VIRNECT

QA Automation, Code Quality, CI/CD, SonarQube, Test Coverage

Why QA Automation & Code Quality?

VIRNECT QA Team

Sungtae Kim

I. Intro -

Self-introduction

- Software Test Management & Process Improvement
- Test Design · Execution · Reporting
- Test Automation Design & Implementation

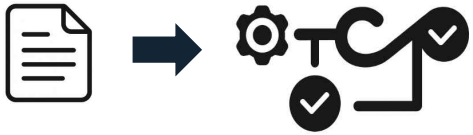
I. Project Overview

Background

- Increasing project complexity
- Growing importance of quality management

Problem

- Lack of objective metrics for test & quality levels
- Missing systematic static analysis and unit testing
- Code duplication and issue accumulation → reduced stability



Before
Code Quality
Unknown

Automated
QA Pipeline

- Establish standardized and automated code quality management
- Ensure quality based on static analysis and unit testing
- Apply quality gates at PR (Pull Request) / release stages
- Build a stable service deployment environment

I. Project Overview

- Introduce test automation into CI/CD
- Establish an integrated SonarQube dashboard
- Ensure stable releases through test automation

- Static Analysis (SonarQube + Jenkins)
- Secure test coverage (Jest, JUnit, Istanbul, JaCoCo)
- Apply Quality Gates (block PR merges when unmet)
- Improve collaboration efficiency (Slack/GitHub integration)

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

II. Implementation Strategy

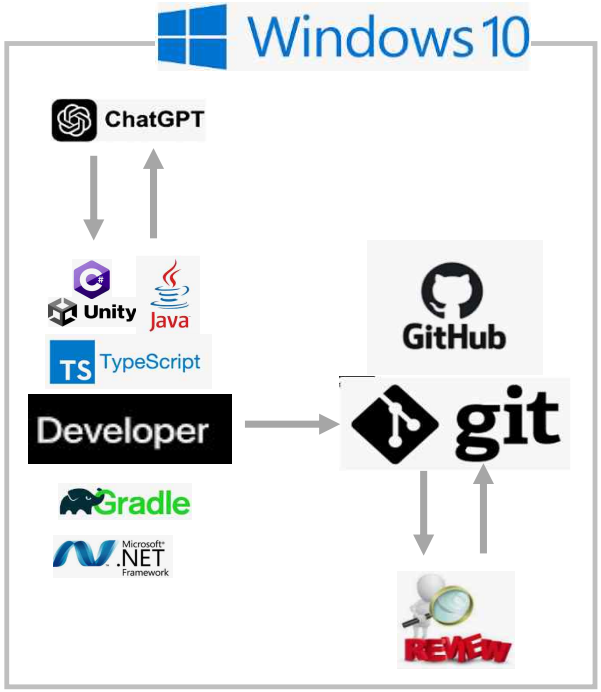
Static Analysis (SonarQube + Jenkins)

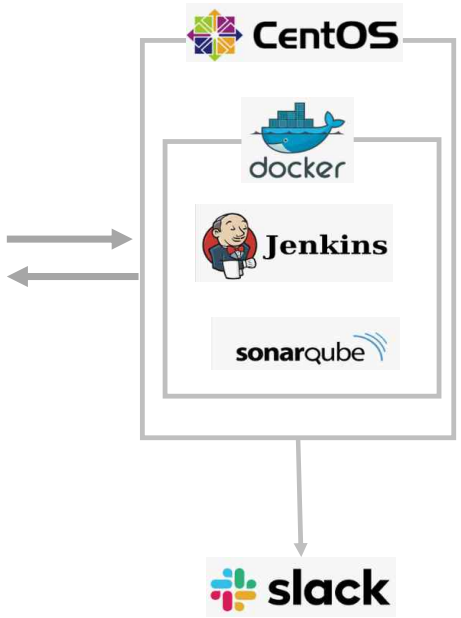
Static Analysis: Early detection of code smells and vulnerabilities with Jenkins and SonarQube

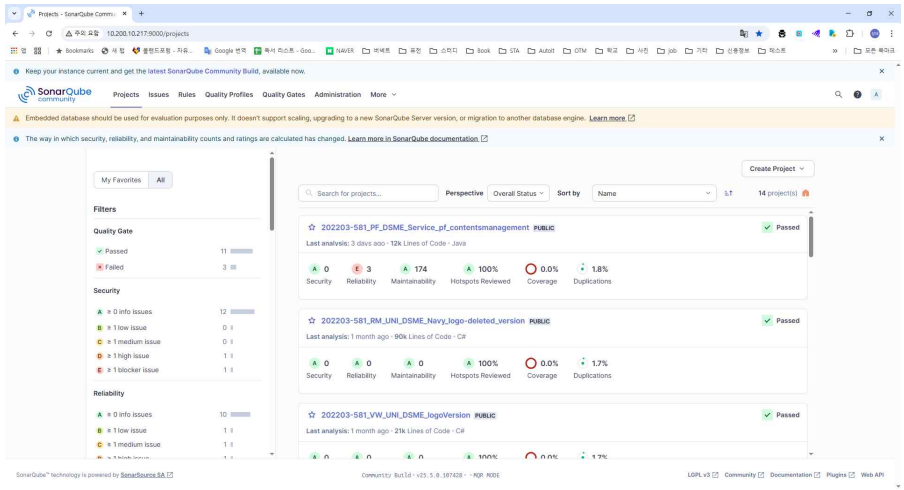
Test Coverage: Coverage measurement based on Jest, JUnit, and JaCoCo

