Quality Assurance Results

CRAISELION : Handong Team Meeting Archiving and Exchange Web Platform

Version:	1.0
Date:	2024-06-08
Client:	Student
Confidential:	No
developer:	Seokjae Ma, Donggyu Kim, Sechang Jang, Junhyeok Choi, Minseo Lee

Team

Name	Email	ID	Role
Seokjae Ma	21800239@handong.ac.kr	21800239	Project Manager
Donggyu Kim	22000063@handong.ac.kr	22000063	Scrum Master
Sechang Jang	21900628@handong.ac.kr	21900628	Documenter
Junhyeok Choi	21900764@handong.ac.kr	21900764	Developer
MinSeo Lee	22100503@handong.ac.kr	22100503	Developer

Version

Version	Date	Description
1.0		

Table of Contents

1 Introduction

1.1 Project Background

Handong University hosts a variety of Residential Colleges (RCs), within which numerous team meetings are regularly held. These meetings play a crucial role in fostering community formation and interaction among students. However, the current system faces several significant challenges.

First, there are difficulties in exchanging information between teams within the same RC, and even more so between different RCs. This lack of interaction within and between RCs hinders the development of a community-centered culture at Handong, which the university aims to promote.

Second, there is a scarcity of guidance provided to team leaders, leading to challenges in leadership and team management. This situation diminishes the efficiency and effectiveness of team activities, negatively affecting student engagement and enthusiasm.

Third, the preservation of materials and information generated during team meetings throughout the year is not adequately addressed. Due to the absence of a proper archiving system, important documents are often lost, posing significant challenges to long-term project management and the reuse of materials.

To resolve these issues, we propose a web platform that facilitates communication and exchange among teams or RCs and preserves information and materials related to team meetings, thereby aiding in the development of the entire community at Handong University. This platform will serve as an essential resource for the student support team and the students (team executives), who are the main stakeholders, in enhancing the team culture and leadership development within Handong University.

2 Software Quality

2.1 Quality Assurance

Quality Assurance (QA) refers to the systematic processes and methodologies used to ensure that products or services meet specified requirements and are free of defects. It is a critical aspect of software development, manufacturing, and various other industries where quality and reliability are paramount.

Key Principles of Quality Assurance:

1. **Prevention over Detection**: QA emphasizes preventing defects rather than detecting and fixing them after they occur. This involves setting standards, procedures, and guidelines to ensure that errors are minimized from the outset.
2. **Continuous Improvement**: QA involves continually improving processes, methods, and tools to enhance product quality. This is often achieved through feedback loops, analyzing defects, and refining processes.
3. **Customer Focus**: QA focuses on meeting customer expectations and requirements. Understanding customer needs helps in defining quality standards and criteria.
4. **Standardization**: QA promotes standardization of processes and practices across the organization. This ensures consistency and reliability in product quality.
5. **Documentation and Traceability**: QA emphasizes documenting processes, procedures, and decisions to maintain traceability and accountability. This helps in identifying the root causes of issues and implementing corrective actions.

Examples of Quality Assurance

1. **Software Development**:
    - **Code Reviews**: Reviewing code by peers or using automated tools to ensure code quality and adherence to coding standards.
    - **Automated Testing**: Implementing automated tests (unit tests, integration tests, etc.) to verify software functionality and performance.
    - **Continuous Integration/Continuous Deployment (CI/CD)**: Using CI/CD pipelines to automate build, test, and deployment processes, ensuring consistent quality across releases.
    - **Bug Tracking**: Using bug tracking systems to capture and prioritize defects for resolution.
2. **Manufacturing**:
    - **Quality Control Inspections**: Performing inspections at various stages of production to ensure products meet quality standards.
    - **Statistical Process Control (SPC)**: Using statistical methods to monitor and control production processes, ensuring consistent product quality.

    - **Supplier Quality Management**: Assessing and monitoring the quality of materials and components sourced from suppliers.
3. **Service Industries**:
    - **Service Level Agreements (SLAs)**: Establishing SLAs to define and measure service quality levels provided to customers.
    - **Customer Feedback and Surveys**: Collecting and analyzing customer feedback to improve service quality and satisfaction.
    - **Training and Development**: Providing ongoing training and development to employees to enhance service delivery and customer interactions.

2.2 Quality Management

Quality Management (QM) in software development is a systematic approach to ensuring that the required level of quality is achieved throughout the software development lifecycle. It encompasses processes, standards, and activities aimed at delivering software products that meet defined quality goals and customer expectations. Here's an explanation of the key aspects mentioned:

Organizational Level:

1. Establishing Organizational Processes and Standards:
    - At the organizational level, Quality Management involves defining and implementing a framework of processes, methodologies, and standards. These are designed to ensure that software development activities consistently produce high-quality products.
    - This includes defining quality policies, guidelines, and procedures that all projects within the organization should follow.
    - Example: Implementing quality assurance practices, defining coding standards, establishing testing frameworks, and conducting regular audits to ensure compliance with these standards.

Project Level:

1. Application of Specific Quality Processes:
    - Quality Management at the project level involves applying specific processes and methodologies to ensure that quality objectives are met.
    - This includes implementing quality assurance activities such as code reviews, testing, and quality audits throughout the project lifecycle.
    - Example: Conducting code reviews to ensure adherence to coding standards, performing automated and manual testing to verify functionality and performance, and using tools for static code analysis.
2. Checking Compliance with Planned Processes:
    - Project-level Quality Management also includes monitoring and verifying that planned quality processes and standards are being followed.
    - This ensures that deviations from quality plans are identified early and corrective actions can be taken promptly.
    - Example: Monitoring test coverage, reviewing adherence to project documentation standards, and tracking compliance with defined quality metrics.
3. Establishing a Quality Plan:
    - Quality Management also involves creating a Quality Plan for each project.
    - The Quality Plan outlines quality goals, objectives, and criteria specific to the project. It defines the processes, standards, and metrics that will be used to achieve and measure quality.
    - Example: Defining acceptance criteria, specifying testing strategies, outlining configuration management processes, and establishing metrics for defect tracking and resolution.

Benefits of Quality Management in Software Development:

- Consistency: Ensures consistent delivery of high-quality software products across projects.
- Risk Mitigation: Identifies and addresses quality issues early in the development process, reducing risks associated with defects and failures.
- Customer Satisfaction: Increases customer satisfaction by delivering software that meets or exceeds expectations in terms of functionality, reliability, and performance.
- Efficiency: Improves efficiency through streamlined processes, optimized workflows, and reduced rework.
- Compliance: Ensures compliance with industry standards, regulations, and best practices.
- Continuous Improvement: Facilitates continuous improvement by collecting feedback, analyzing metrics, and implementing lessons learned from previous projects.

3. Source Code Management

GitHub

GitHub, as a Source Control Management (SCM) system, provides a comprehensive platform for version control and collaboration on software development projects. Here are the key features and functionalities that GitHub offers as an SCM:

Version Control

1. **Git-Based Version Control:**
    - GitHub uses Git, a distributed version control system that tracks changes in source code during software development.
    - It allows multiple developers to work on a project simultaneously without overwriting each other's changes.
2. **Commit and History:**
    - Developers make changes to the codebase and commit these changes with messages describing what was done.
    - The commit history provides a detailed log of all changes, who made them, and why.
3. **Branches and Merging:**
    - Branches allow developers to work on different features or bug fixes in isolation from the main codebase (often called the `main` or `master` branch).
    - Once changes in a branch are ready, they can be merged back into the main branch.

Collaboration

1. **Pull Requests:**
    - Pull requests (PRs) are a way to propose changes to the codebase. Developers create PRs to merge their branch into another branch.
    - PRs facilitate code reviews, discussions, and approvals before the changes are merged.
2. **Code Reviews:**
    - Code reviews are an integral part of PRs where team members can comment on the code, suggest improvements, and approve or request changes.
    - This ensures that the code meets the project's quality standards and helps share knowledge among team members.
3. **Issues and Project Management:**
    - GitHub provides an integrated issue tracking system to manage bugs, feature requests, and other tasks.
    - Labels, milestones, and project boards can be used to organize and prioritize work.

Continuous Integration and Deployment (CI/CD)

1. **Third-Party Integrations:**

- GitHub integrates with many CI/CD tools like Jenkins, CircleCI, Travis CI, and others.
- These integrations help streamline the build and deployment process, ensuring that code changes are tested and deployed efficiently.

Collaboration and Community

1. **Forks and Contributions:**
   - Forking allows users to create their own copy of a repository, which they can modify without affecting the original repository.
   - This is commonly used in open-source projects where contributors fork the repository, make changes, and then submit pull requests to the original project.

Security and Permissions

1. **Access Controls:**
   - GitHub allows repository owners to set permissions and manage access for collaborators.
   - Teams and organizations can have different access levels, from read-only to full administrative rights.

