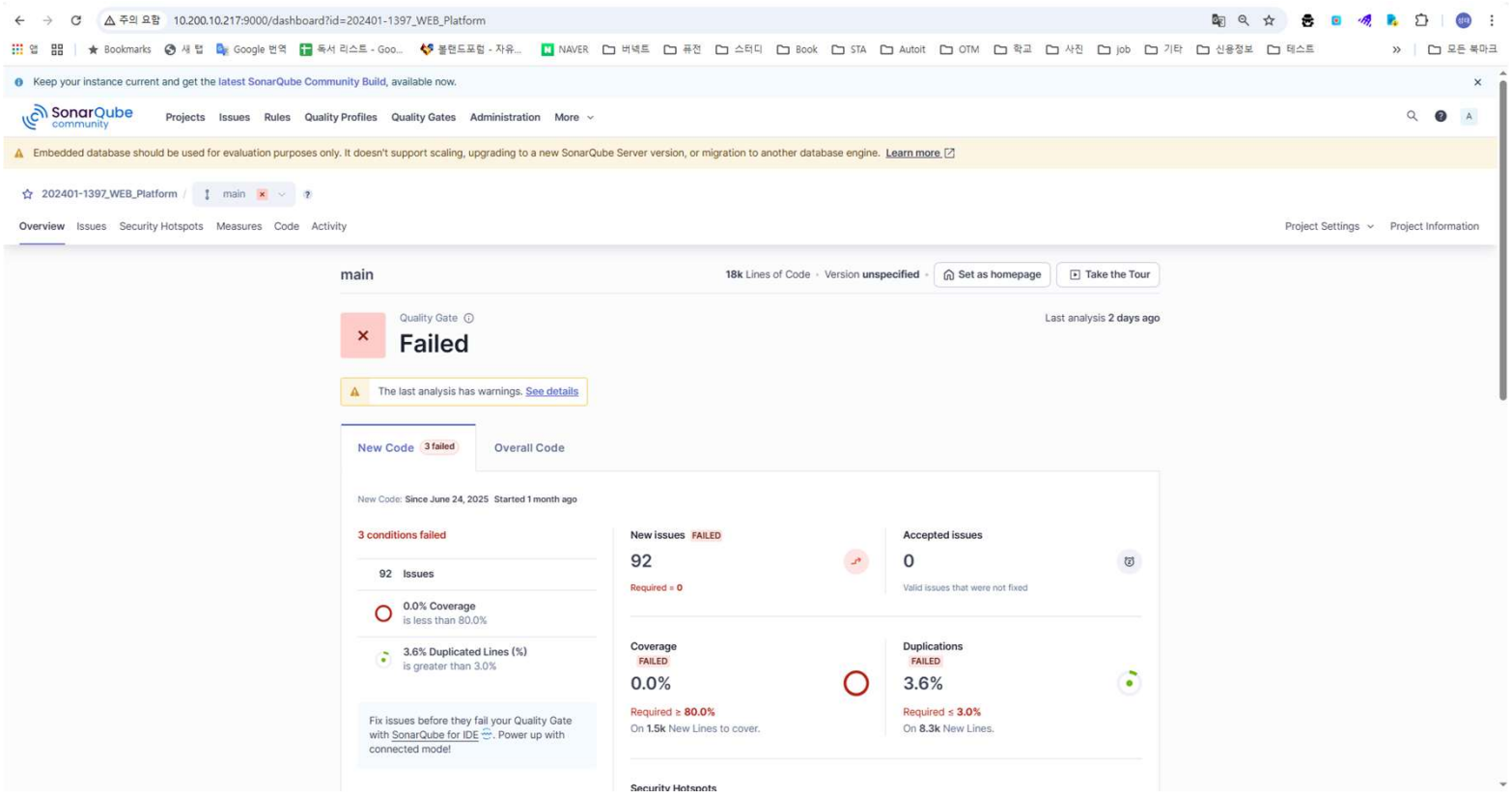
1. SonarQube Static Analysis Status – Frontend (TypeScript)

- **Language / Environment**
  - TypeScript (Web Frontend)

- **Analysis Metrics**
  - Coverage: 0.0% (Needs collection/integration)
  - New issues: 92
  - Duplication: 3.6% (Exceeds threshold of ≤3.0%)
  - Quality Gate: X Failed (3 conditions failed)
  - Last analysis: 2 days ago

- **Key Points**
  - Current coverage not collected → Plan to integrate with Jest lcov.
  - Code duplication and new issues are the main reasons for Quality Gate failure.

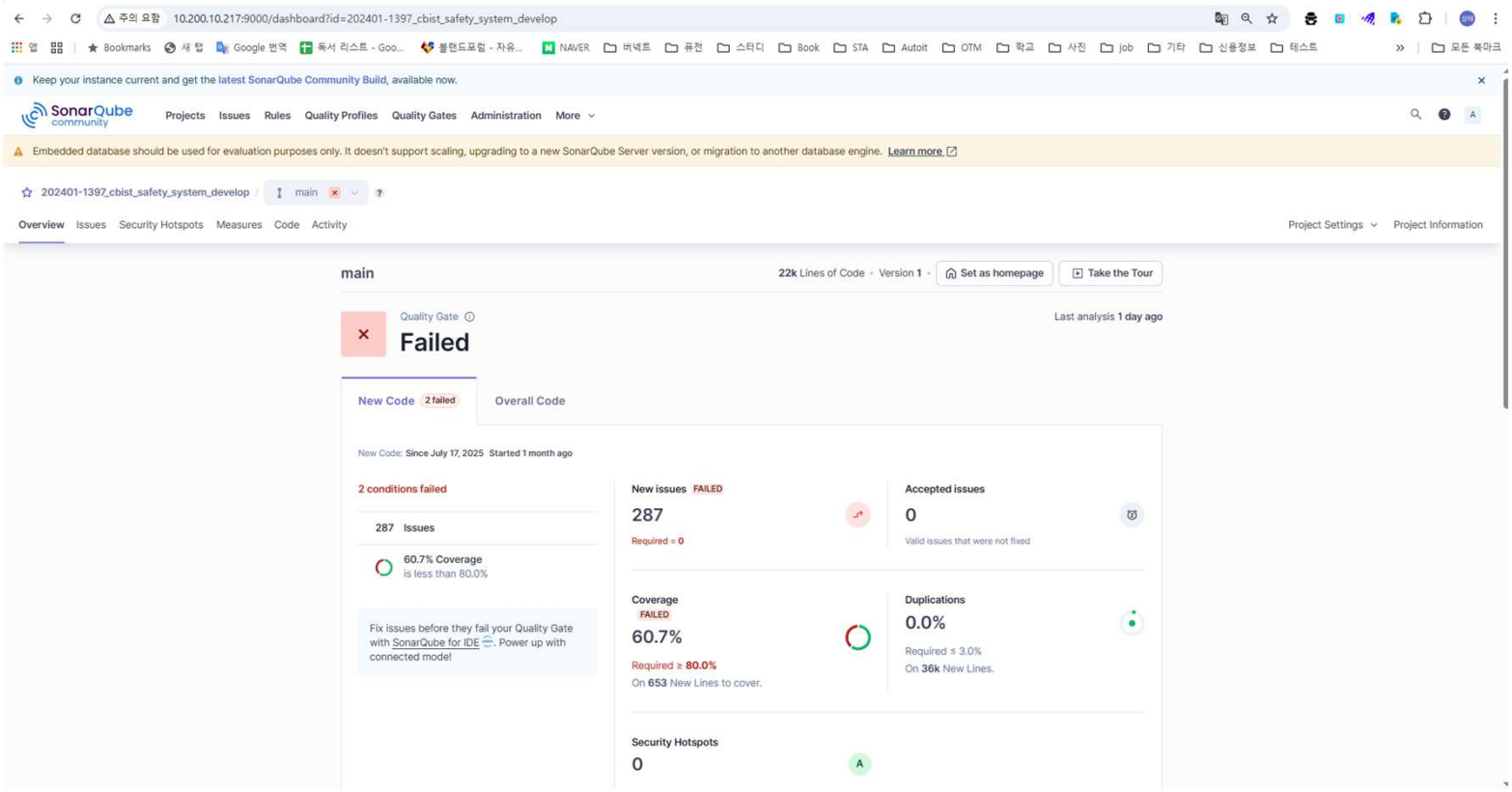
- **Role**
  - Build SonarQube environment & integrate with CI pipeline.
  - Configure language-specific rules and Quality Gate settings.
  - Visualize and manage coverage, issues, and code duplication.