Quality Gates: Block merges at the PR stage if standards are not met

Collaboration: Real-time quality alerts and enhanced reviews via Slack/GitHub







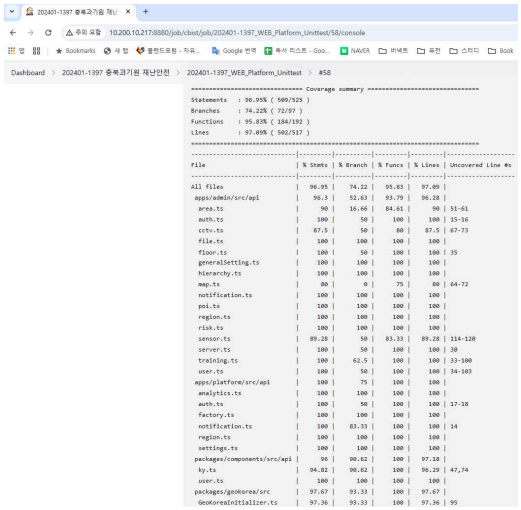
II. Implementation Strategy

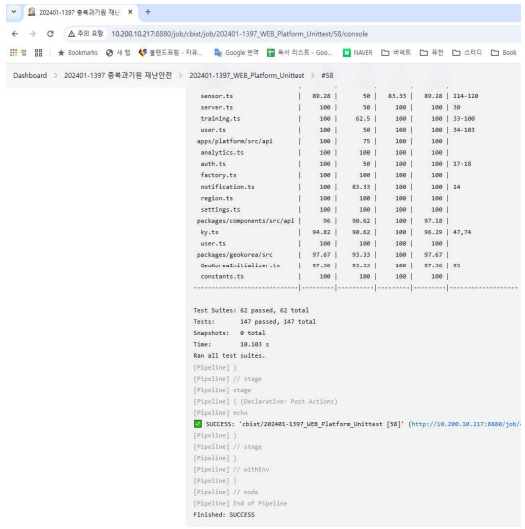
Ensuring test coverage (Jest, JUnit, Istanbul, JaCoCo)

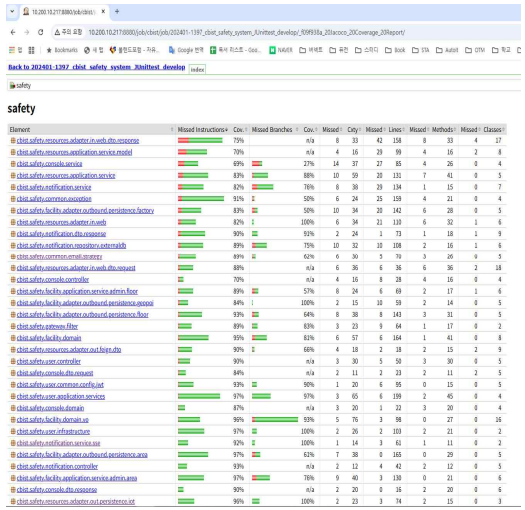
Automated unit test coverage measurement and reporting

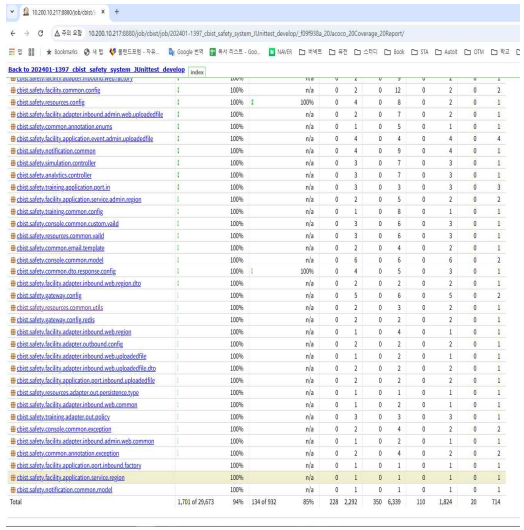
Coverage visualization via Jest (TS), JUnit + JaCoCo (Java), and Istanbul

