

# TQS: Quality Assurance manual

Filipe Sousa [114196], Gonalo Calvo [120131], Daniel Simbe [110235]

v2025-12-19

## Table of Contents

<b>1 Project management</b>	<b>2</b>
1.1 Assigned roles	2
1.2 Backlog Grooming and Progress Monitoring	2
Backlog Structure:	2
Git and CI/CD Integration:	3
<b>2 Code quality management</b>	<b>4</b>
2.1 Team policy for the use of generative AI	4
2.2 Guidelines for contributors	4
Coding style	4
Code reviewing	4
2.3 Code quality metrics and dashboards	5
<b>3 Continuous delivery pipeline (CI/CD)</b>	<b>6</b>
3.1 Development workflow	6
Coding workflow	6
Definition of done	6
3.2 CI/CD pipeline and tools	7
<b>4 Software testing</b>	<b>8</b>
4.1 Overall testing strategy	8
4.2 Functional testing and ATDD	8
4.3 Developer facing tests (unit, integration)	8
Unit Tests:	8
4.4 Exploratory testing	9
Manual Exploratory Testing:	9
4.5 Non-function and architecture attributes testing	9
Performance and Monitoring	9
Security:	9

# 1 Project management

## 1.1 Assigned roles

The team consists of a team coordinator, a product owner, a QA engineer, a DevOps master, and developers.

The assigned person and the responsibilities of each role are as follows:

**Team Coordinator (Filipe Sousa):** Ensure a fair distribution of tasks and that members adhere to the plan. Promote optimal collaboration within the team and proactively address any arising issues. Ensure timely delivery of the requested project outcomes.

**Product owner (Daniel Simbe):** Represents the interests of stakeholders and possesses a deep understanding of the product and the application domain, the team will turn to the Product Owner to clarify any questions about expected product features. Should be involved in accepting incremental solutions.

**QA Engineer (Gonçalo Calvo):** Responsible, in articulation with other roles, to promote the quality assurance practices and put in practice instruments to measure the quality of the deployment. Monitors that team follows agreed QA practices.

**DevOps master (Filipe Sousa):** Responsible for the (development and production) infrastructure and required configurations. Ensures that the development framework works properly. Leads the preparation of the deployment machine(s)/containers, git repository, cloud infrastructure, databases operations, etc.

**Developer (All members of the team):** Development tasks which can be tracked by monitoring the pull requests/commits in the team repository.

## 1.2 Backlog Grooming and Progress Monitoring

### Backlog Structure:

Our project backlog is organized into iterations and categorized by statuses on the board:

**IN DOING:** Tasks prioritized for the current iteration.

**IN PROCESS:** Tasks currently in progress.

**DONE:** Completed tasks.

## Git and CI/CD Integration:

**Branches:** Created in Jira/GitHub and linked to the corresponding issue.

**Pull Requests:** Associated with tasks on the board.

### Merge Criteria:

Mandatory review by another team member.

Passing automated tests (CI Pipeline Action).

quadro Sprint 4 12 Dec - 19 Dec (13 work items)		1 0 12	Complete sprint	...
<input checked="" type="checkbox"/> SCRUMDEMO-144	PayPal payments	PAYMENT WITH PAYP...	DONE ✓	GC
<input checked="" type="checkbox"/> SCRUMDEMO-158	Sonarcloud Problems		DONE ✓	PS
<input checked="" type="checkbox"/> SCRUMDEMO-151	LoginPage hashmap	USER PROFILES & AC...	DONE ✓	GC
<input checked="" type="checkbox"/> SCRUMDEMO-152	Visual UI problems		DONE ✓	PS
<input checked="" type="checkbox"/> SCRUMDEMO-167	User Rent Management Page	USER PROFILES & AC...	DONE ✓	PS
<input checked="" type="checkbox"/> SCRUMDEMO-166	Forced Logout when placing a tool to rent with a negative price	USER PROFILES & AC...	DONE ✓	PS
<input checked="" type="checkbox"/> SCRUMDEMO-177	Coding Style	CORE INFRASTRUCT...	DONE ✓	PS
<input checked="" type="checkbox"/> SCRUMDEMO-180	Improve Documentation	CORE INFRASTRUCT...	DONE ✓	PS
<input type="checkbox"/> SCRUMDEMO-195	XRAY	CORE INFRASTRUCT...	TO DO	
<input checked="" type="checkbox"/> SCRUMDEMO-157	Deploy	CORE INFRASTRUCT...	DONE ✓	PS
<input checked="" type="checkbox"/> SCRUMDEMO-201	Calendar wrong blocks		DONE ✓	PS
<input checked="" type="checkbox"/> SCRUMDEMO-200	NGINX	CORE INFRASTRUCT...	DONE ✓	PS
<input checked="" type="checkbox"/> SCRUMDEMO-170	k6 test		DONE ✓	GC