VIRNECT

2. SonarQube Static Analysis Status – Backend (Java)

- **Language / Environment**
  - Java (Spring Boot)
  - Gradle + JUnit + JaCoCo
- **Analysis Metrics**
  - Coverage: 60.7% (Target ≥ 80%)
  - New issues: 287
  - Duplication: 0.0%
  - Quality Gate: ✗ Failed (2 conditions failed)
  - Last analysis: 1 day ago
- **Key Points**
  - Coverage collection enabled through JaCoCo integration.
  - Coverage has been collected but does not meet the target.
  - A large number of new issues are the main reason for Quality Gate failure.
- **Role**
  - Build SonarQube environment & integrate with CI pipeline.
  - Configure quality rules and Quality Gate settings per language.
  - Visualize and manage coverage, issues, and code duplication.



VIRNECT

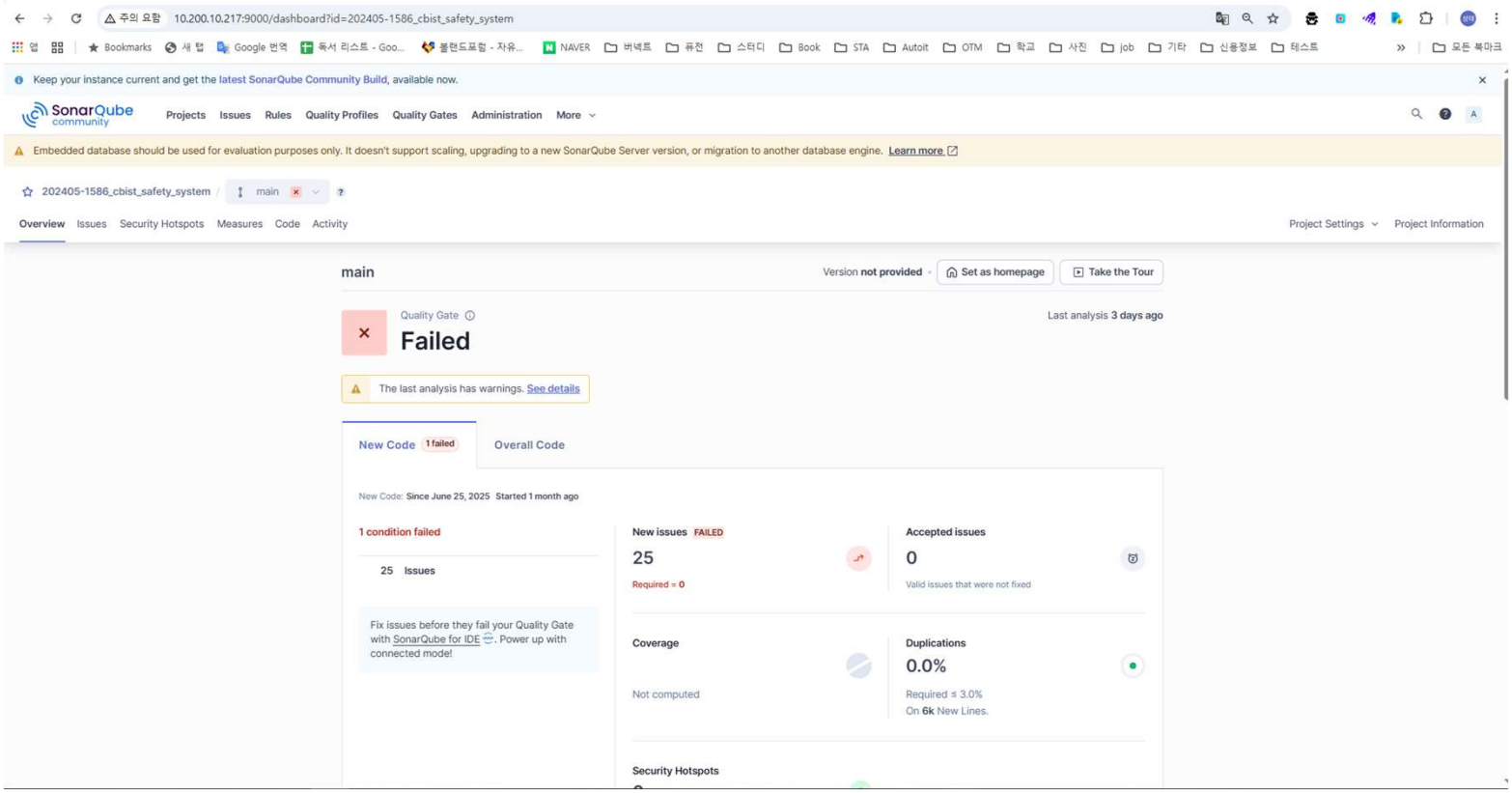
3. SonarQube Static Analysis Status – Unity (C#)

- **Language / Environment**
  - C# (Unity WebGL)

- **Analysis Metrics**
  - Coverage: Not computed (Unable to collect)
  - New issues: 25
  - Duplication: 0.0%
  - Quality Gate: X Failed (1 condition failed)
  - Last analysis: 3 days ago

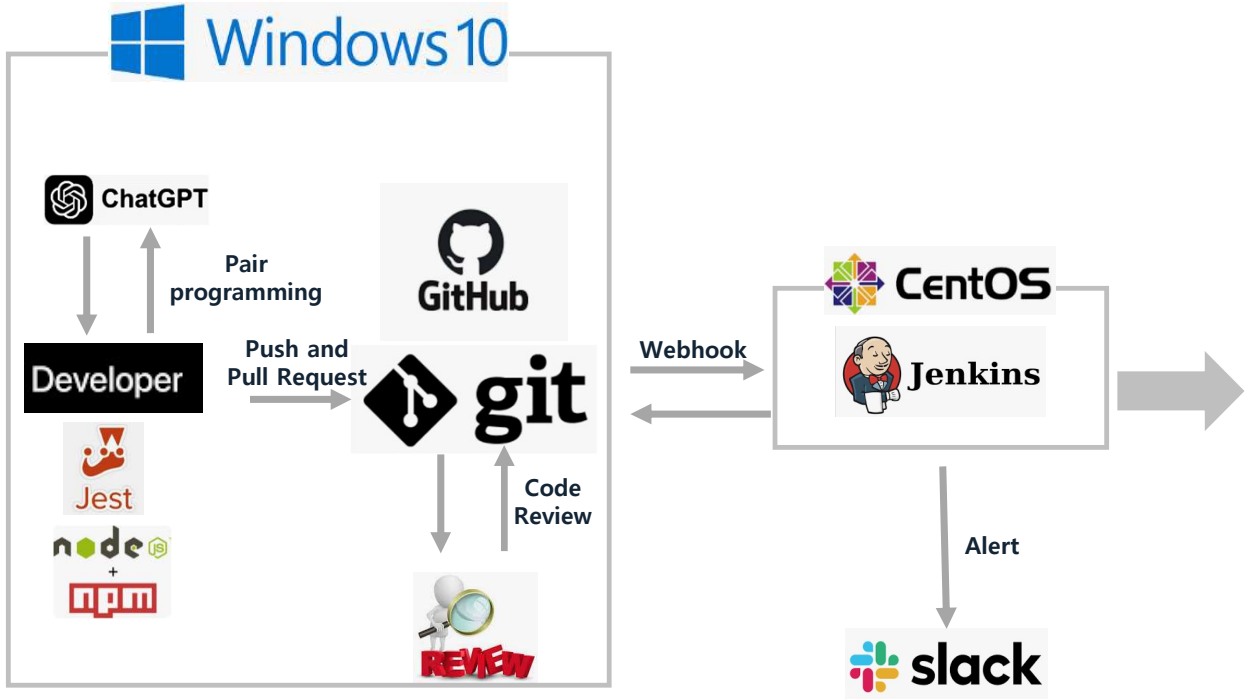
- **Key Points**
  - Unity project also managed under SonarQube.
  - Environment makes coverage collection difficult (Test framework not integrated).
  - 25 new issues caused Quality Gate failure.