Coverage target management by statement, branch, and line









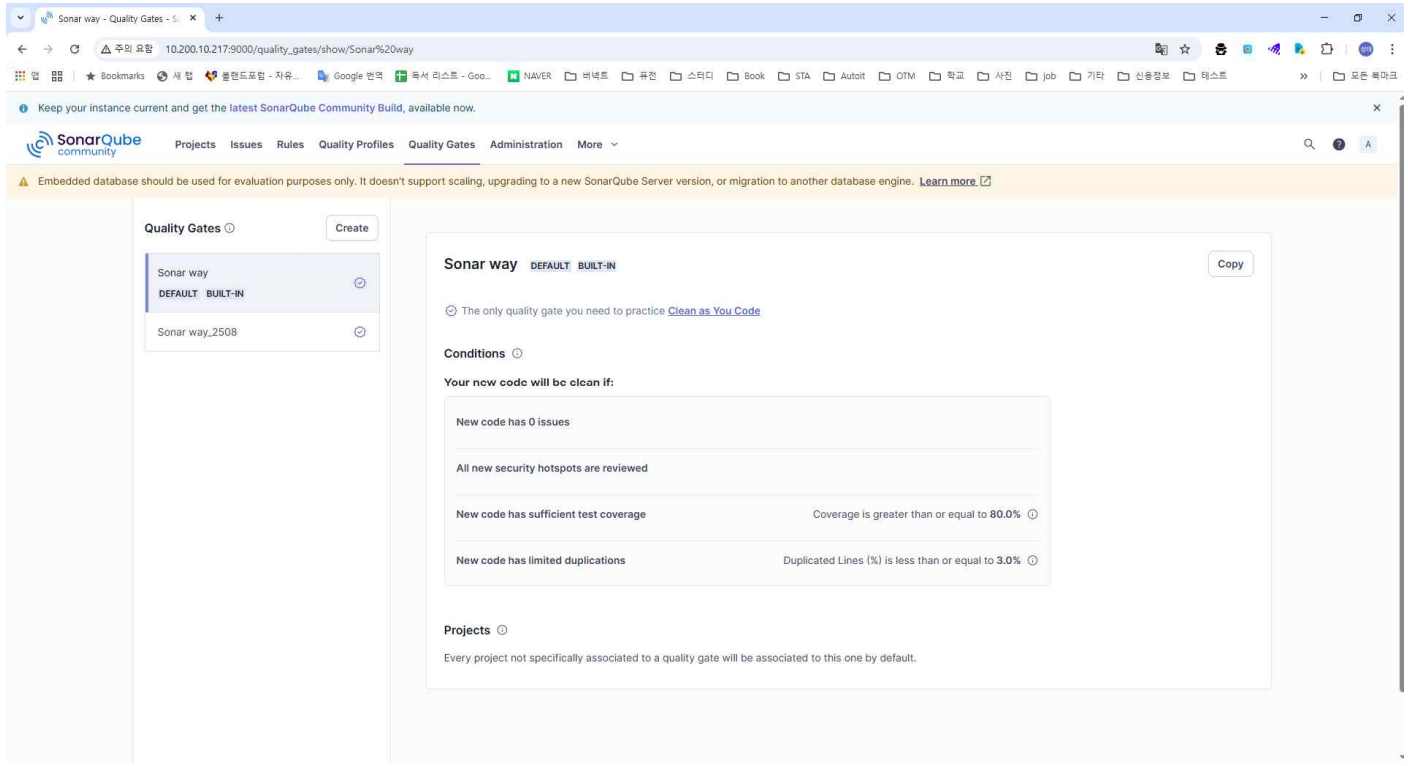
II. Implementation Strategy

Apply Quality Gate (PR Merge Block)

Automated quality gate checks at the Pull Request stage

Block merges if standards are not met → Prevent defects before they occur

Encourage compliance with quality standards during the development phase



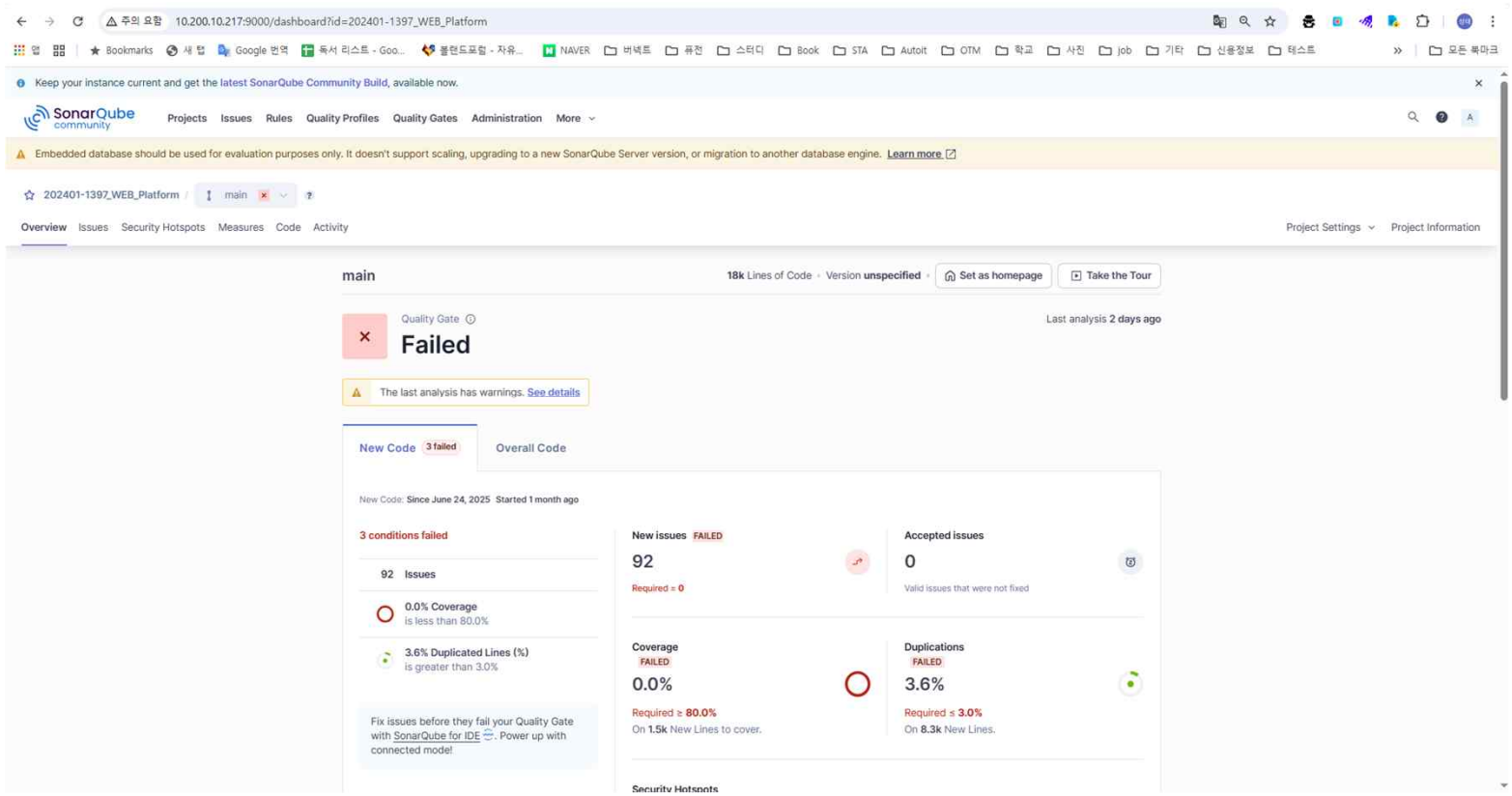
1. SonarQube Static Analysis Status – Frontend (TypeScript)

- **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.