2. **Security Alerts and Dependency Management:**
   - GitHub can automatically scan repositories for vulnerabilities in dependencies and suggest updates.
   - Security alerts notify maintainers about potential security issues in their code or dependencies.

https://github.com/2024-SE-Project

RAONz: Coding Standard

1. Creating an Issue

✹ **How to Use Git Issues**

- Create an issue for the feature you will work on.
- Standardize the issue title as [Type] - Description (e.g., [Style] - Add text style).
- Tag the person responsible for the task in Assignees.
- Tag the relevant type in Labels.
- In the description field, include details about the task, any related issue numbers, and any references.
- Once the issue is created, it will be assigned an issue number #N.

2. Creating a Branch

✹ **How to Use Git Branches**

- Work only on the branch you have created. (Refer to How to Use Issues for branch creation.)
- The branch naming convention is <yourname_type/#issuenumber> (e.g., ehdrb01_feat/#1).

3. Changing Branch with Checkout

shell

```
git checkout ehdrb01/#12
```

4. Working

- Commit after completing each task.
- Merge the branch you worked on into the develop branch via a pull request.
- Merge after code review.

✹ **How to Write Commit Messages**

- **Type**: Short description (in Korean)
  - feat: 로그인
  - e.g., style: 텍스트 디자인 시스템 구축

Type	Description	Example
feat	Add or implement a new feature	feat: 로그인 기능 구현
edit	Fix a simple typo	edit: 로그인 캐시 처리 방식 수정
style	Add or modify UI/Style-related files	style: 폰트 등록
add	Add asset files (images, icons, etc.)	add: 위젯 이미지 추가
chore	Move or rename files/paths	chore: feet -> feat 이름 변경
delete	Delete dump files	delete: Empty.md 파일 삭제
merge	Merge branches	merge: pull request #3 from ehdrb01_style/#1
fix	Fix bugs	fix: Color 버그 수정
docs	Work on documentation	docs: Readme 작성
refactor	Refactor code	refactor: 변수명 수정

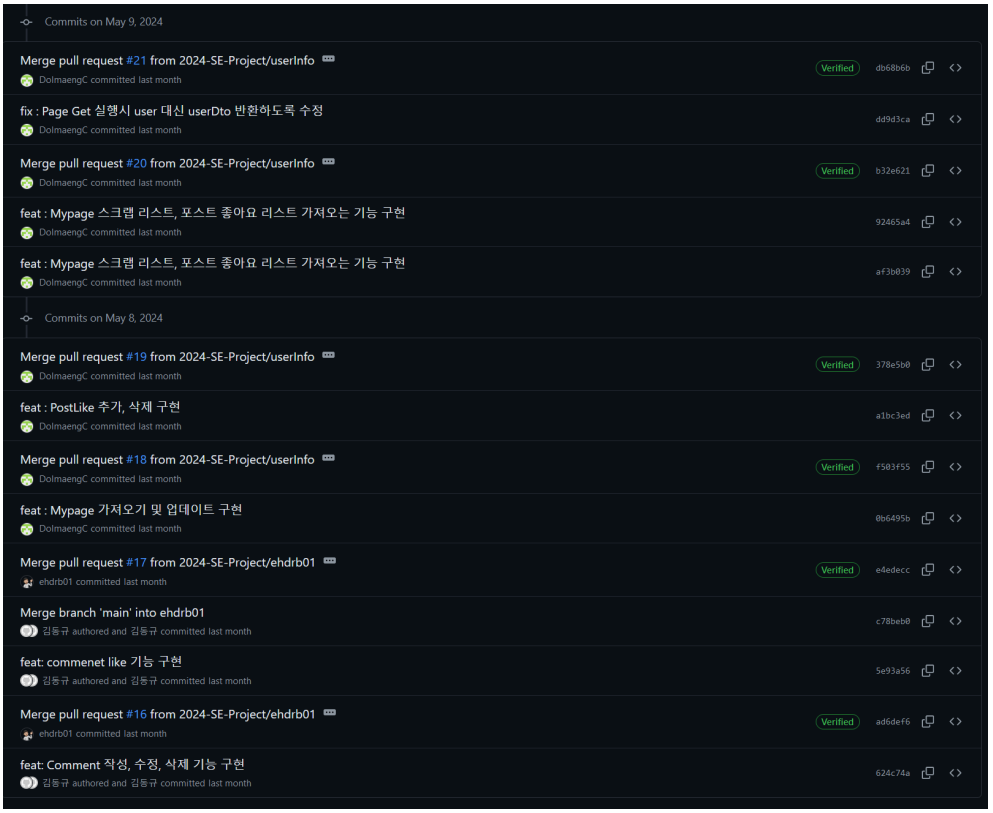
model	Work on the database (model)	model: 데이터 모델 생성
init	Initialize the project	init: 프로젝트 생성
test	Create test cases	test: 테스트 케이스 생성
build	Rebuild	build: 동일 버전 재빌드(x.xx)
version	Update version	version: 버전(2.0.0) 업데이트

✷ **How to Write Pull Request Titles**

- Format: name_type/#issuenumber -> target branch (e.g., ehdrb01_style/#1 -> dev)

Example


Commits on May 11, 2024
Merge pull request #27 from 2024-SE-Project/ehdrb01
ehdrb01 committed 3 weeks ago
Verified
76313cf
feat: Post response 오류 수정
김동규 authored and 김동규 committed 3 weeks ago
cf37a09
feat: Post Response 오류 수정
김동규 authored and 김동규 committed 3 weeks ago
2e86a88
feat: 변경사항 merge
김동규 authored and 김동규 committed 3 weeks ago
0869aa6
Merge pull request #26 from 2024-SE-Project/ehdrb01
ehdrb01 committed 3 weeks ago
Verified
6f47e05
Merge branch 'main' into ehdrb01
ehdrb01 committed 3 weeks ago
Verified
11b9442
feat: 변경사항 merge
김동규 authored and 김동규 committed 3 weeks ago
1815ece
Merge pull request #25 from 2024-SE-Project/post
DolmaengC committed 3 weeks ago
Verified
371421e
feat : Post fileUpload 추가 임시 구현
DolmaengC committed 3 weeks ago
bd33835
Commits on May 10, 2024
Merge pull request #24 from 2024-SE-Project/team
DolmaengC committed 3 weeks ago
Verified
3403edd
feat : Team 관련 기능 구현
DolmaengC committed 3 weeks ago
57605da
Merge pull request #23 from 2024-SE-Project/myPage
DolmaengC committed 3 weeks ago
Verified
6fd222e
feat : Team 생성, Team member 추가 구현
DolmaengC committed 3 weeks ago
5a0d44a
Merge pull request #22 from 2024-SE-Project/getPost
DolmaengC committed last month
Verified
24a3598
feat : Page Get 실행시 좋아요 및 스크랩 여부 반환 구현
DolmaengC committed last month
0564295


Commits on May 9, 2024
Merge pull request #21 from 2024-SE-Project/userInfo
DolmaengC committed last month
Verified
db68b6b
fix : Page Get 실행시 user 대신 userDto 반환하도록 수정
DolmaengC committed last month
dd9d3ca
Merge pull request #20 from 2024-SE-Project/userInfo
DolmaengC committed last month
Verified
b32e621
feat : Mypage 스크랩 리스트, 포스트 좋아요 리스트 가져오는 기능 구현
DolmaengC committed last month
92465a4
feat : Mypage 스크랩 리스트, 포스트 좋아요 리스트 가져오는 기능 구현
DolmaengC committed last month
af3b039
Commits on May 8, 2024
Merge pull request #19 from 2024-SE-Project/userInfo
DolmaengC committed last month
Verified
378e5b0
feat : PostLike 추가, 삭제 구현
DolmaengC committed last month
a1bc3ed
Merge pull request #18 from 2024-SE-Project/userInfo
DolmaengC committed last month
Verified
f503f55
feat : Mypage 가져오기 및 업데이트 구현
DolmaengC committed last month
0b6495b
Merge pull request #17 from 2024-SE-Project/ehdrb01
ehdrb01 committed last month
Verified
e4edecc
Merge branch 'main' into ehdrb01
김동규 authored and 김동규 committed last month
c78beb0
feat: commenet like 기능 구현
김동규 authored and 김동규 committed last month
5e93a56
Merge pull request #16 from 2024-SE-Project/ehdrb01
ehdrb01 committed last month
Verified
ad6def6
feat: Comment 작성, 수정, 삭제 기능 구현
김동규 authored and 김동규 committed last month
624c74a

4. Continous Integration

4.1 Jenkins

Jenkins is an open-source automation server widely used for continuous integration (CI) and continuous delivery (CD) of software projects. It helps automate the build, test, and deployment phases of the software development lifecycle, facilitating faster and more reliable software delivery. Here's an overview of Jenkins and its key features:

Key Concepts in Jenkins:

1. **Continuous Integration (CI)**:
    - **Automated Builds**: Jenkins automates the process of compiling code, running tests, and generating artifacts (e.g., JAR files).
    - **Triggered Builds**: Builds can be triggered automatically whenever new code is committed to the repository, ensuring early detection of integration issues.
2. **Continuous Delivery/Deployment (CD)**:
    - **Automated Deployment**: Jenkins supports automating deployment processes to various environments (e.g., development, staging, production).
    - **Pipeline as Code**: Jenkins allows defining pipelines (sequences of steps) as code using Groovy DSL or Jenkinsfile, enabling version control and reuse.
3. **Plugins Ecosystem**:
    - **Extensibility**: Jenkins has a vast ecosystem of plugins (~1700 plugins) that extend its functionality for various tasks such as integrating with version control systems, build tools (Maven, Gradle), testing frameworks, cloud platforms (AWS, Azure), and more.
    - **Customization**: Plugins allow customization of Jenkins to suit specific project requirements and workflows.
4. **Monitoring and Reporting**:
    - **Dashboard**: Jenkins provides a web-based interface (dashboard) for monitoring build statuses, trends, and project health.
    - **Build Logs and Reports**: Detailed logs and reports help developers diagnose build failures, track test results, and analyze performance metrics.