- **Role**
  - Build SonarQube environment & integrate with CI pipeline.
  - Configure quality rules and Quality Gate settings per language.
  - Visualize and manage coverage, issues, and code duplication.

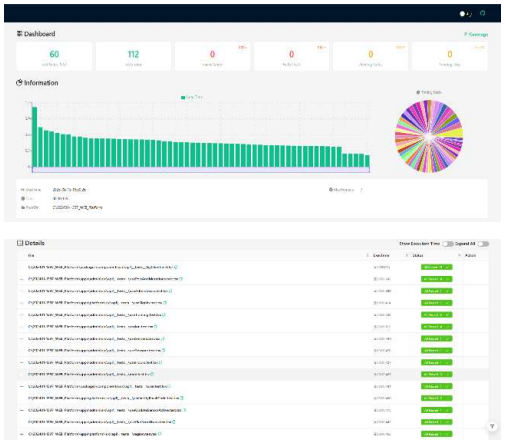


I. Project Overview

4. Unit Test Automation Pipeline – Frontend (TypeScript)



<GitHub → Jenkins → Jest → Slack>

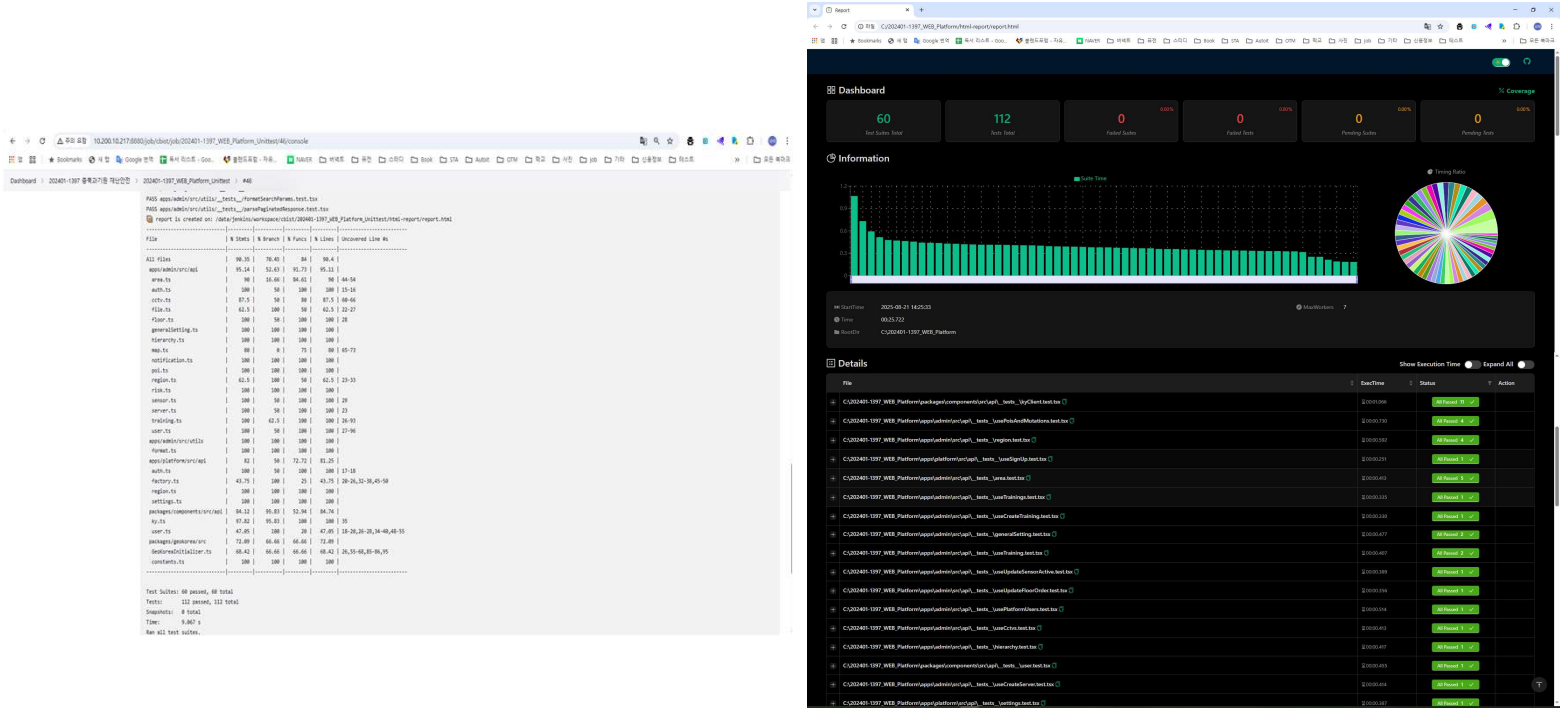


<Jest Unit test Reports>

<Jest Unit test Coverage>

5. Frontend (TypeScript) – Test Execution Results & Coverage

- **QA Environment**
  - Node.js / pnpm-based Frontend Monorepo environment
  - CI Environment: Automated tests executed via Jenkins
- **Technologies Used**
  - TypeScript-based module-level unit testing
  - React Query / Hooks / API unit-level validation
- **Tools Used**
  - Jest (Test framework)
  - Jenkins (Test automation and execution history management)
  - HTML Report (Visualization of execution time and duration by module)
- **Achievements**
  - Test Suites: 60 / Tests: 112 → 100% passed (All tests passed)
  - Visualized execution time and per-module duration of tests
  - Automatically reflected results in Jenkins pipeline → Strengthened regression test reliability