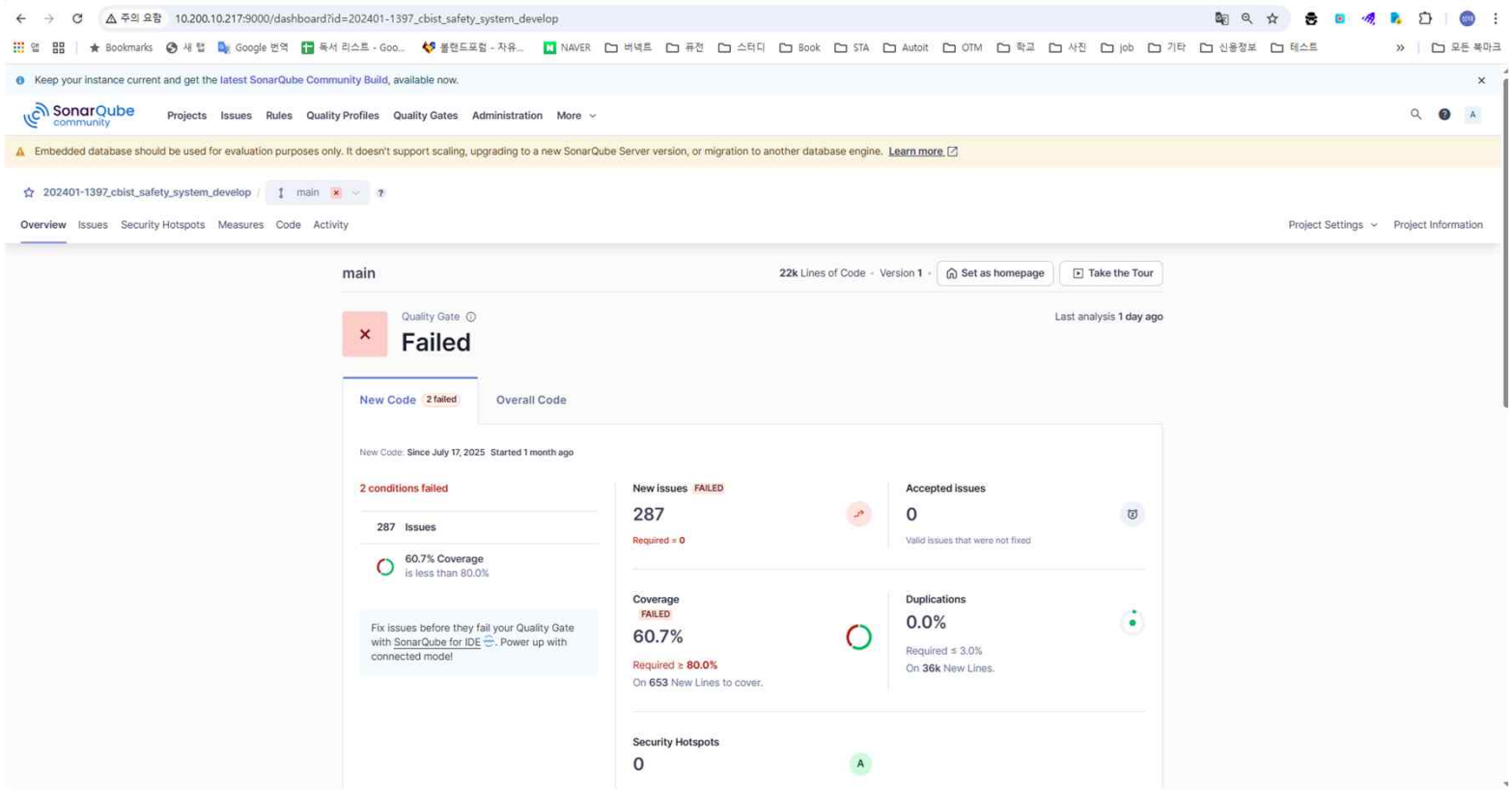
2. SonarQube Static Analysis Status – Backend (Java)

- Java (Spring Boot)
- Gradle + JUnit + JaCoCo

- Coverage: 60.7% (Target ≥ 80%)
- New issues: 287
- Duplication: 0.0%
- Quality Gate: ✗ Failed (2 conditions failed)
- Last analysis: 1 day ago

- 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.