Typical Use Cases for Jenkins:

- **CI/CD Pipelines**: Automating the build, test, and deployment processes to achieve continuous integration and delivery.
- **Scheduled Jobs**: Running periodic tasks such as backups, reports generation, or maintenance scripts.
- **Automated Testing**: Running unit tests, integration tests, and other types of automated tests as part of the build process.
- **Version Control Integration**: Integrating with version control systems (e.g., Git, SVN) to trigger builds on code changes and manage source code.
- **Release Management**: Managing and automating software releases across different environments (dev, test, prod).

Our team used Jenkins which is provided.

http://203.252.112.23:8181/

How to Integrate Your GitHub Repository to Your Jenkins Project

4.2 Setting Up for the FrontPage

The pipeline is triggered by push and pull requests.
This pipeline is to test react frontend.

Pipeline

```
pipeline {
	agent any

	stages {
	stage('Git Checkout') {
	steps {
	git branch: 'main', credentialsId: 'Group1', url: 'https://github.com/2024-SE-Project/FrontEnd.git'
	}
	}

	stage('Install Dependencies') {
	steps {
	dir('/data/jenkins/.jenkins/workspace/2024-1-SE-Group-1-Frontend/hgu_tmaew_front') {
	script {
		sh 'npm install'
	}
	}
	}
	}

	stage('Run Tests') {
	steps {
	dir('/data/jenkins/.jenkins/workspace/2024-1-SE-Group-1-Frontend/hgu_tmaew_front') {
	script {
		sh 'npm test'
	}
	}
	}
	}

	stage('Build') {
	steps {
	dir('/data/jenkins/.jenkins/workspace/2024-1-SE-Group-1-Frontend/hgu_tmaew_front') {
	script {
		sh 'npm run build'
	}
	}
	}
	}
	}

	post {
	always {
	archiveArtifacts artifacts: '**/target/*.war', allowEmptyArchive: true
	}

	success {
	echo 'Build and tests were successful!'
	}

	failure {
	echo 'Build or tests failed.'
	}
	}
}
```

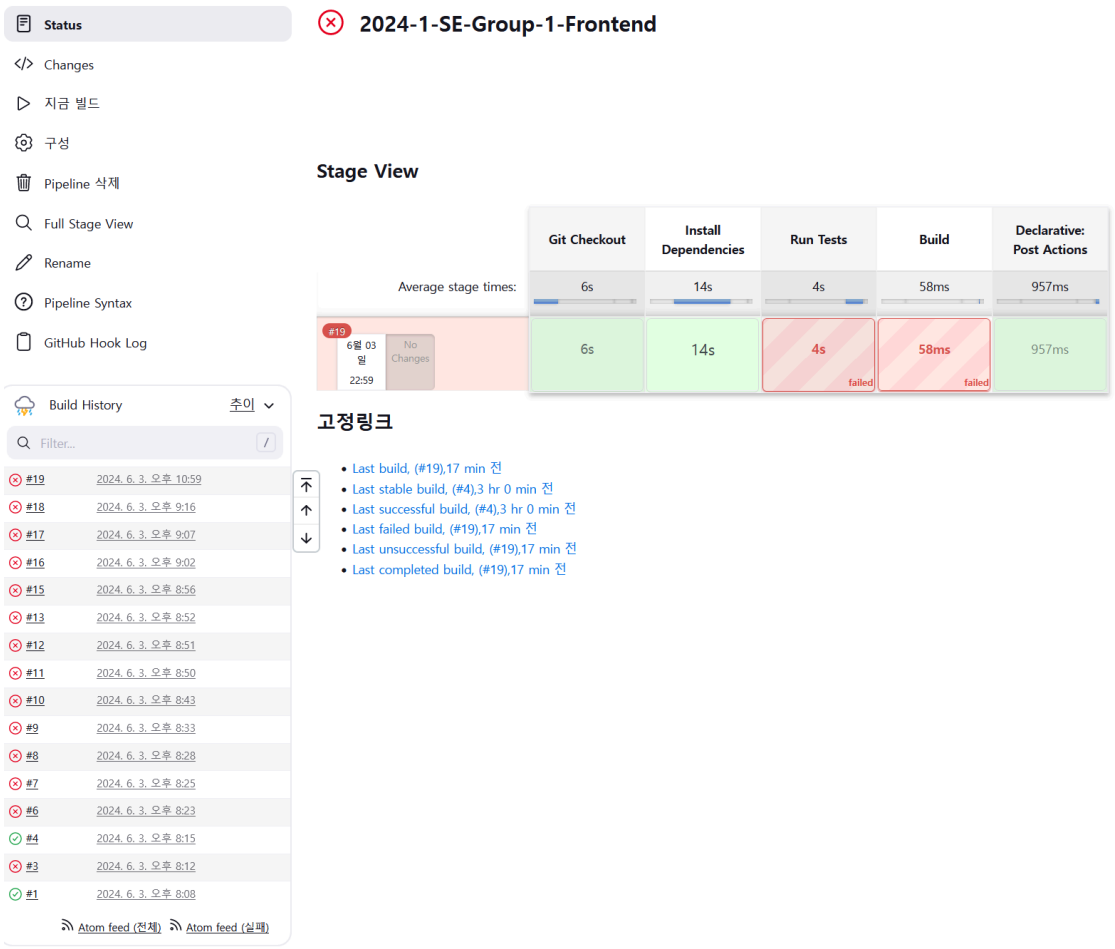
Explanation of the Pipeline:

1. **Pipeline Block (`pipeline`)**:
    - `agent any`: Specifies that Jenkins can allocate any available agent (slave node) to run this pipeline.
2. **Stages**:
    - **Git Checkout**:
        - Uses the `git` step to clone the repository (`https://github.com/2024-SE-Project/FrontEnd.git`) on the `main` branch using credentials stored in Jenkins (`credentialsId: 'Group1'`).
    - **Install Dependencies**:
        - Changes directory to `/data/jenkins/.jenkins/workspace/2024-1-SE-Group-1-Frontend/hgu_tmaew_front`.
        - Executes `npm install` to install project dependencies. This assumes it's a Node.js project (`package.json` exists).
    - **Run Tests**:
        - Similarly changes directory to the project directory.
        - Executes `npm test` to run tests. Adjust this command based on how your tests are configured (e.g., Jest, Mocha, etc.).
    - **Build**:
        - Changes directory to the project directory.
        - Executes `npm run build` to build the project. Adjust this command based on how your project is configured to build (e.g., webpack, gulp, etc.).
3. **Post Actions**:
    - **archiveArtifacts**: Archives artifacts matching `**/target/*.war` after each run. Change this to match your artifact's location and type.
    - **success**: Echoes a message when the pipeline completes successfully.
    - **failure**: Echoes a message when the pipeline fails.

Additional Notes:

- **Directory (`dir`)**: Used to change the current directory for specific stages. This ensures that commands (`sh`) are executed in the correct context.
- **Environment Setup**: Ensure that the Jenkins agent running this pipeline has Node.js and npm installed, and that the paths (`/data/jenkins/.jenkins/workspace/2024-1-SE-Group-1-Frontend/hgu_tmaew_front`) match your Jenkins environment.
- **Credentials (`credentialsId`)**: Replace `'Group1'` with the actual credentials ID configured in Jenkins for accessing the repository.

Example



It is the example that all the recent commits didn’t pass the test cases, and failed.

4.3 Setting Up for the FrontPage

The pipeline is triggered by push and pull requests.
This pipeline is to test the spring boot backend page.