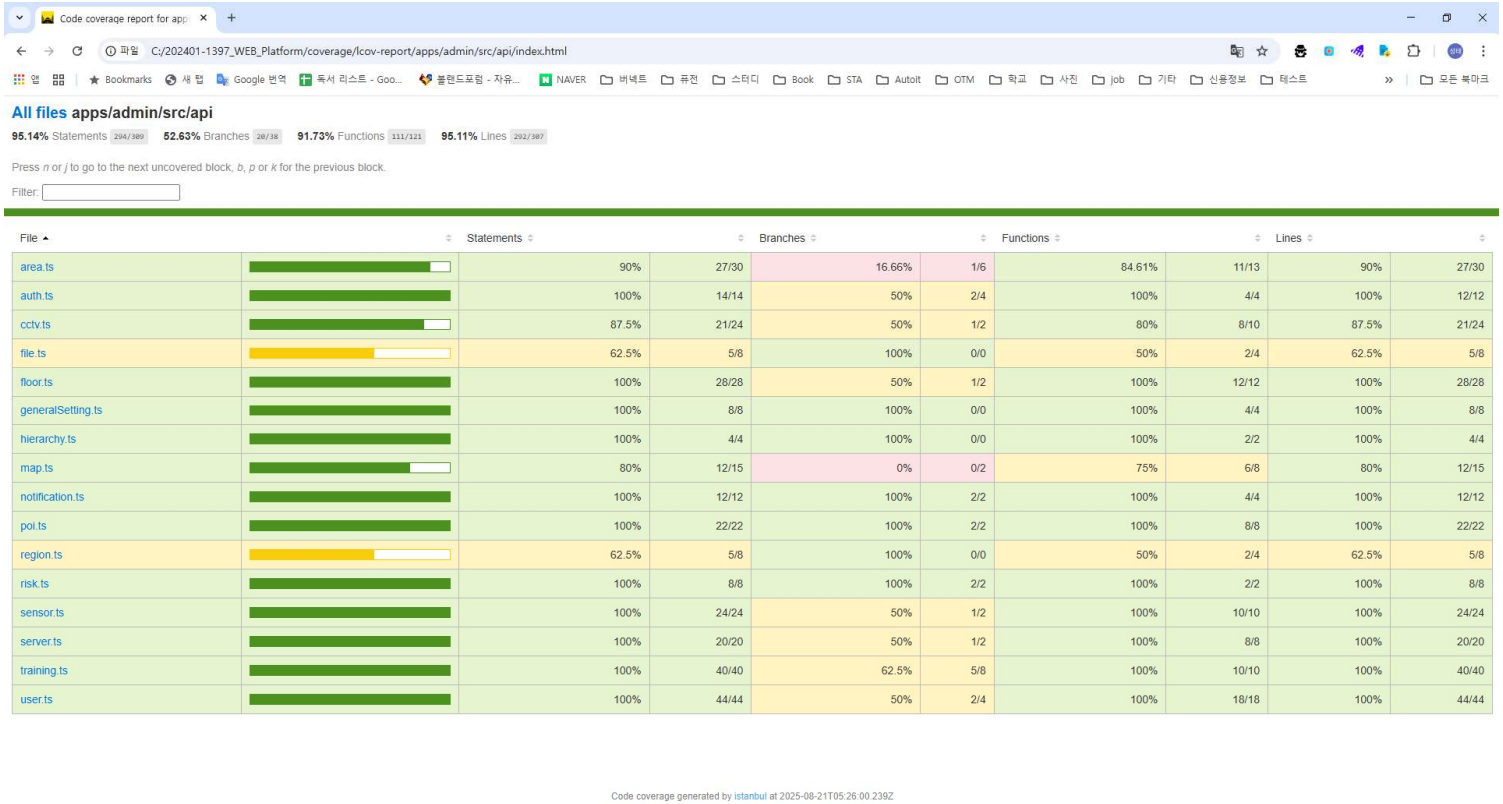


<Jenkins Log · Jest HTML Report · Coverage>

VIRNECT

6. Run the Docker Compose file

- **QA Environment**
  - Unit tests applied per business domain based on TypeScript/React.
- **Technologies Used**
  - Measure Statement, Branch, Function, and Line Coverage per feature.
- **Tools Used**
  - Jest + Istanbul (Coverage collection)
  - HTML Coverage Report (Visualize coverage by file)
- **Achievements**
  - Achieved 96%+ average Statement coverage overall.
  - Achieved 100% Line coverage for most critical business files.
  - Identified low Branch coverage (conditional logic) areas.
  - Able to visually track per-feature coverage differences for risk management.
- **Role**
  - Standardized frontend test code writing practices.
  - Built Jenkins Job & Slack notification automation pipeline.



Code coverage report for app

C:/202401-1397_WEB_Platform/coverage/lcov-report/apps/admin/src/api/index.html

All files apps/admin/src/api

95.14% Statements 294/309 52.63% Branches 20/38 91.73% Functions 111/121 95.11% Lines 292/307

Press n or j to go to the next uncovered block, b, p or k for the previous block.

Filter:

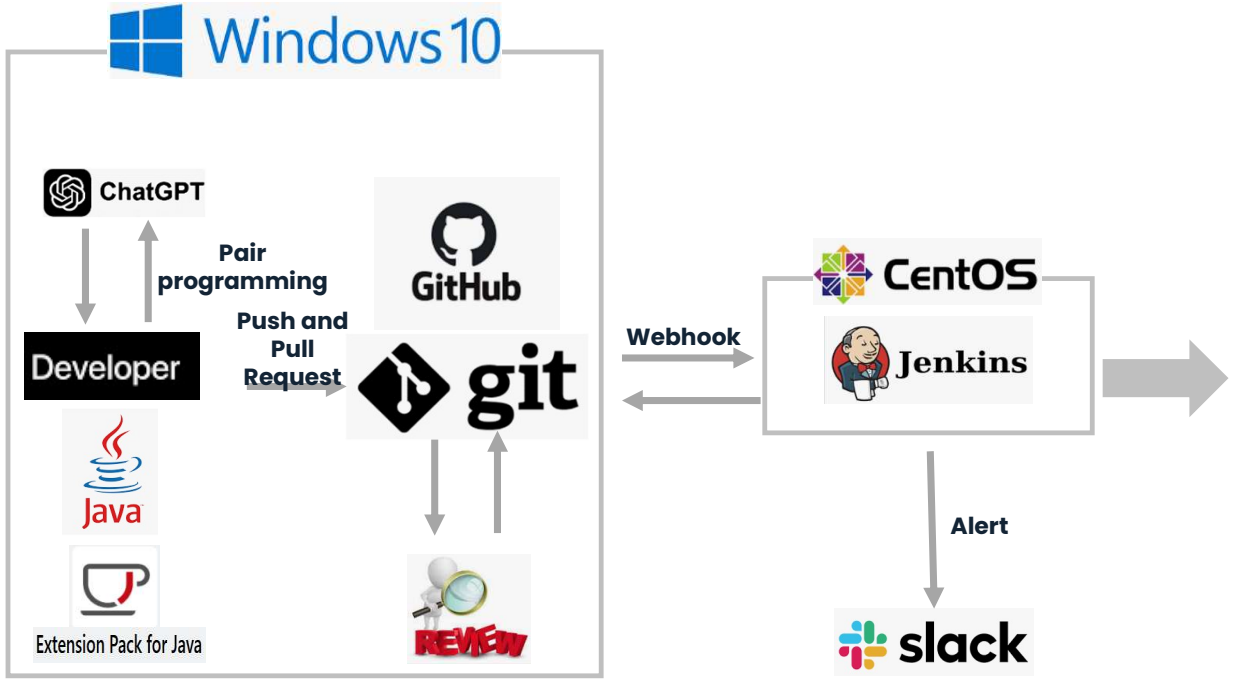
File	Statements		Branches		Functions		Lines	
area.ts	90%	27/30	16.66%	1/6	84.61%	11/13	90%	27/30
auth.ts	100%	14/14	50%	2/4	100%	4/4	100%	12/12
cctv.ts	87.5%	21/24	50%	1/2	80%	8/10	87.5%	21/24
file.ts	62.5%	5/8	100%	0/0	50%	2/4	62.5%	5/8
floor.ts	100%	28/28	50%	1/2	100%	12/12	100%	28/28
generalSetting.ts	100%	8/8	100%	0/0	100%	4/4	100%	8/8
hierarchy.ts	100%	4/4	100%	0/0	100%	2/2	100%	4/4
map.ts	80%	12/15	0%	0/2	75%	6/8	80%	12/15
notification.ts	100%	12/12	100%	2/2	100%	4/4	100%	12/12
pol.ts	100%	22/22	100%	2/2	100%	8/8	100%	22/22
region.ts	62.5%	5/8	100%	0/0	50%	2/4	62.5%	5/8
risk.ts	100%	8/8	100%	2/2	100%	2/2	100%	8/8
sensor.ts	100%	24/24	50%	1/2	100%	10/10	100%	24/24
server.ts	100%	20/20	50%	1/2	100%	8/8	100%	20/20
training.ts	100%	40/40	62.5%	5/8	100%	10/10	100%	40/40
user.ts	100%	44/44	50%	2/4	100%	18/18	100%	44/44

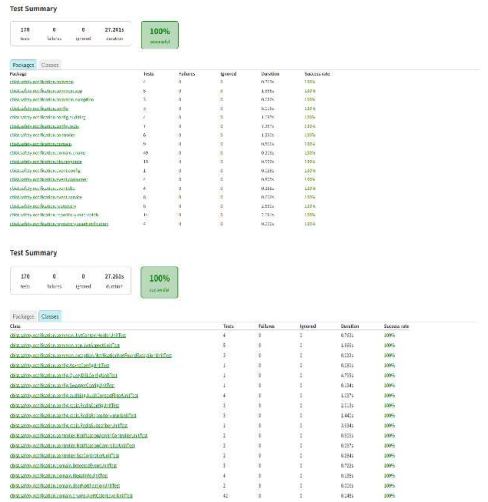
Code coverage generated by istanbul at 2025-08-21T05:26:00.239Z