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



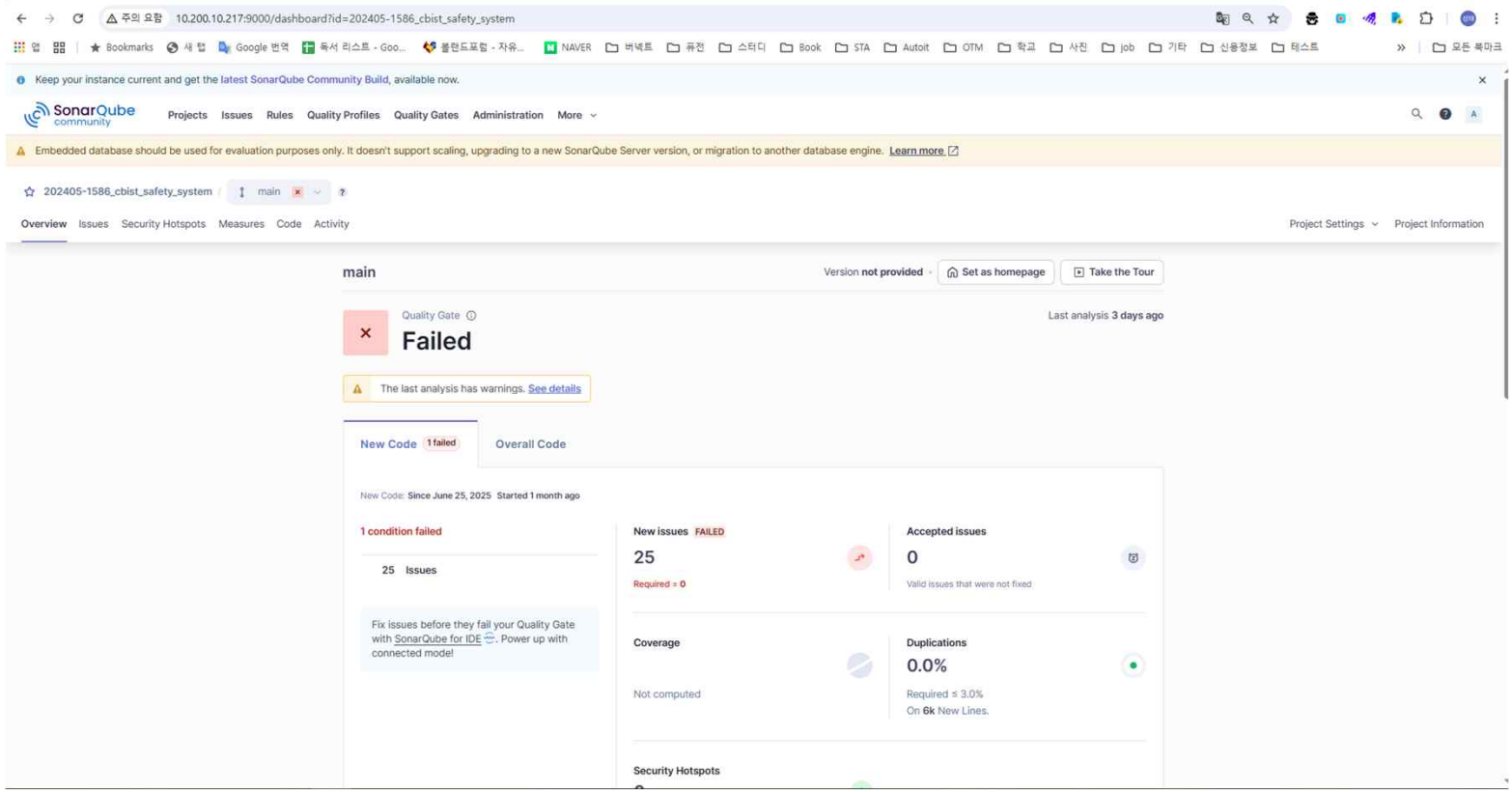
3. SonarQube Static Analysis Status – Unity (C#)

- C# (Unity WebGL)

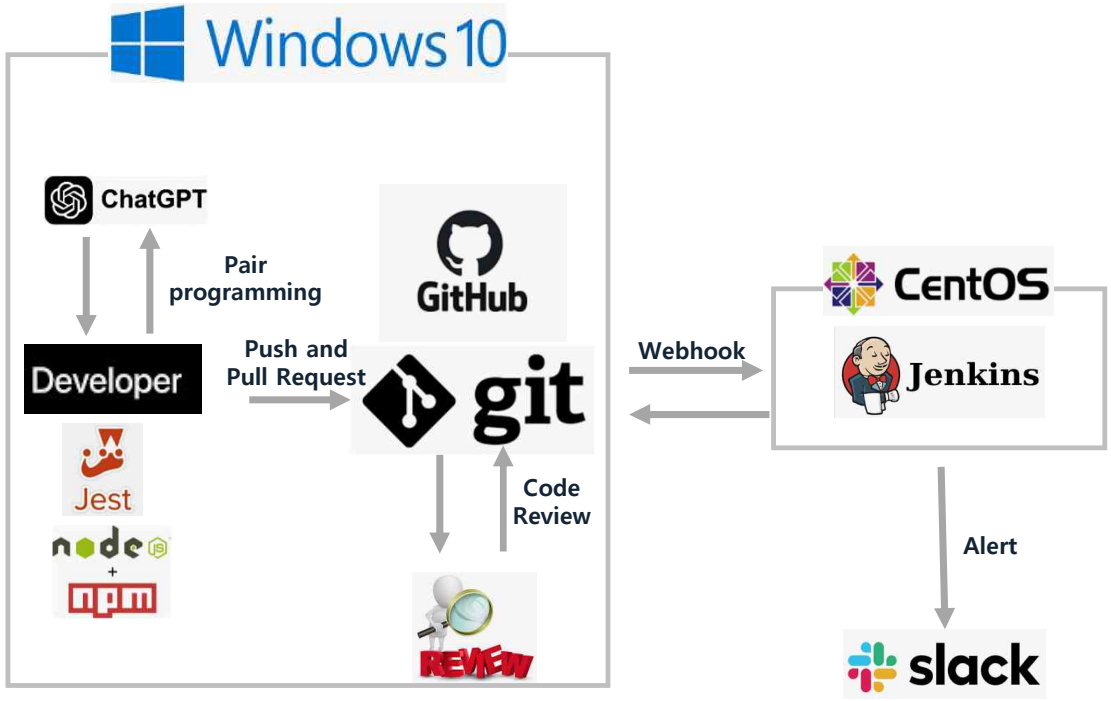
- Coverage: Not computed (Unable to collect)
- New issues: 25
- Duplication: 0.0%
- Quality Gate: X Failed (1 condition failed)
- Last analysis: 3 days ago

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

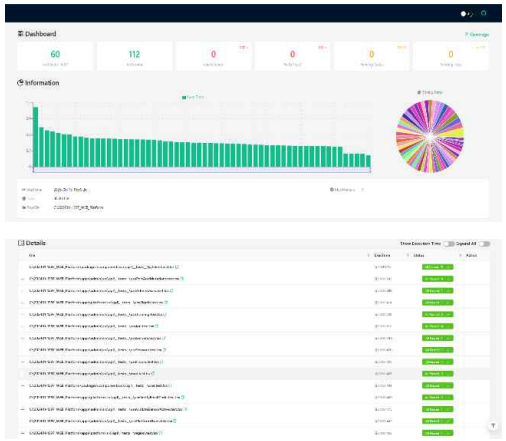
- Build SonarQube environment & integrate with CI pipeline.
- Configure quality rules and Quality Gate settings per language.
- Visualize and manage coverage, issues, and code duplication.