Pipeline

```
{
    agent any

    environment {
        // Define any environment variables if needed
        GRADLE_OPTS = '-Xmx1024m'
    }

    stages {
        stage('Git Checkout') {
            steps {
                git branch: 'main', credentialsId: 'Group1', url: 'https://github.com/2024-SE-Project/BackEnd.git'
            }
        }

        stage('Build') {
            steps {
                dir('/data/jenkins/.jenkins/workspace/2024-1-SE-Group-1-Backend/raonz') {
                    script {
                        sh './gradlew clean build'
                    }
                }
            }
        }

        stage('Run Tests') {
            steps {
                dir('/data/jenkins/.jenkins/workspace/2024-1-SE-Group-1-Backend/raonz') {
                    script {
                        sh './gradlew test'
                    }
                }
            }
        }

        stage('Package') {
            steps {
                dir('/data/jenkins/.jenkins/workspace/2024-1-SE-Group-1-Backend/raonz') {
                    script {
                        sh './gradlew assemble'
                    }
                }
            }
        }
    }

    post {
        always {
            archiveArtifacts artifacts: 'build/libs/*.jar', allowEmptyArchive: true
            junit 'build/test-results/test/*.xml'
        }
        success {
            echo 'Build, tests, and deployment were successful!'
        }
        failure {
            echo 'Build, tests, or deployment failed.'
        }
    }
}
```

Explanation of the Pipeline:

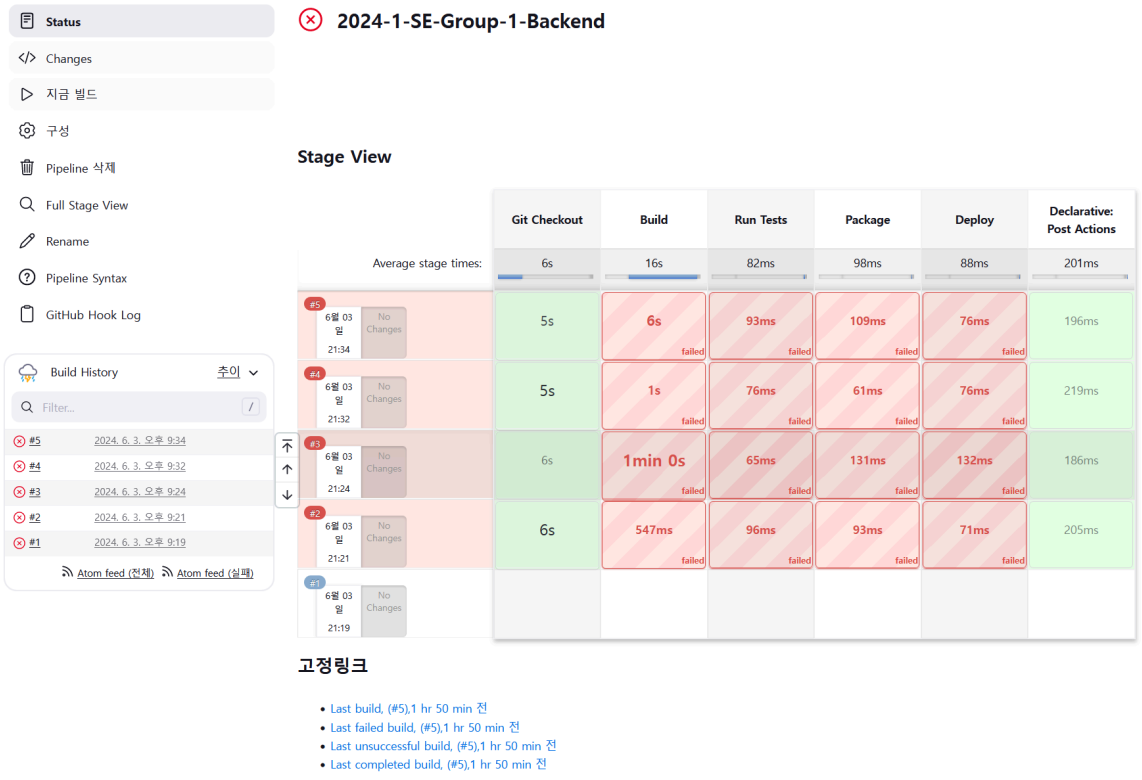
1. **Pipeline Block (`pipeline`)**:
   - `agent any`: Specifies that Jenkins can allocate any available agent (slave node) to run this pipeline.
2. **Environment Block (`environment`)**:
   - `GRADLE_OPTS = '-Xmx1024m'`: Sets an environment variable `GRADLE_OPTS` with a specific memory configuration (`-Xmx1024m`). This is optional but can be useful for adjusting JVM memory settings for Gradle builds.
3. **Stages**:
   - **Git Checkout**:
     - Checks out the code from the Git repository (`https://github.com/2024-SE-Project/BackEnd.git`) on the `main` branch using credentials stored in Jenkins (`credentialsId: 'Group1'`).
   - **Build**:
     - Changes directory to `/data/jenkins/.jenkins/workspace/2024-1-SE-Group-1-Backend/raonz`.
     - Executes `./gradlew clean build` to clean previous build artifacts and then build the project using Gradle.
   - **Run Tests**:
     - Similarly changes directory to the project directory.
     - Executes `./gradlew test` to run the test suite using Gradle. Adjust this command based on how your tests are configured in your Gradle project.
   - **Package**:
     - Changes directory to the project directory.
     - Executes `./gradlew assemble` to package the application into a distributable format (e.g., JAR, WAR) using Gradle.
4. **Post Actions**:
   - **archiveArtifacts**: Archives artifacts matching `build/libs/*.jar` after each run. This assumes your Gradle build produces JAR files in the `build/libs` directory. Adjust the path if your artifacts are generated elsewhere.
   - **junit**: Publishes JUnit test results located in `build/test-results/test/*.xml`. Adjust the path if your test result files are located elsewhere.
5. **Success and Failure Handlers**:
   - Executes `echo` statements based on the pipeline's outcome (`success` or `failure`).

Additional Considerations:

- **Directory (`dir`)**: Used to change the current directory for specific stages (`Build`, `Run Tests`, `Package`). This ensures that Gradle commands (`sh './gradlew ...'`) are executed in the correct project context.
- **Credentials (`credentialsId`)**: Replace `'Group1'` with the actual credentials ID configured in Jenkins for accessing the repository.
- **Paths and Commands**: Ensure paths (`/data/jenkins/.jenkins/workspace/...`) match your Jenkins environment and project setup.
- **Artifact and Test Result Handling**: Adjust `archiveArtifacts` and `junit` paths according to your project's build and test output locations.

Example

Our springboot project requires Java 17, but the current Java version of the Jenkins server is Java 11. The build is not supported, unfortunately. However, we found out that it is buildable and testable using another laptop (that we can't use all day). It is just an example that jenkins is integrated with the Backend Development.



5. Static Analyzer

5.1 FeeDet

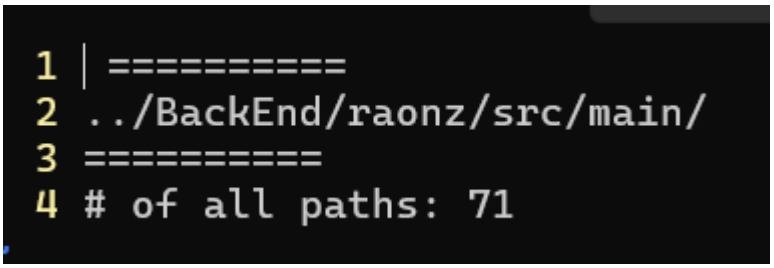
Jaechang Nam, Song Wang, Xi Yuan, and Lin Tan, A Bug Finder Refined by a Large Set of Open-Source Projects, Information and Software Technology, Volume 112, August 2019, Pages 164-175, DOI: https://doi.org/10.1016/j.infsof.2019.04.014, [Preprint].

This research focused on enhancing static bug detection tools by refining bug detection rules through an iterative feedback-based approach. By analyzing 1622 patches from various software projects, the study developed ten effective bug detection rules and implemented them in FeeFin. Applied to 1880 GitHub projects, FeeFin identified 98 new bugs, with developer validation confirming 57 as true bugs and 9 as false positives. The study concludes that this feedback-driven process not only mitigates false positives but also discovers new bug patterns, suggesting its potential to advance static bug detection tool effectiveness in real-world applications.