<Jenkins 로그 · Jest HTML Report · Coverage>

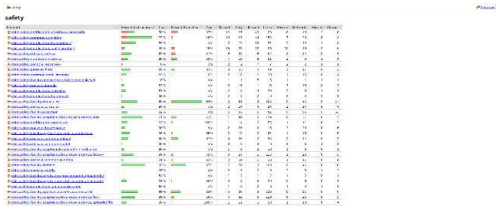
VIRNECT

7. Unit Test Automation Pipeline – Backend (Java)





‹JUnit test Reports›

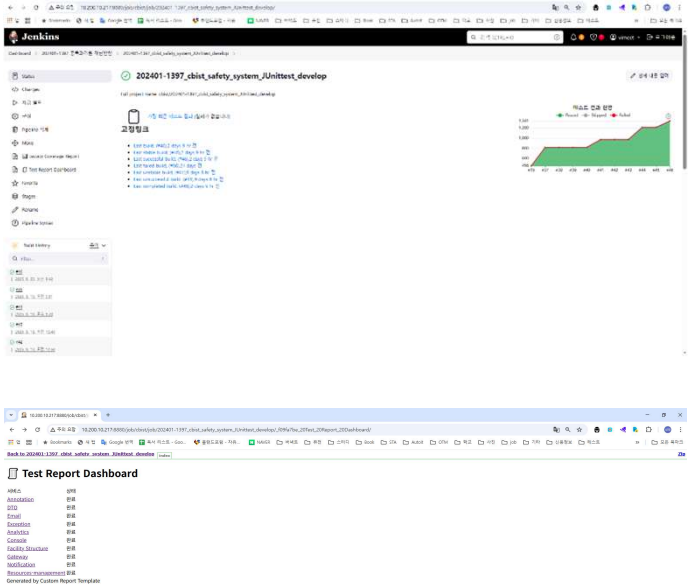


‹JUnit test Coverage›

GitHub → Jenkins → JUnit → Slack

8. Backend (Java) – Test Execution Results & Coverage

- **QA Environment**
  - Spring Boot-based modular architecture
  - Automated test execution in a Jenkins-based CI environment
- **Technologies Used**
  - Java + JUnit-based unit testing
  - Verification at Domain / Application / Adapter layer levels
- **Tools Used**
  - JUnit Report Dashboard
  - Jenkins Test Trend (Graph & History)
  - Gradle-based automated report generation
- **Achievements**
  - Recent builds: 1,341 tests executed, 100% success rate
  - Established a module-level test execution status dashboard
  - Built a foundation for regression test automation and continuous expansion

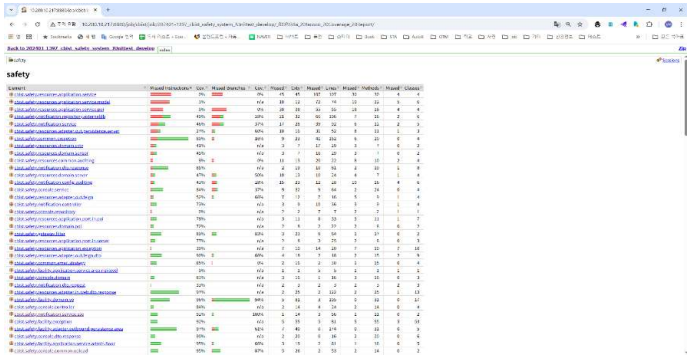


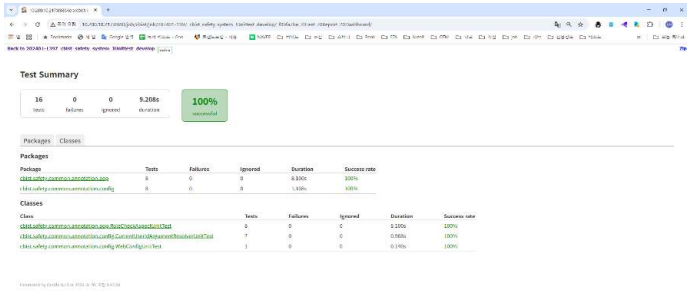
‹Jenkins Trend · JUnit Report · JaCoCo Coverage›

VIRNECT

9. Verify that SonarQube is installed in Docker

- **QA Environment**
  - Spring Boot-based modular architecture
  - Automated test execution in a Jenkins-based CI environment
- **Technologies Used**
  - Measure Statement, Branch, Line Coverage using JaCoCo
  - Detailed coverage tracking per module and detection of uncovered lines
- **Tools Used**
  - JaCoCo Coverage Report
  - Jenkins Coverage Integrated View
  - Gradle automated report generation scripts
- **Achievements**
  - Achieved 60%+ average Statement coverage overall
  - Achieved 90%+ coverage for core business modules
  - Identified coverage blind spots for future improvement
- **Role**
  - Designed and implemented backend service unit test cases
  - Configured JaCoCo-based coverage collection and automated reports