## 2 Code quality management

### 2.1 Team policy for the use of generative AI

Regarding AI, we as a team decided to use it only for bug fixes when it took us too much time in our hands to deal with, and for generating boilerplate code examples, ensuring the code is always reviewed by a team member.

### 2.2 Guidelines for contributors

#### Coding style

The code style adopted for this project is prioritized by uniformity and the consistency rule in order to keep a more readable and consistent code.

For this project, we rely on the official Google Checkstyle and complement it with a custom Checkstyle file tailored to our codebase. Together, these configurations enforce Google's style rules while allowing a small set of project-specific adjustments.

#### Tools to use:

Java - Checkstyle

React + TypeScript - ESLint + @typescript-eslint

#### Code reviewing

Using GitHub Copilot as a reviewer on push and pull requests on the main branch as a helper on code review.

Also, have at least one team member to review pushes and PR to main branch of the project.

## Scrumdemo 170 k6 test #33

Merged Goncasmager20 merged 3 commits into main from SCRUMDEMO-170-k6-test 2 hours ago

Conversation 2 Commits 3 Checks 5 Files changed 78 +14,041 -8,008

**Goncasmager20** commented 2 hours ago

This pull request introduces several improvements to the frontend, focusing on user interface consistency, accessibility, and navigation. The main changes include updating layout styles for better scroll behavior and visual alignment, enhancing modal accessibility and centering, improving calendar usability in the tool details page, and removing the "Analytics" navigation link from admin sections. These updates are supported by new and improved tests to ensure correct behavior and appearance.

**Layout and Styling Improvements:**

- Updated global and page-specific styles in `App.css`, `About.css`, and multiple component styles to use `min-height: 100vh` and proper overflow settings, ensuring pages are scrollable and visually consistent across different screen sizes. ( `frontend/src/App.css` [1] `frontend/src/pages/About.css` [2] [3] `frontend/src/pages/Catalog.tsx` [4] `frontend/src/pages/User/AddRent.tsx` [5] [6] )
- Added a test to verify the catalog page's root container is scrollable and styled correctly. ( `frontend/src/pages/Catalog.test.tsx` `frontend/src/pages/Catalog.test.tsxR345-R364` )

**Modal Accessibility and Centering:**

- Refactored the register modal in `LoginPage` to use a backdrop `div` for accessibility, keyboard navigation, and proper centering, with improved styles for padding and alignment. ( `frontend/src/pages/LoginPage.tsx` [1] [2] [3] )
- Added a test to ensure the register modal appears centered on the screen. ( `frontend/src/pages/LoginPage.test.tsx` `frontend/src/pages/LoginPage.test.tsxR127-R149` )

**Calendar Component Usability:**

- Enhanced the mini calendar in the tool details page to support month navigation, selection mode toggling between start/end dates, and display of weekday headers, improving rental date selection. ( `frontend/src/pages/User/ToolDetails.tsx` [1] [2] )

**Reviewers**

**FilipePinaSousa** ✓

**Assignees**

No one—assign yourself

**Labels**

None yet

**Projects**

None yet

**Milestone**

No milestone

**Development**

Successfully merging this pull request may close these issues.