The FeeFin process described in the document runs through several stages to refine bug detection rules and improve the accuracy of static bug detection tools:

1. **Manual Patch Analysis:** Initially, common bug patterns are identified by analyzing patches from software repositories. This involves examining bug-fixing changes and linking them to bug reports to understand bug-introducing changes.
2. **Identification of Bug Patterns:** Based on the analysis of patches from projects like Lucene, Jackrabbit, Hadoop-common, and HBase, ten initial bug patterns are identified. These patterns include issues like redundant instantiations, illogical conditions, and incorrect directory slashes.
3. **Feedback-Based Rule Design:** The bug detection rules are iteratively refined using a feedback-based approach. This involves applying the initial rules to a large dataset of software projects (599 Java projects from ASF and Google), analyzing the results, and revising the rules to reduce false positives. Questions are formulated (e.g., "Does a condition statement compare the same expression with '=='?") to guide the rule refinement process.
4. **Implementation with FeeFin:** The refined bug detection rules are implemented in FeeFin, a static bug detection tool. FeeFin operates in two scenarios:
    - **Just-in-time (JIT) FeeFin:** Detects bugs before changes are committed, aiding developers during coding or before version control commits.
    - **Snapshot FeeFin:** Analyzes source code snapshots or release candidates for bugs, similar to traditional static analysis tools.
5. **Evaluation:** FeeFin's effectiveness is evaluated through case studies. This includes verifying FeeFin's detection results against known bugs in past commits of 599 projects (JIT scenario) and detecting new bugs in 1880 Java projects on GitHub (Snapshot FeeFin scenario).

Result on the Backend(Java) Process



Didn’t find any bug template

5.2 SonarQube

SonarQube is an open-source platform used for continuous inspection of code quality. It performs automatic reviews of code to detect bugs, code smells, and security vulnerabilities. SonarQube integrates with your existing workflows and provides detailed reports on the quality of your codebase, enabling development teams to maintain high standards of code quality and security.

We are planning to integrate SonarQube, Jenkins, and Github actions with the stable and independent server.

Code Analysis: SonarQube supports static code analysis for a wide range of programming languages including Java, C#, JavaScript, TypeScript, Python, PHP, and many more. It can analyze source code, test code, and even configuration files.

Quality Gates: SonarQube allows you to define quality gates, which are sets of conditions that your code must meet before it can be considered acceptable. This helps ensure that code changes meet your organization's quality standards before they are integrated into the main codebase.

Metrics and Reports: SonarQube provides detailed metrics and reports on various aspects of code quality, including:

- Code Smells: Maintainability issues that make code harder to read and maintain.
- Bugs: Defects that can lead to incorrect program behavior.
- Vulnerabilities: Security issues that can be exploited by attackers.
- Duplications: Identical or very similar blocks of code.
- Complexity: Measures of how complex code is, which can indicate potential maintenance issues.

Continuous Integration and Continuous Delivery (CI/CD): SonarQube integrates seamlessly with CI/CD tools like Jenkins, Azure DevOps, GitLab CI, GitHub Actions, and others. This allows you to run code analysis automatically as part of your build and deployment pipelines.

Integration with IDEs: SonarQube integrates with popular Integrated Development Environments (IDEs) like IntelliJ IDEA, Eclipse, and Visual Studio through plugins. This allows developers to get immediate feedback on code quality as they write code.

Customizable Rules: SonarQube comes with a rich set of default rules for code analysis. Additionally, you can customize these rules or create new ones to meet your specific coding standards and practices.

SonarLint: SonarLint is a companion product that provides on-the-fly feedback in your IDE, helping developers to fix issues before committing code.

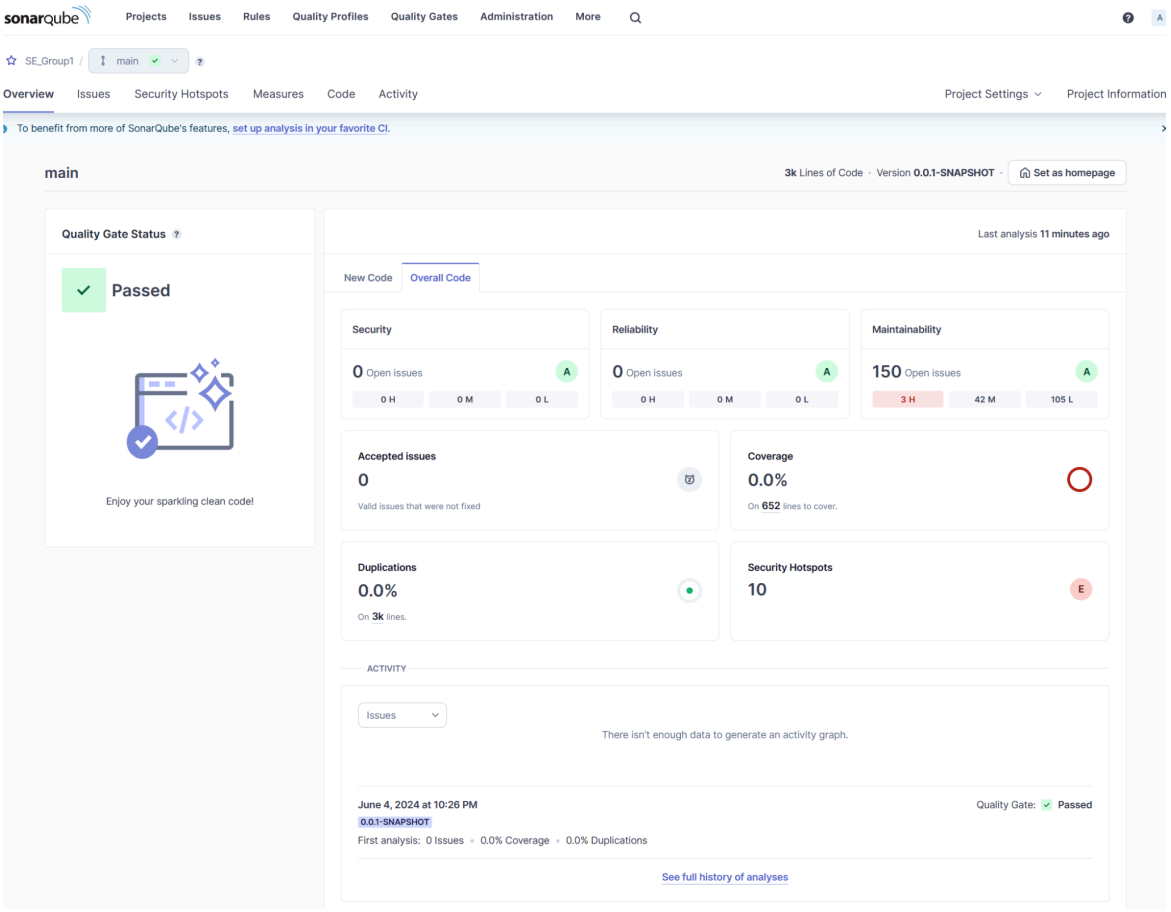
How SonarQube Works:

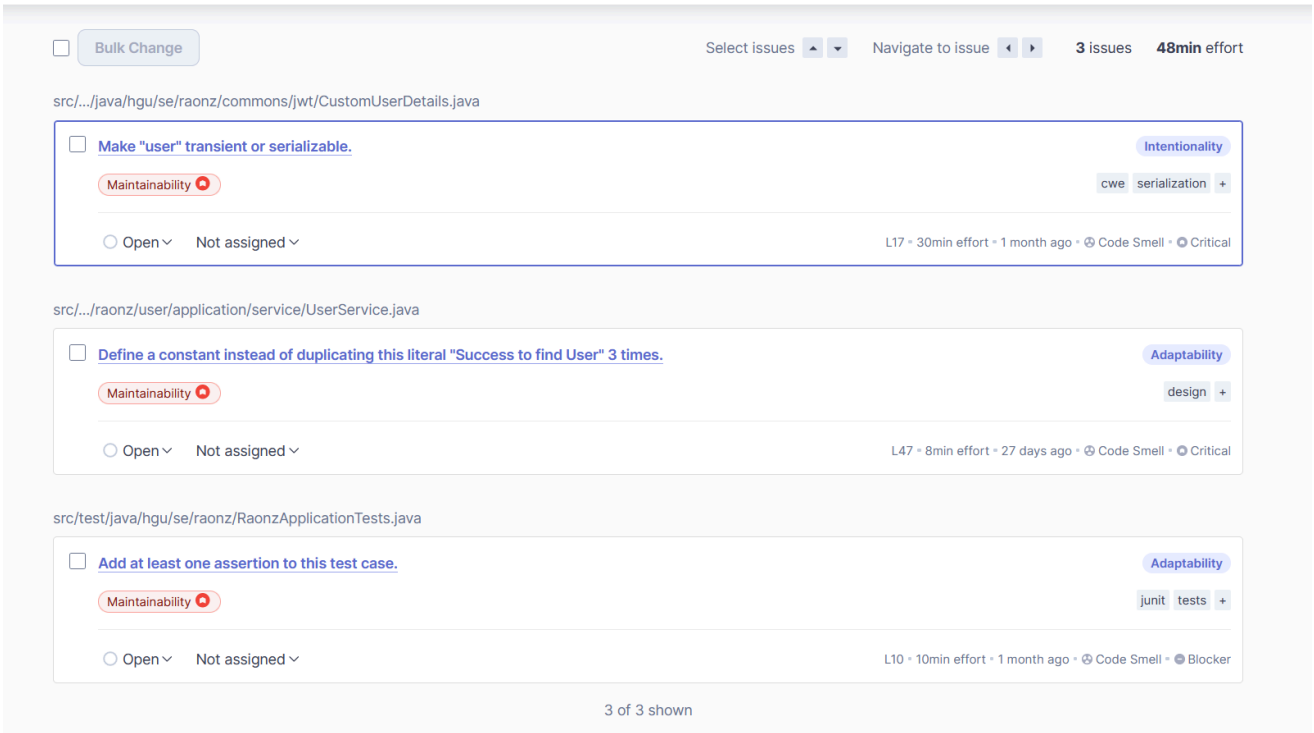
- Code Analysis: When you run a SonarQube scan, it analyzes the source code of your project based on predefined rules. The analysis results are sent to the SonarQube server where they are processed and stored.
- Results Visualization: The SonarQube server provides a web interface where you can view detailed reports and dashboards. These reports include information on code smells, bugs, vulnerabilities, test coverage, and more.
- Quality Gate Evaluation:
    - The analyzed results are evaluated against the configured quality gates.
    - If the code fails to meet the quality gate conditions, it can block the merge or deployment of the code.