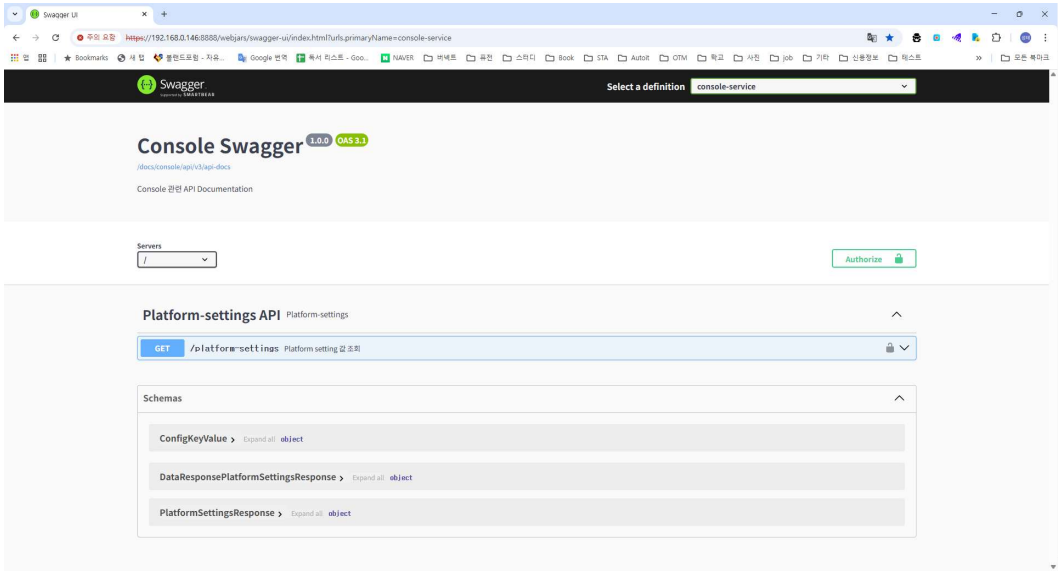


‹Jenkins Trend · JUnit Report · JaCoCo Coverage›

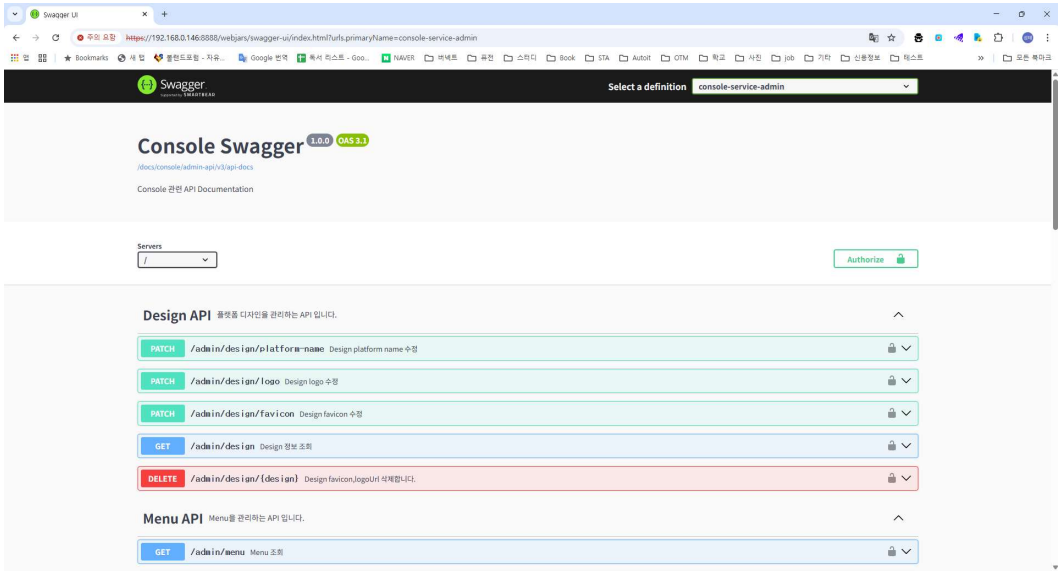
VIRNECT

10. API(Application Programming Interface) Testing

- RESTful API integration testing



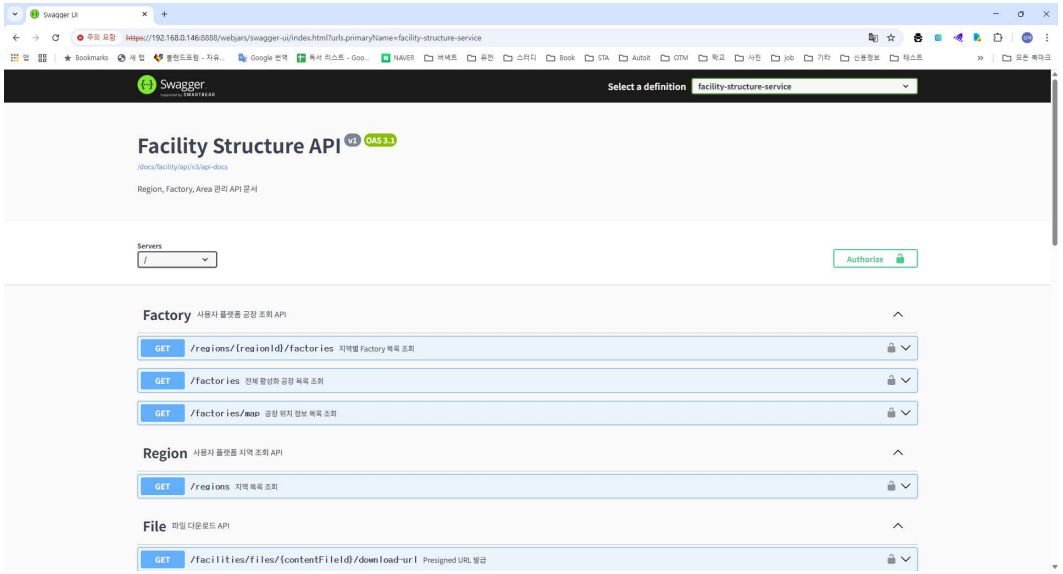
- RESTful API integration testing



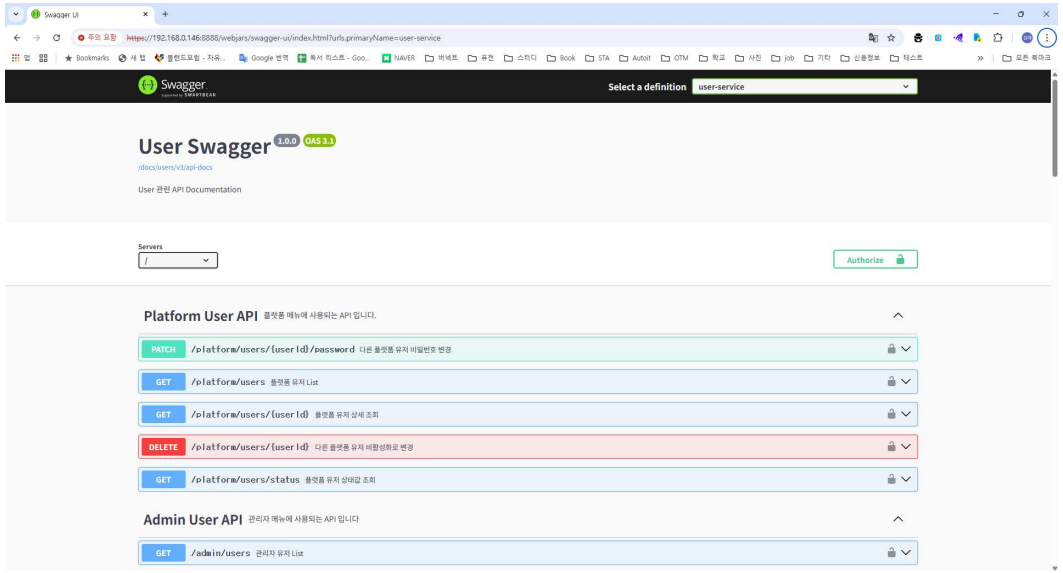
VIRNECT

10. API(Application Programming Interface) Testing

- RESTful API integration testing

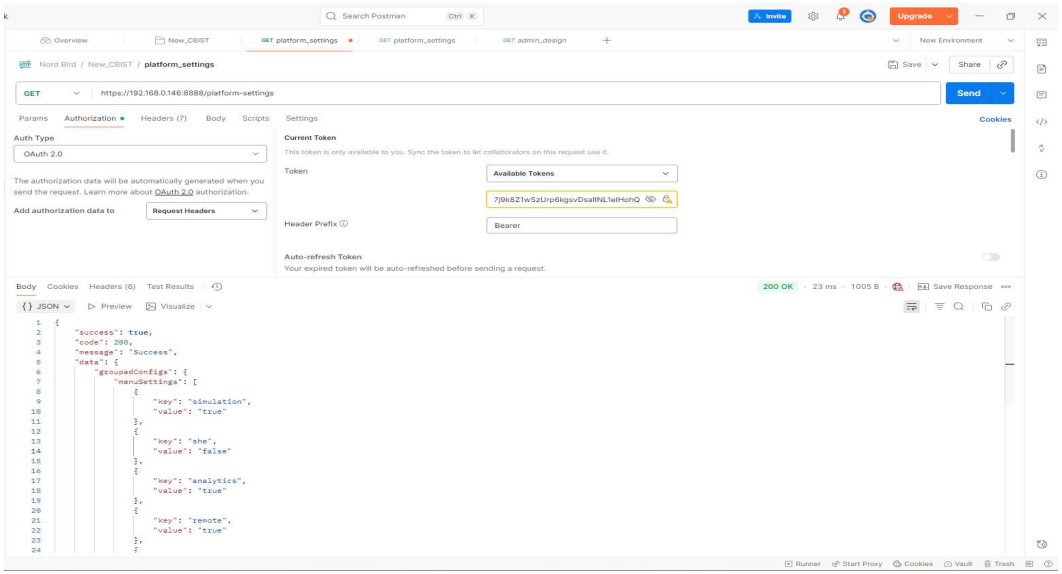


- RESTful API integration testing

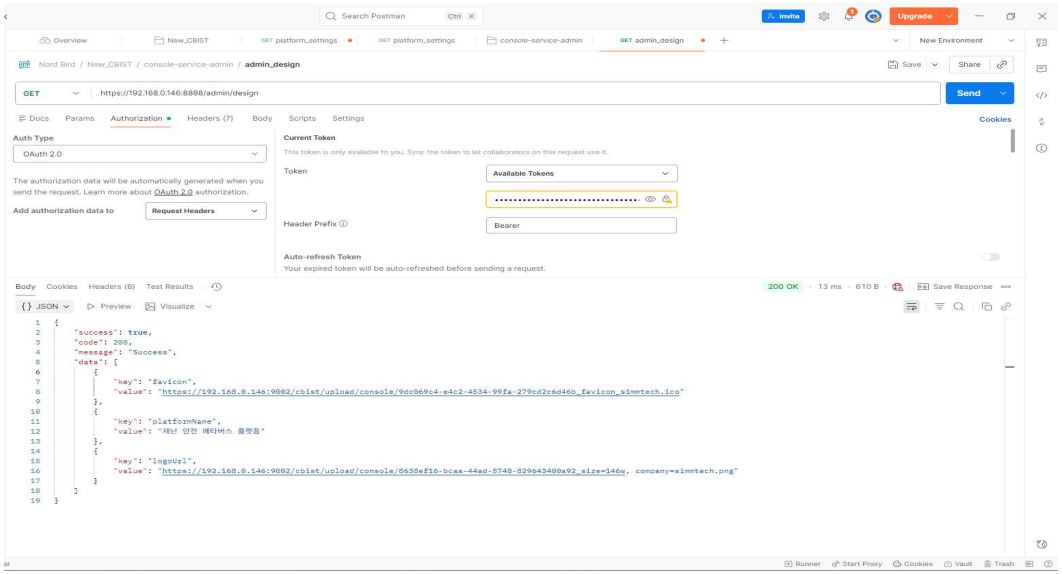


10. API(Application Programming Interface) Testing

- RESTful API integration testing

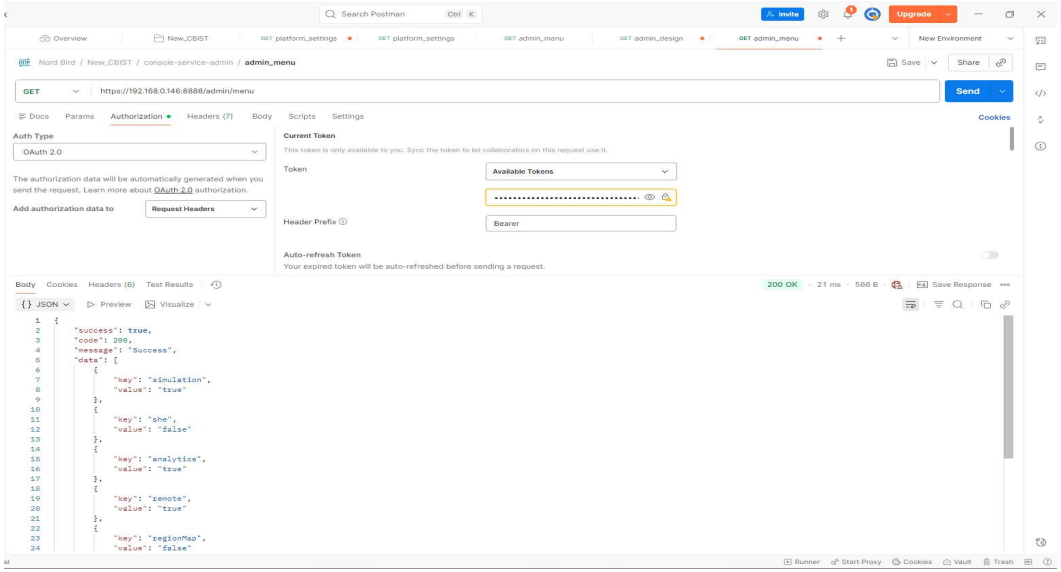


- RESTful API integration testing

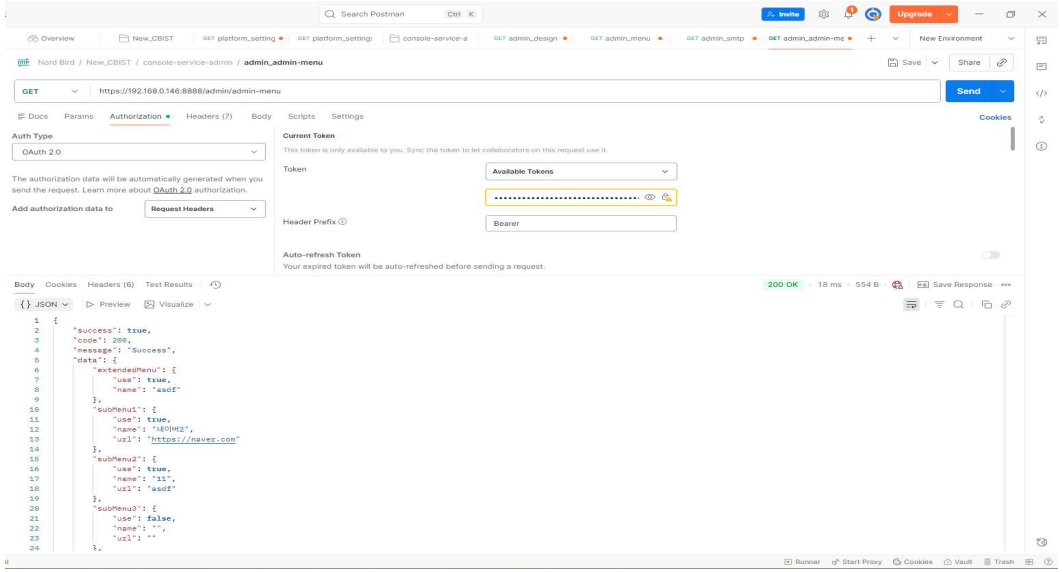


10. API(Application Programming Interface) Testing

- RESTful API integration testing



- RESTful API integration testing



VIRNECT

11. Performance Testing (JMeter)

- **Test Environment**
  - RESTful API integration testing
  - Platform & Admin APIs
- **Test Scenarios**

Platform APIs
- Platform settings, Region, Factory, Facility hierarchy
- Notification, Analytics, Resource (POI, Event) APIs

Admin APIs
- Factory / Floor / Area CRUD
- Facility hierarchy & evacuation routes
- File upload/download (Presigned URL)
- IoT & POI management APIs

- **Key Results**

API Coverage
- Total APIs tested: 50+
- Platform & Admin roles verified

Validation Results
- HTTP Status: 100% expected responses
- Authentication & authorization: No issues
- CRUD & hierarchy flow: All verified

- **Role**
  - Designed and executed API integration tests
  - Verified request/response and data consistency
  - Supported release readiness decision

API Integration Test Summary Table

API Test Metrics

Metric	Platform APIs	Admin APIs	Analysis
API Coverage	Core user-facing APIs	Management & configuration APIs	Full functional coverage
Authentication	OAuth 2.0 (Bearer)	OAuth 2.0 (Bearer)	Stable authentication handling
HTTP Status Accuracy	100% expected	100% expected	No unexpected responses
Error Rate	0%	0%	Highly stable
CRUD Validation	Partial (read 중심)	Full CRUD verified	Data integrity confirmed
Integration Flow	Region → Factory → Floor → Area	Same hierarchy verified	Consistent hierarchy