None yet

**Notifications** Customize

Unsubscribe

You're receiving notifications because you modified the open/close state.

## 2.3 Code quality metrics and dashboards

For static code analysis, we opted to have a workflow with GitHub Actions which runs every time a push or a pull request is executed. It runs all tests, sends the code to be analyzed with SonarQube Cloud Service integrated on Github repository.

This creates issues in case the quality gate doesn't pass, and the developer must check and fix those issues. The default quality gate provided by the SonarQube is the used one with the duplicated code at 0.3% or greater and code coverage equal or greater than 80%.

**Quality Gates**

Sonar way (Default) (Built-in)

Sonar way for AI Code (Built-in)

My way

**Sonar way** (Default) (Built-in)

This quality gate reflects Sonar recommended practices.

**Conditions**

Conditions on New Code apply to all branches and to Pull Requests.

No new bugs are introduced	Reliability rating is A
No new vulnerabilities are introduced	Security rating is A
New code has limited technical debt	Maintainability rating is A
All new security hotspots are reviewed	Security Hotspots Reviewed is 100%
New code has sufficient test coverage	Coverage is greater than or equal to 80.0%
New code has limited duplications	Duplicated Lines (%) is less than or equal to 3.0%

**Projects**

Every project not specifically associated to a Quality Gate will be associated to this one by default.

## 3 Continuous delivery pipeline (CI/CD)

### 3.1 Development workflow

#### Coding workflow

Our project is organized in a single repository with the following modules:

- |— backend
- |— frontend
- |— performance
- |— docs
- |— proxy
- |— scripts

Regarding the division of branches, we do it as follows:

Main Branch: master (protected, merge only via Pull Request);

Secondary Branches: Created from backlog tasks.

Therefore, the work is done on different branches and commits are given to them. When the feature is completely implemented, it is given PR, with reviews and tests passed to the master branch.

#### Definition of done

A user story is only considered ready when it meets all of these points:

**Functionality Implemented:**

It does exactly what was requested in the task acceptance criteria.

**Code Cleaned and Reviewed:**

Another team member approved the code in the Pull Request.

It passed SonarQube analysis.

**Testing Done and Approved:**

Tests passed.

**System Integration:**

The code was merged into the main branch (master).

GitHub Actions, SonarCloud and CodeQL executed everything without errors (build, tests, deploy).

## 3.2 CI/CD pipeline and tools

Several CI/CD pipelines were implemented on GitHub Actions with the backup of SonarCloud quality tests.

For this project on GitHub it was created on a single repository it was created the following workflows:

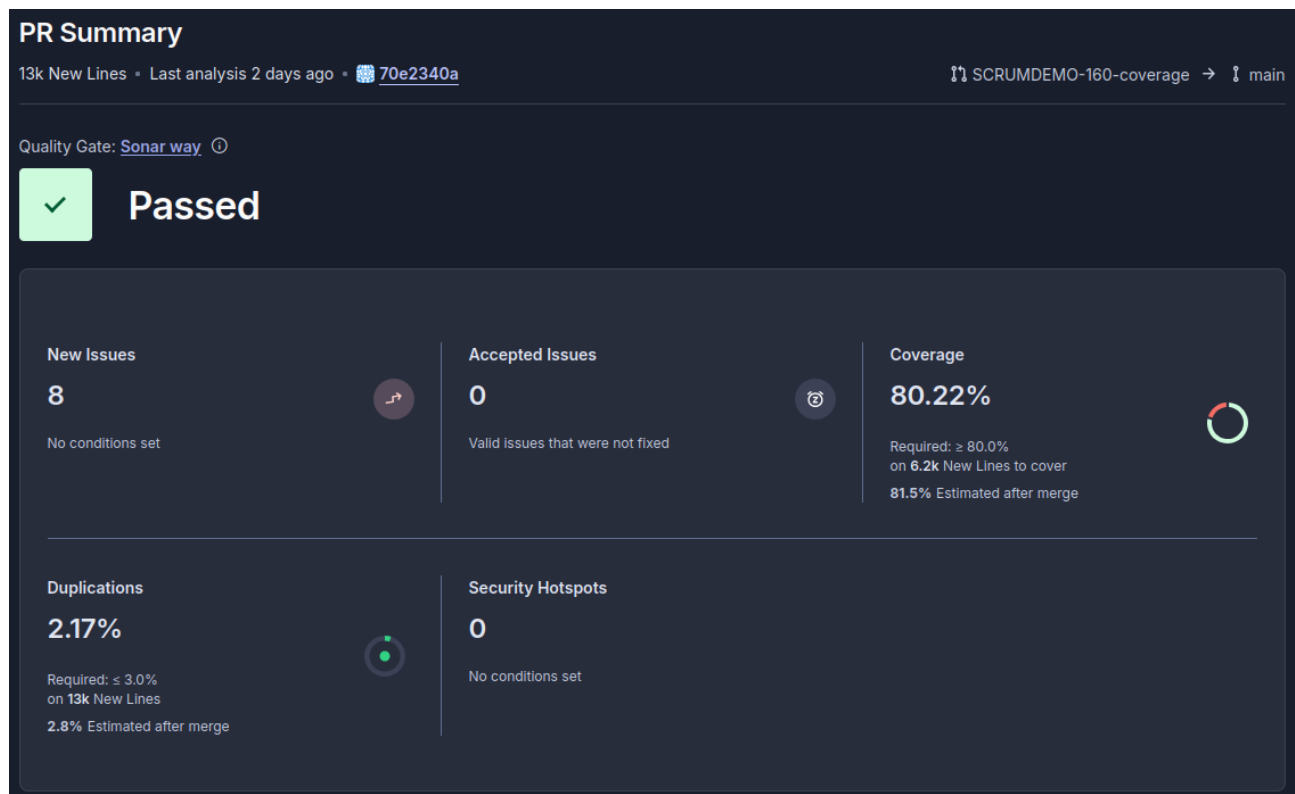
### Backend

- Unit and integration backend tests, using JUnit and Cucumber
- Upload of results to Sonar Cloud

### Frontend

- Unit/E2E tests with Vitest and Cypress
- SonarCloud analysis
- K6 performance and metrics evaluation

After tests are run, SonarCloud comment on the Pull Request with the output of those tests.



## 4 Software testing

### 4.1 Overall testing strategy

To ensure coverage and efficiency, we use a mix of TDD (Test-Driven Development) and functional tests, automation for regression and CI enforcement:

- Test suites run locally and in CI (GitHub Actions); merges are gated on passing CI tests.
- Unit tests and small fast tests are used for API and domain logic.
- Integration and functional tests verify end-to-end flows and external integrations.
- BDD (Cucumber): Used for backend functional scenarios where acceptance criteria are expressed as Gherkin.
- SonarCloud is used for static code analysis and coverage reporting.

### 4.2 Functional testing and ATDD

**Focus:** User-facing behaviors (black-box testing) and acceptance criteria described in user stories.

**When:** For each user story with clear acceptance criteria (automated where possible).

**Tools:**

Cucumber (backend functional/BDD tests).

Frontend E2E is performed with Cypress.

### 4.3 Developer facing tests (unit, integration)

**Unit Tests:**

- **Tools:** JUnit (and Spring test utilities).
- **Goal:** High coverage for business logic; coverage measured with JaCoCo.

**Integration Tests:**

- **Tools:** Spring Boot integration tests executed with Maven (Surefire/Failsafe).
- **Goal:** Validate communication between controllers, services and the database.

### 4.4 Exploratory testing

**Manual Exploratory Testing:**

- Complements automation and is performed by team members.
- **Focus on scenarios not covered by automated tests, such as:**
  - Usability;
  - Complex edge cases;
  - Compatibility checks.



## 4.5 Non-function and architecture attributes testing

### Performance and Monitoring

k6 performance test scripts locally runned

### Security:

SonarQube