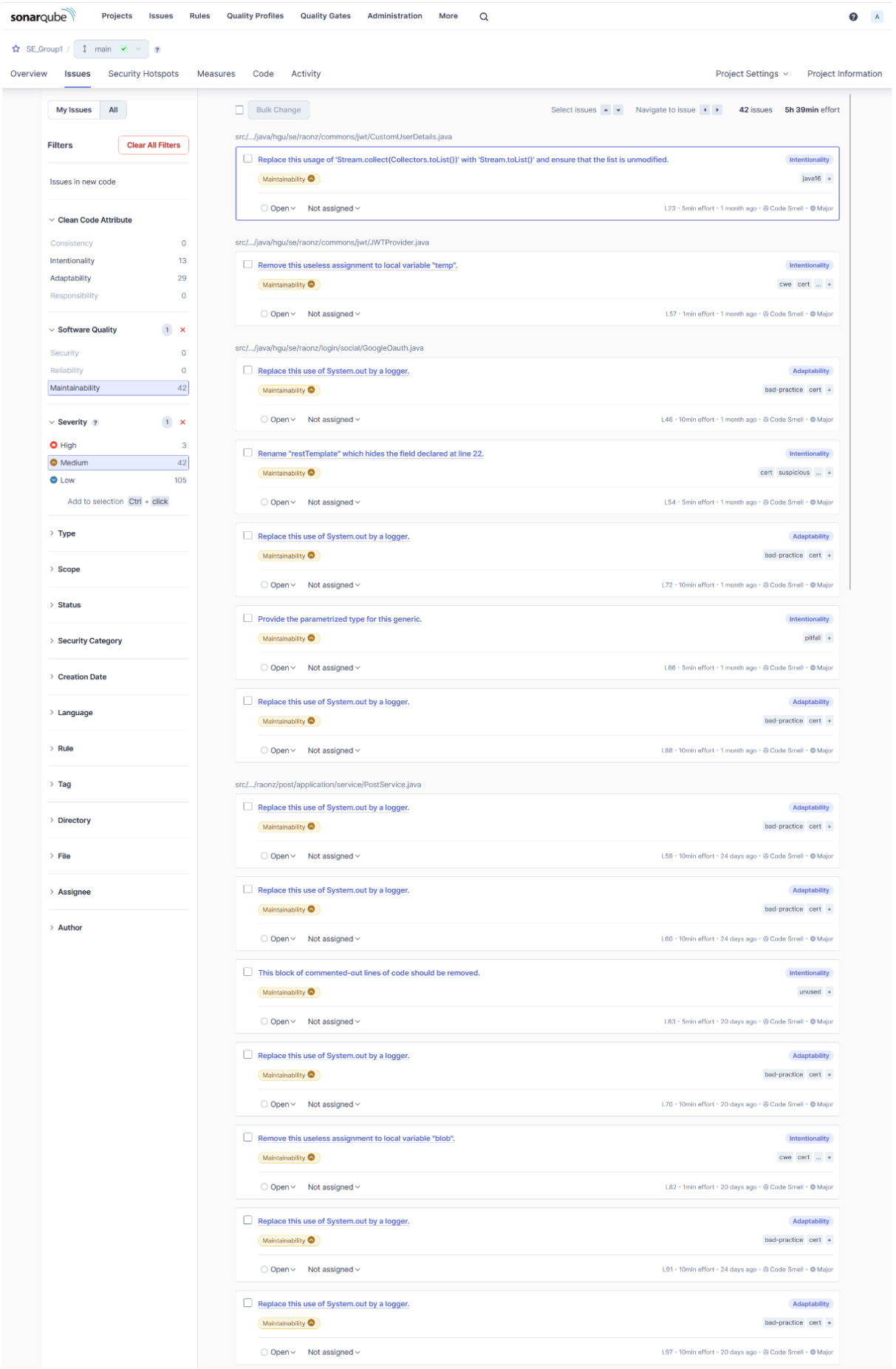
Sonarqube result

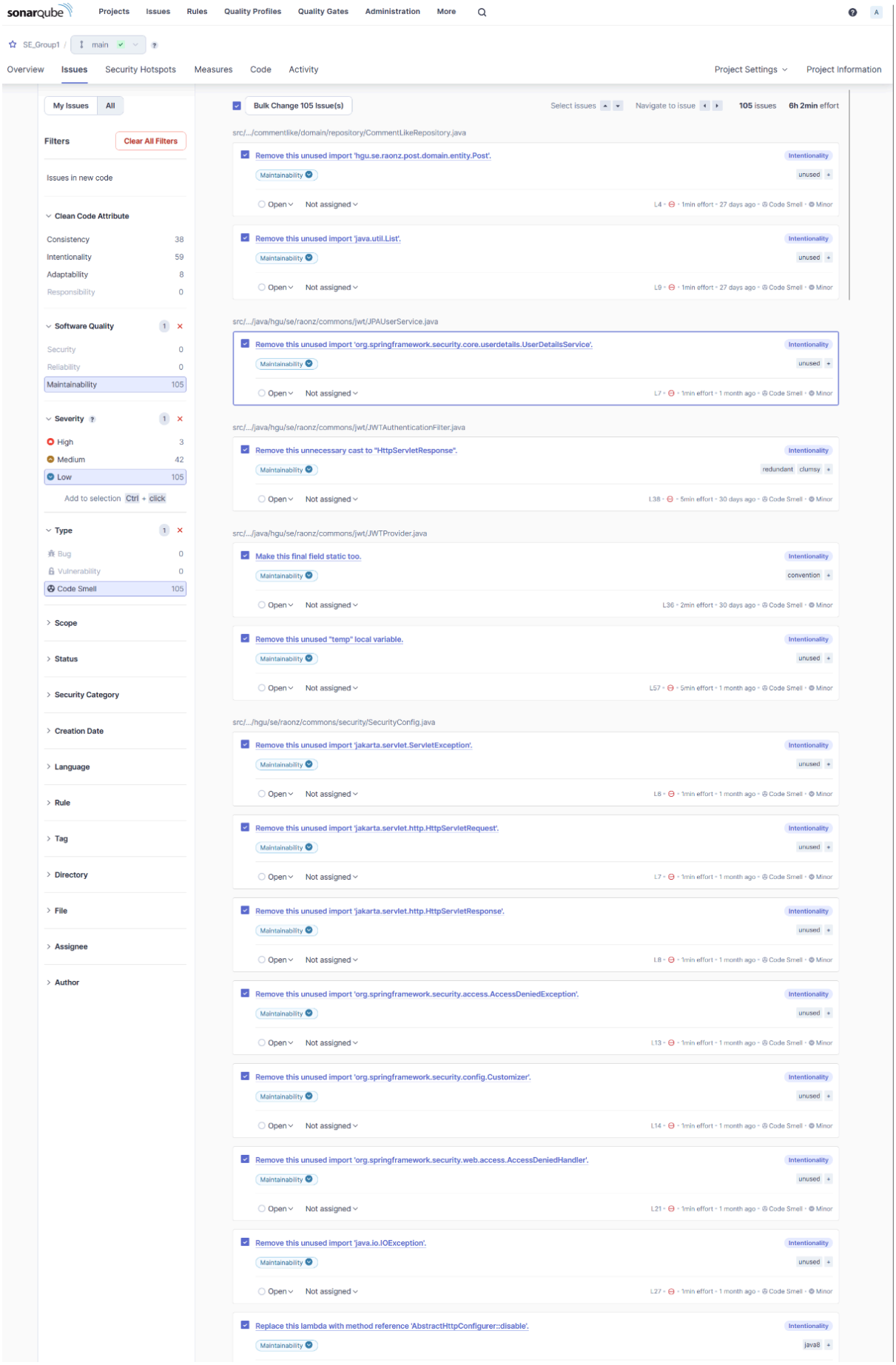
- “A” on security
- 0% of duplicated
- “A” on the maintainability
- “A” on the readability
- Need work on Jacoco to calculate the code coverage
- Most of the issues were false positives