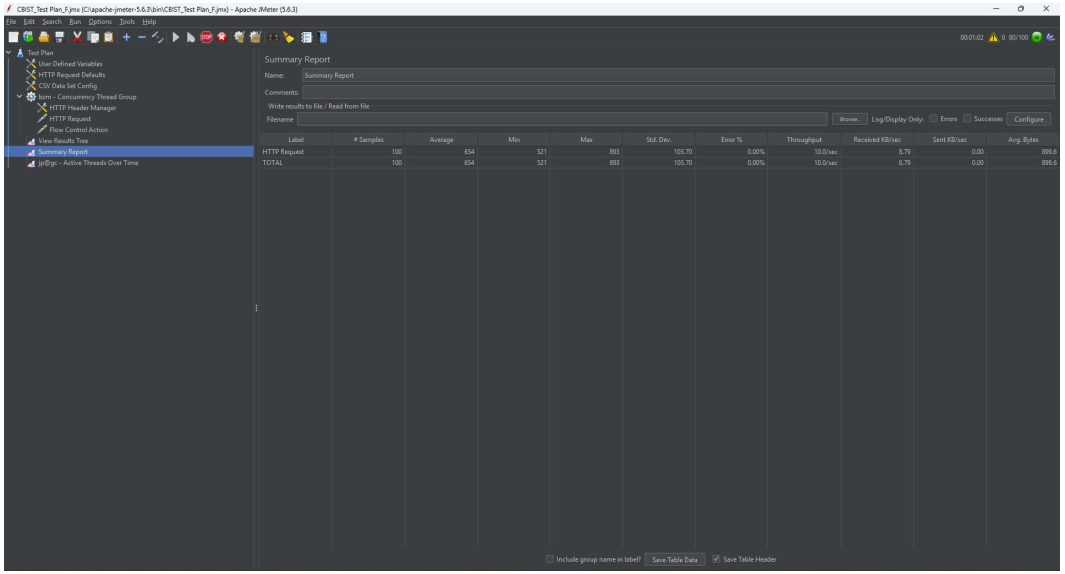
Summary Statement

✓ All platform and admin APIs operate stably and consistently

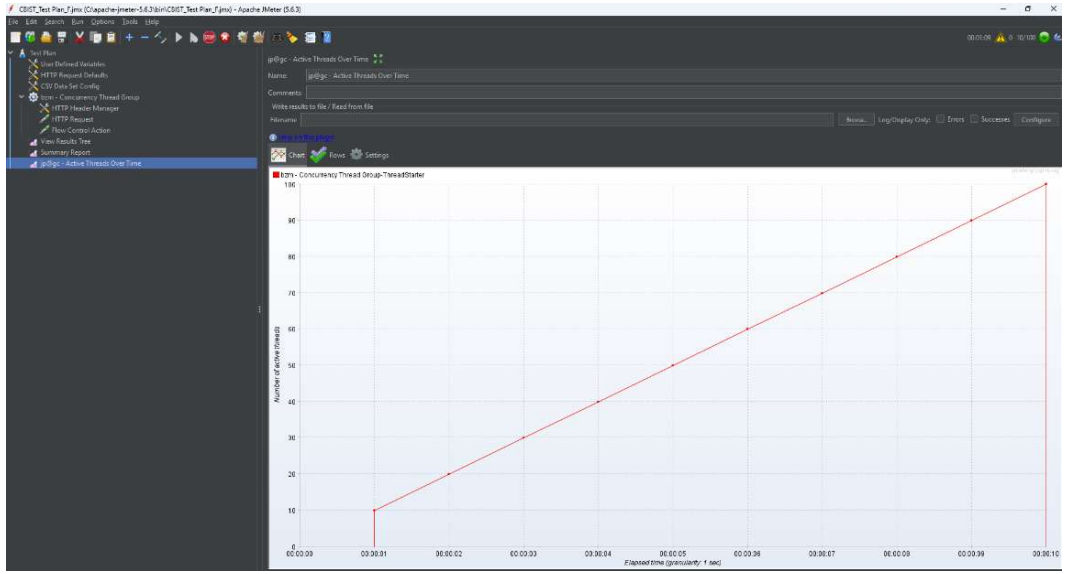
✓ API integration requirements are fully satisfied

12. Performance Testing

- Summary Report of 100 concurrent users

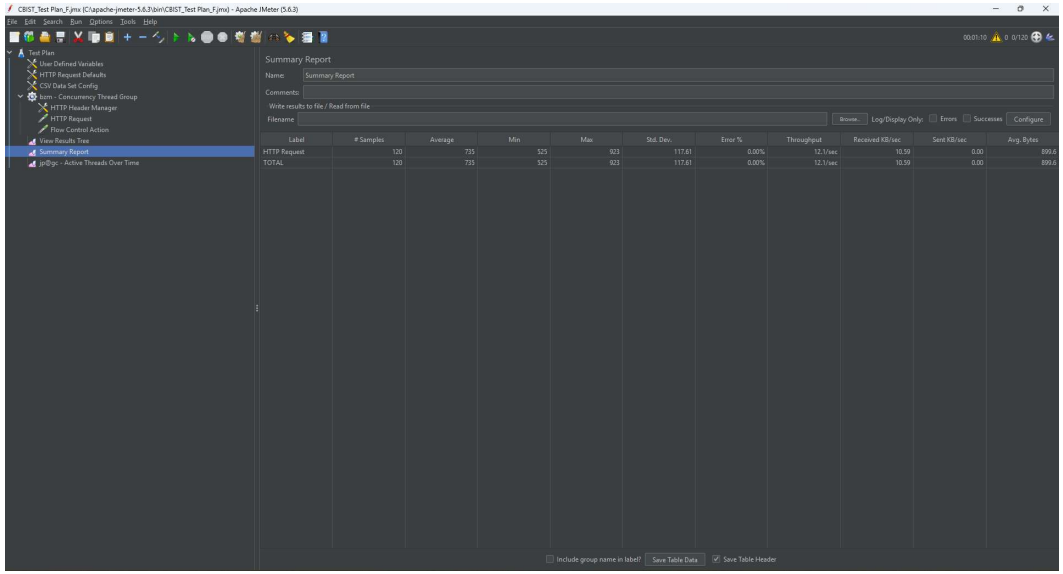


- Graph of 100 concurrent users

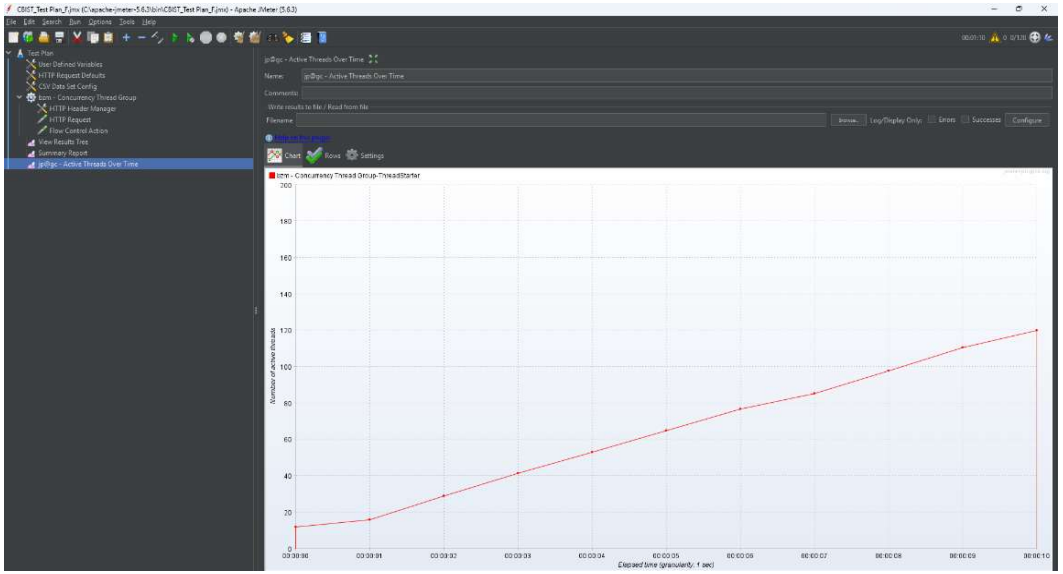


12. Performance Testing

- Summary Report of 120 concurrent users



- Graph of 120 concurrent users



VIRNECT

13. Performance Testing (JMeter)

- **Test Environment**
  - JMeter-based performance testing environment
  - Backend API load and concurrency testing
  - Step load pattern with gradual user increase

- **Test Scenarios**

100 Virtual Users
- Increased by 10 users step-by-step and sustained

120 Virtual Users
- Increased by 12 users step-by-step and sustained

- **Key Results**

100 Users
- Avg Response Time: 654 ms
- Error Rate: 0%, Success Rate: 100%

120 Users
- Avg Response Time: 735 ms
- Error Rate: 0%, Success Rate: 100%

- **Role**
  - Designed and executed JMeter performance tests
  - Analyzed results and supported release readiness assessment

Metric	100 Users	120 Users	Analysis
Average Response Time	654 ms	735 ms	Approx. **+12% increase**, acceptable
Maximum Response Time	893 ms	923 ms	No significant difference
Error Rate	0%	0%	Highly stable
Success Rate	100%	100%	Successful
Throughput	10.0 / sec	12.1 / sec	Linear increase

Key Strengths
Excellent response time (under 1 second)
Zero errors → high service reliability
Linear scaling with increased load → good scalability

Summary Statement

✓ The login API remains stable and operates reliably **up to 120 concurrent users without performance degradation**.

2. Project Results

- Successfully deployed and configured SonarQube using Docker and Docker Compose on Linux, establishing a stable environment for continuous code quality analysis.
- Integrated SonarQube with Jenkins CI/CD pipelines, enabling automated static analysis and quality gate enforcement for frontend and backend services.
- Performed static analysis of automated test code using SonarScanner CLI, identifying code smells, bugs, and security vulnerabilities early in the development cycle.
- Built centralized SonarQube dashboards to visualize key quality metrics such as code duplication, coverage trends, and security hotspots, with real-time notifications via Slack.
- Designed and executed JMeter-based performance tests to validate backend API stability under concurrent user load.
- Conducted step-load performance scenarios with 100 and 120 virtual users, confirming that average response times remained under 1 second with 0% error rate and 100% success rate.
- Observed linear throughput growth as concurrency increased, demonstrating stable scalability and supporting release readiness.
- Achieved measurable improvements in test maintainability, system stability, and regression reliability, significantly reducing technical and performance-related risks.