4. Unit Test Automation Pipeline – Frontend (TypeScript)



<GitHub → Jenkins → Jest → Slack>



<Jest Unit test Reports>

<Jest Unit test Coverage>

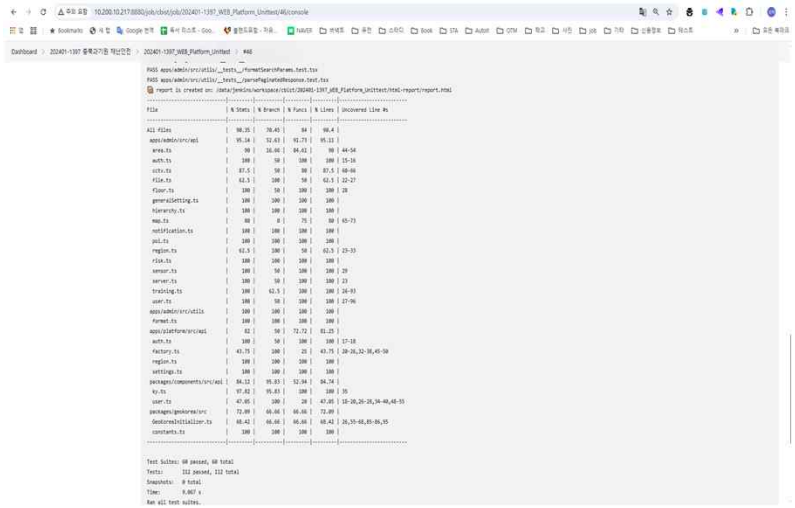
5. Frontend (TypeScript) – Test Execution Results & Coverage

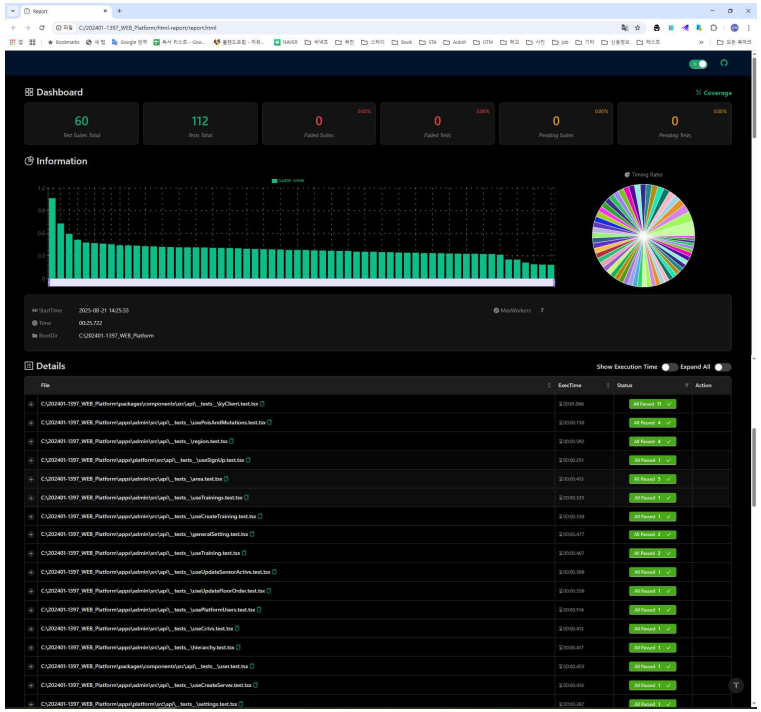
- Node.js / pnpm-based Frontend Monorepo environment
- CI Environment: Automated tests executed via Jenkins

- TypeScript-based module-level unit testing
- React Query / Hooks / API unit-level validation

- Jest (Test framework)
- Jenkins (Test automation and execution history management)
- HTML Report (Visualization of execution time and duration by module)