Sonarqube Report Example






sonarqube
Projects
Issues
Rules
Quality Profiles
Quality Gates
Administration
More
SE_Group1 / main
Overview
Issues
Security Hotspots
Measures
Code
Activity
Project Settings
Project Information
My Issues
All
Filters
Clear All Filters
Issues in new code
Clean Code Attribute
Consistency 0
Intentionality 13
Adaptability 29
Responsibility 0
Software Quality 1
Security 0
Reliability 0
Maintainability 42
Severity 1
High 3
Medium 42
Low 105
Add to selection Ctrl + click
Type
Scope
Status
Security Category
Creation Date
Language
Rule
Tag
Directory
File
Assignee
Author
Bulk Change
Select issues
Navigate to issue
42 issues
5h 39min effort
src/.../java/hgu/se/raonz/commons/jwt/CustomUserDetails.java
Replace this usage of 'Stream.collect(Collectors.toList())' with 'Stream.toList()' and ensure that the list is unmodified.
Intentionality
Maintainability
java16
Open
Not assigned
L23 · 5min effort · 1 month ago · Code Smell · Major
src/.../java/hgu/se/raonz/commons/jwt/JWTProvider.java
Remove this useless assignment to local variable "temp".
Intentionality
Maintainability
cwe cert ...
Open
Not assigned
L57 · 1min effort · 1 month ago · Code Smell · Major
src/.../java/hgu/se/raonz/login/social/GoogleOauth.java
Replace this use of System.out by a logger.
Adaptability
Maintainability
bad-practice cert
Open
Not assigned
L46 · 10min effort · 1 month ago · Code Smell · Major
Rename "restTemplate" which hides the field declared at line 22.
Intentionality
Maintainability
cert suspicious ...
Open
Not assigned
L54 · 5min effort · 1 month ago · Code Smell · Major
Replace this use of System.out by a logger.
Adaptability
Maintainability
bad-practice cert
Open
Not assigned
L72 · 10min effort · 1 month ago · Code Smell · Major
Provide the parametrized type for this generic.
Intentionality
Maintainability
pitfall
Open
Not assigned
L86 · 5min effort · 1 month ago · Code Smell · Major
Replace this use of System.out by a logger.
Adaptability
Maintainability
bad-practice cert
Open
Not assigned
L88 · 10min effort · 1 month ago · Code Smell · Major
src/.../raonz/post/application/service/PostService.java
Replace this use of System.out by a logger.
Adaptability
Maintainability
bad-practice cert
Open
Not assigned
L59 · 10min effort · 24 days ago · Code Smell · Major
Replace this use of System.out by a logger.
Adaptability
Maintainability
bad-practice cert
Open
Not assigned
L60 · 10min effort · 24 days ago · Code Smell · Major
This block of commented-out lines of code should be removed.
Intentionality
Maintainability
unused
Open
Not assigned
L63 · 5min effort · 20 days ago · Code Smell · Major
Replace this use of System.out by a logger.
Adaptability
Maintainability
bad-practice cert
Open
Not assigned
L70 · 10min effort · 20 days ago · Code Smell · Major
Remove this useless assignment to local variable "blob".
Intentionality
Maintainability
cwe cert ...
Open
Not assigned
L82 · 1min effort · 20 days ago · Code Smell · Major
Replace this use of System.out by a logger.
Adaptability
Maintainability
bad-practice cert
Open
Not assigned
L91 · 10min effort · 24 days ago · Code Smell · Major
Replace this use of System.out by a logger.
Adaptability
Maintainability
bad-practice cert
Open
Not assigned
L97 · 10min effort · 20 days ago · Code Smell · Major


sonarqube
Projects
Issues
Rules
Quality Profiles
Quality Gates
Administration
More
SE_Group1 / main
Overview
Issues
Security Hotspots
Measures
Code
Activity
Project Settings
Project Information
My Issues
All
Filters
Clear All Filters
Issues in new code
Clean Code Attribute
Consistency 38
Intentionality 59
Adaptability 8
Responsibility 0
Software Quality 1
Security 0
Reliability 0
Maintainability 105
Severity 1
High 3
Medium 42
Low 105
Add to selection Ctrl + click
Type 1
Bug 0
Vulnerability 0
Code Smell 105
Scope
Status
Security Category
Creation Date
Language
Rule
Tag
Directory
File
Assignee
Author
Bulk Change 105 Issue(s)
Select issues
Navigate to issue
105 issues
6h 2min effort
src/.../commentlike/domain/repository/CommentLikeRepository.java
Remove this unused import 'hgu.se.raonz.post.domain.entity.Post'.
Intentionality
Maintainability
unused
Open
Not assigned
L4 • 1min effort • 27 days ago • Code Smell • Minor
Remove this unused import 'java.util.List'.
Intentionality
Maintainability
unused
Open
Not assigned
L9 • 1min effort • 27 days ago • Code Smell • Minor
src/.../java/hgu/se/raonz/commons/jwt/JPAUserService.java
Remove this unused import 'org.springframework.security.core.userdetails.UserDetailsService'.
Intentionality
Maintainability
unused
Open
Not assigned
L7 • 1min effort • 1 month ago • Code Smell • Minor
src/.../java/hgu/se/raonz/commons/jwt/JWTAuthenticationFilter.java
Remove this unnecessary cast to "HttpServletResponse".
Intentionality
Maintainability
redundant
clumsy
Open
Not assigned
L38 • 5min effort • 30 days ago • Code Smell • Minor
src/.../java/hgu/se/raonz/commons/jwt/JWTProvider.java
Make this final field static too.
Intentionality
Maintainability
convention
Open
Not assigned
L36 • 2min effort • 30 days ago • Code Smell • Minor
Remove this unused "temp" local variable.
Intentionality
Maintainability
unused
Open
Not assigned
L57 • 5min effort • 1 month ago • Code Smell • Minor
src/.../hgu/se/raonz/commons/security/SecurityConfig.java
Remove this unused import 'jakarta.servlet.ServletException'.
Intentionality
Maintainability
unused
Open
Not assigned
L6 • 1min effort • 1 month ago • Code Smell • Minor
Remove this unused import 'jakarta.servlet.http.HttpServletRequest'.
Intentionality
Maintainability
unused
Open
Not assigned
L7 • 1min effort • 1 month ago • Code Smell • Minor
Remove this unused import 'jakarta.servlet.http.HttpServletResponse'.
Intentionality
Maintainability
unused
Open
Not assigned
L8 • 1min effort • 1 month ago • Code Smell • Minor
Remove this unused import 'org.springframework.security.access.AccessDeniedException'.
Intentionality
Maintainability
unused
Open
Not assigned
L13 • 1min effort • 1 month ago • Code Smell • Minor
Remove this unused import 'org.springframework.security.config.Customizer'.
Intentionality
Maintainability
unused
Open
Not assigned
L14 • 1min effort • 1 month ago • Code Smell • Minor
Remove this unused import 'org.springframework.security.web.access.AccessDeniedHandler'.
Intentionality
Maintainability
unused
Open
Not assigned
L21 • 1min effort • 1 month ago • Code Smell • Minor
Remove this unused import 'java.io.IOException'.
Intentionality
Maintainability
unused
Open
Not assigned
L27 • 1min effort • 1 month ago • Code Smell • Minor
Replace this lambda with method reference 'AbstractHttpConfigurer::disable'.
Intentionality
Maintainability
java8

6. Dynamic Analyzer

6.1 Selenium Web GUI Test Tool

Selenium is a powerful tool for automating web browser interactions. It is commonly used for web testing, but it can also be used for web scraping and automating repetitive tasks performed on web applications.

Key Features of Selenium

1. Cross-browser Compatibility: Selenium supports multiple browsers, including Chrome, Firefox, Safari, and Internet Explorer.
2. Multiple Programming Languages: Selenium can be used with various programming languages such as Java, Python, C#, Ruby, and JavaScript.
3. Test Framework Integration: Selenium integrates well with test frameworks like JUnit, TestNG, and pytest.
4. Selenium Grid: Allows running tests on multiple machines and browsers simultaneously.
5. Selenium WebDriver: Provides a programming interface to create and execute test scripts to control browser actions.