- 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





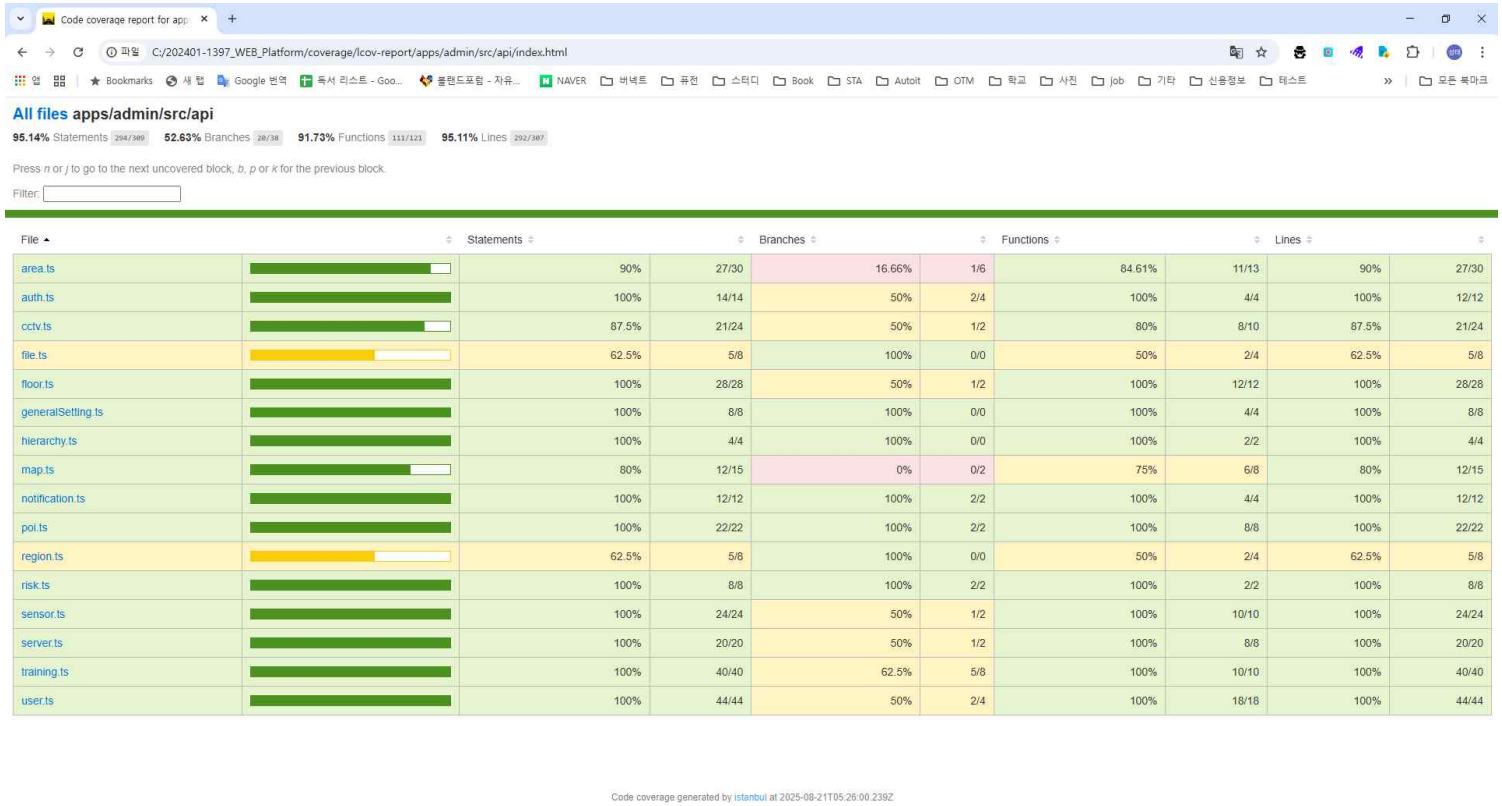
<Jenkins Log · Jest HTML Report · Coverage>

VIRNECT

6. Run the Docker Compose file

- Unit tests applied per business domain based on TypeScript/React.
- Measure Statement, Branch, Function, and Line Coverage per feature.
- Jest + Istanbul (Coverage collection)
- HTML Coverage Report (Visualize coverage by file)
- 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.

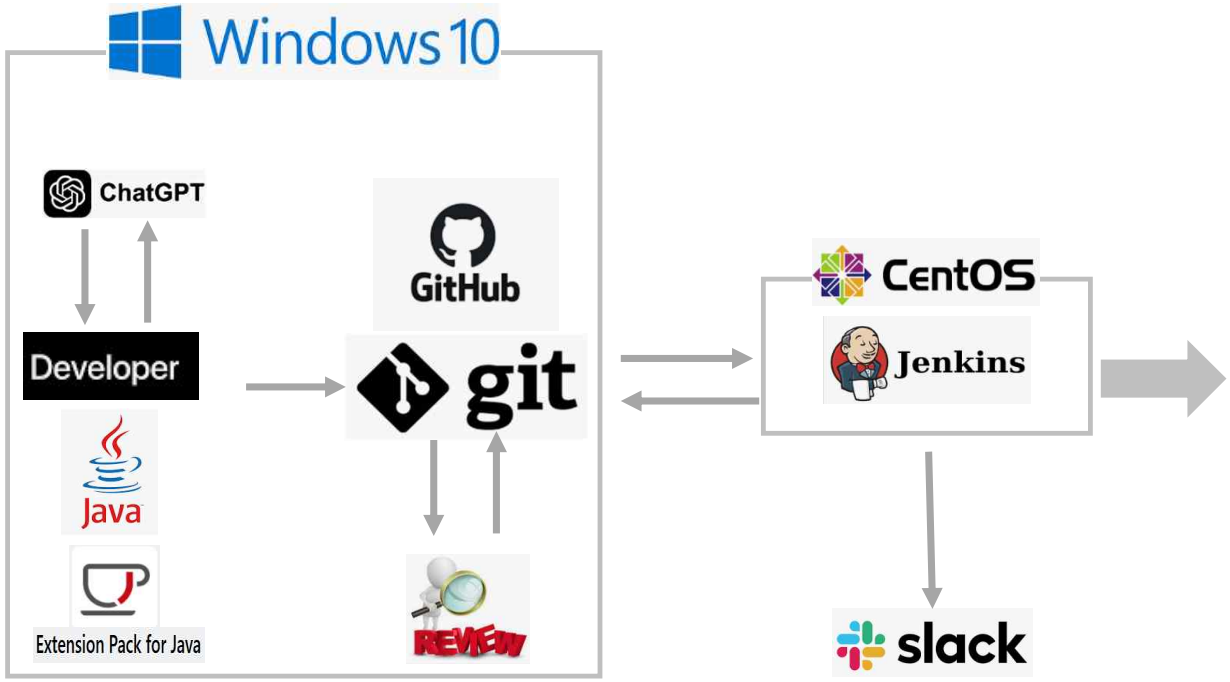
- **Role**
- Standardized frontend test code writing practices.
- Built Jenkins Job & Slack notification automation pipeline.

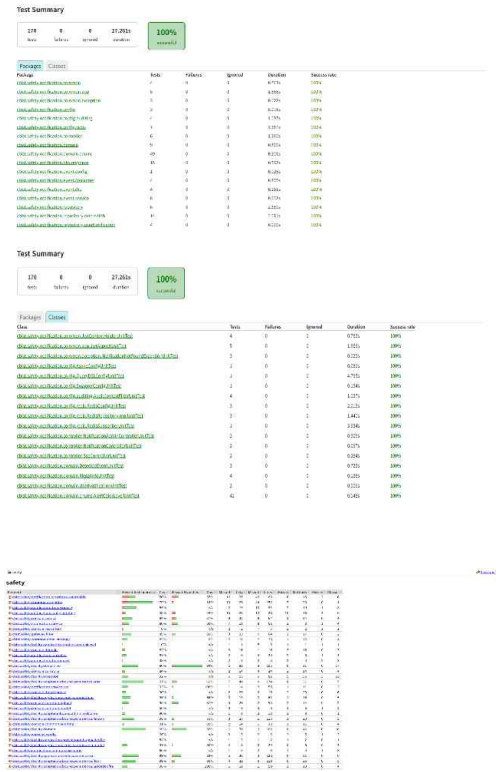


File	Statements		Branches		Functions		Lines	
area.ts	90%	27/30	16.66%	1/6	84.61%	11/13	90%	27/30
auth.ts	100%	14/14	50%	2/4	100%	4/4	100%	12/12
cctv.ts	87.5%	21/24	50%	1/2	80%	8/10	87.5%	21/24
file.ts	62.5%	5/8	100%	0/0	50%	2/4	62.5%	5/8
floor.ts	100%	28/28	50%	1/2	100%	12/12	100%	28/28
generalSetting.ts	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
poi.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

<Jenkins 로그 · Jest HTML Report · Coverage>

7. Unit Test Automation Pipeline – Backend (Java)





GitHub → Jenkins → JUnit → Slack

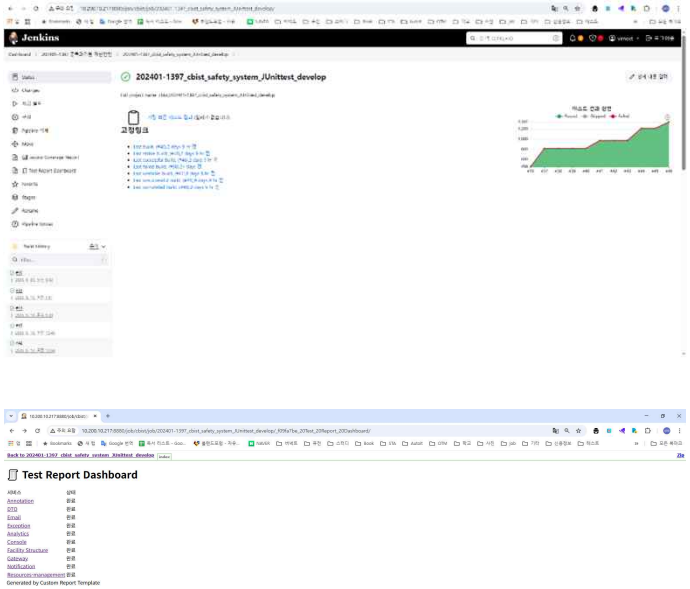
8. Backend (Java) – Test Execution Results & Coverage

- Spring Boot-based modular architecture
- Automated test execution in a Jenkins-based CI environment

- Java + JUnit-based unit testing
- Verification at Domain / Application / Adapter layer levels

- JUnit Report Dashboard
- Jenkins Test Trend (Graph & History)
- Gradle-based automated report generation

- 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

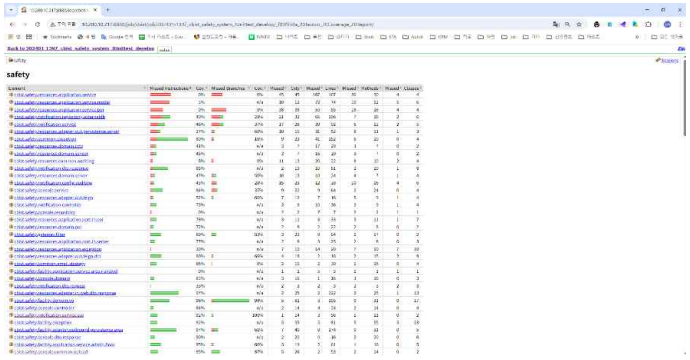


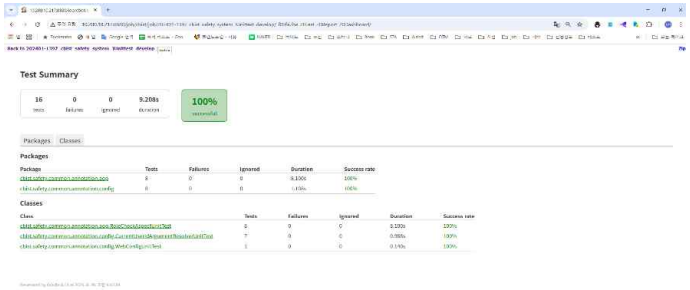
‹Jenkins Trend · JUnit Report · JaCoCo Coverage›

VIRNECT

9. Verify that SonarQube is installed in Docker

-
  - Spring Boot-based modular architecture
  - Automated test execution in a Jenkins-based CI environment
-
  - Measure Statement, Branch, Line Coverage using JaCoCo
  - Detailed coverage tracking per module and detection of uncovered lines
-
  - JaCoCo Coverage Report
  - Jenkins Coverage Integrated View
  - Gradle automated report generation scripts
-
  - 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›

2. Project Results

- Successfully deployed and configured SonarQube using Docker and Docker Compose on Linux, enabling a stable environment for code quality analysis.
- Integrated SonarQube with Jenkins CI/CD pipelines, ensuring continuous static code analysis for both frontend and backend services.
- Performed static analysis of automated test scripts using the SonarScanner CLI, identifying code smells, bugs, and vulnerabilities early in the development cycle.
- Established quality gates and code review workflows to enforce coding standards and reduce technical risks.
- Built centralized SonarQube dashboards to visualize key metrics such as code duplication, coverage trends, and security hotspots.
- Implemented Slack notifications and automated reporting, allowing real-time visibility of test quality status to the development and QA teams.
- Achieved measurable improvements in test script maintainability, significantly reducing technical debt and strengthening regression testing reliability.