Selenium Components

1. Selenium WebDriver: The most popular component. It allows for the creation of browser-based regression automation suites and tests.
2. Selenium IDE (Integrated Development Environment): A tool for recording and playing back tests in Firefox and Chrome.
3. Selenium Grid: Allows running tests in parallel across different machines and browsers.

All the test code conducted using Selenium is on the following links.

https://github.com/2024-SE-Project/GUI_Testing-

Example code

The following code is to test the element in the team meeting matching page

teamMeetingMatching.py > ...

```python
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import time

# Initialize the WebDriver (Make sure the path to your WebDriver is correct)
driver = webdriver.Chrome(executable_path='./chromedriver')

try:
    # Open the web application
    driver.get("http://localhost:3000/dashboard/teammatch")  # Change URL to your application's URL

    # Wait for the team matching container to load
    WebDriverWait(driver, 10).until(
        EC.presence_of_element_located((By.CLASS_NAME, "team-matching-container"))
    )

    # Verify the presence of the search bar
    search_bar = driver.find_element(By.CLASS_NAME, "search-bar")
    assert search_bar is not None, "Search bar not found."

    # Perform a search operation
    search_input = search_bar.find_element(By.TAG_NAME, "input")
    search_button = search_bar.find_element(By.CLASS_NAME, "search-button")
    search_input.send_keys("피구")
    search_button.click()
    time.sleep(1)  # Wait for search results to update

    # Verify the presence of the navigation buttons
    nav_buttons = driver.find_elements(By.CLASS_NAME, "reference-nav")[0].find_elements(By.TAG_NAME, "button")
    assert len(nav_buttons) == 2, "Navigation buttons not found or incorrect count."

    # Click each navigation button and verify content updates
    for button in nav_buttons:
        button.click()
        time.sleep(1)  # Wait for content to update

    # Verify the presence of the my team section
    my_team_section = driver.find_element(By.CLASS_NAME, "my-team")
    assert my_team_section is not None, "My Team section not found."

    # Verify the presence of the matching status section
    matching_status_section = driver.find_element(By.CLASS_NAME, "matching-status")
    assert matching_status_section is not None, "Matching Status section not found."

    # Verify the presence of team cards in the matching list
    team_cards = driver.find_elements(By.CLASS_NAME, "team-card")
    assert len(team_cards) > 0, "No team cards found."

    # Interact with the first team card: click details button
    first_team_card = team_cards[0]
    details_button = first_team_card.find_element(By.CLASS_NAME, "details-button")
    details_button.click()
    time.sleep(1)  # Wait for details to be displayed

    print("Test Passed: All interactions and verifications completed successfully.")

except Exception as e:
    print(f"Test Failed: {e}")

finally:
    # Close the browser window
    driver.quit()
```

6.2 Junit

JUnit is a popular framework for writing and running tests in Java. It is an essential tool for ensuring the correctness of your code by allowing you to write repeatable and automated tests. Here's an overview of JUnit testing, including its basic components, annotations, and a simple example.

Why Use JUnit?

1. **Automated Testing**: Run tests automatically to ensure your code behaves as expected.
2. **Repeatable**: Tests can be repeated as many times as needed, ensuring that changes in the code do not introduce new bugs.
3. **Documentation**: Tests can serve as documentation, showing how the code is intended to be used and what its expected behavior is.
4. **Refactoring Support**: Provides confidence to refactor code, knowing that existing functionality is tested.

Basic Components of JUnit

1. **Test Cases**: Methods within a test class that contain code to test specific functionality.
2. **Test Fixtures**: The setup needed to run tests, which typically includes initializing objects that are used in multiple tests.
3. **Assertions**: Statements that check if a condition is true. If the condition is false, the test fails.
4. **Annotations**: Special markers that indicate the role of a method or class in the testing framework.

JUnit Annotations

1. **@Test**: Marks a method as a test case.
2. **@BeforeEach**: Runs before each test. Used for setting up test fixtures.
3. **@AfterEach**: Runs after each test. Used for cleaning up test fixtures.
4. **@BeforeAll**: Runs once before all tests in the class. Used for setting up shared resources.
5. **@AfterAll**: Runs once after all tests in the class. Used for cleaning up shared resources.
6. **@Disabled**: Ignores the test method.

Application on the code

Testcase snippet in the code

```java

import hgu.se.raonz.comment.domain.entity.Comment;
import hgu.se.raonz.comment.domain.repository.CommentRepository;
import hgu.se.raonz.comment.presentation.request.CommentRequest;
import hgu.se.raonz.post.domain.entity.Post;
import hgu.se.raonz.post.domain.repository.PostRepository;
import hgu.se.raonz.user.domain.entity.User;
import hgu.se.raonz.user.domain.repository.UserRepository;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;

import java.util.Optional;

import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.ArgumentMatchers.*;
import static org.mockito.Mockito.*;

public class CommentServiceTest {

    @Mock
    private CommentRepository commentRepository;

    @Mock
    private PostRepository postRepository;

    @Mock
    private UserRepository userRepository;

    @InjectMocks
    private CommentService commentService;

    @BeforeEach
    void setUp() {
        MockitoAnnotations.openMocks(this);
    }

    @Test
    void testToAddComment_Success() {
        Long postId = 1L;
        String userId = "user1";
        CommentRequest commentRequest = new CommentRequest();

        Post post = new Post();
        post.setId(postId);

        User user = new User();
        user.setId(userId);

        when(postRepository.findById(postId)).thenReturn(Optional.of(post));
        when(userRepository.findById(userId)).thenReturn(Optional.of(user));
        when(commentRepository.save(any(Comment.class))).thenAnswer(invocation -> {
            Comment savedComment = invocation.getArgument(0);
            savedComment.setId(1L); // Simulating saving and setting ID
            return savedComment;
        });

        Long commentId = commentService.toAdd(postId, userId, commentRequest);

        assertNotNull(commentId);
        assertEquals(1L, commentId); // Assuming comment ID returned is 1
    }
```

Testcase execution

Simple execution of the Junit test case.

Other classes were difficult to test functionality due to the integration with the database.

Test Summary

28	0	0	3.835s
tests	failures	ignored	duration

100% successful

Packages Classes

Package	Tests	Failures	Ignored	Duration	Success rate
hgu.se.raonz.comment.application.service	9	0	0	3.478s	100%
hgu.se.raonz.comment.domain.entity	2	0	0	0.006s	100%
hgu.se.raonz.commentlike.application.service	6	0	0	0.174s	100%
hgu.se.raonz.commentlike.domain.entity	2	0	0	0.005s	100%
hgu.se.raonz.commons.jwt	1	0	0	0.005s	100%
hgu.se.raonz.post.domain.entity	1	0	0	0.001s	100%
hgu.se.raonz.scrap.application.service	7	0	0	0.166s	100%

7. User Tests

7.1 Alpha Test

Alpha testing is a type of software testing performed by internal teams or a designated group of users at the developer's site. Its purpose is to identify issues with the software's functionality, usability, and overall performance before releasing it to a broader audience or conducting beta testing.

Focused on the following features:

- Usability Testing
- Functional Testing
- Performance Testing
- Error Handling Testing
- Security Testing

Response from Sechang Jang (21900628)

- **Issue:** Users found the registration process confusing
    - **Severity:** Moderate
    - **Feedback:** Users suggested providing clearer instructions and more intuitive design

2. Performance Concerns:

- **Issue:** Loading times for the news feed were slow, especially during peak usage times.
    - **Severity:** High
    - **Feedback:** Users reported frustration with delays and suggested implementing caching mechanisms to improve load times.

3. Functional Concerns:

- **Issue:** Not that I have found

4. Security Vulnerabilities:

- **Issue:** Not that I have found

5. Feature Requests:

- **Request:** Users expressed interest in a dark mode feature for better nighttime usability.
    - **Feedback:** Incorporating a dark mode option could enhance user experience, especially for users who prefer using the app in low-light environments.

6. General Feedback:

- **Feedback:** Overall, users appreciated the intuitive layout and interactive features like liking and commenting on posts.
    - **Action Taken:** Developer plans to enhance existing features based on feedback while addressing critical issues promptly.

Response from Minseo Lee (22100503)

1. Usability Issues:

- **Issue:** Not that I found

2. Performance Concerns:

- **Issue:** Dashboard loading times were slow, especially when rendering card sections
    - **Severity:** High
    - **Feedback:** Users reported delays in accessing data. Fetching and rendering processes to required improvement on the dashboard performance.

3. Functional Bugs:

- **Issue:** Not that I found

4. Security Vulnerabilities:

- **Issue:** Not that I found

5. Feature Requests:

- **Request:** Not that I found

6. General Feedback:

- **Feedback:** Users appreciated the intuitive collaborative features for team communication.
- **Action Taken:** Developer plans to enhance task management capabilities and prioritize performance optimizations.