TQS: Quality Assurance manual

Bruno Lopes [68264], Tiago Albuquerque [112901], Abel Teixeira [113655]
v2025-05-14

Contents

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 (Bruno Lopes): 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 (Abel Teixeira): 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 (Tiago Albuquerque): 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 (Bruno Lopes): 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.

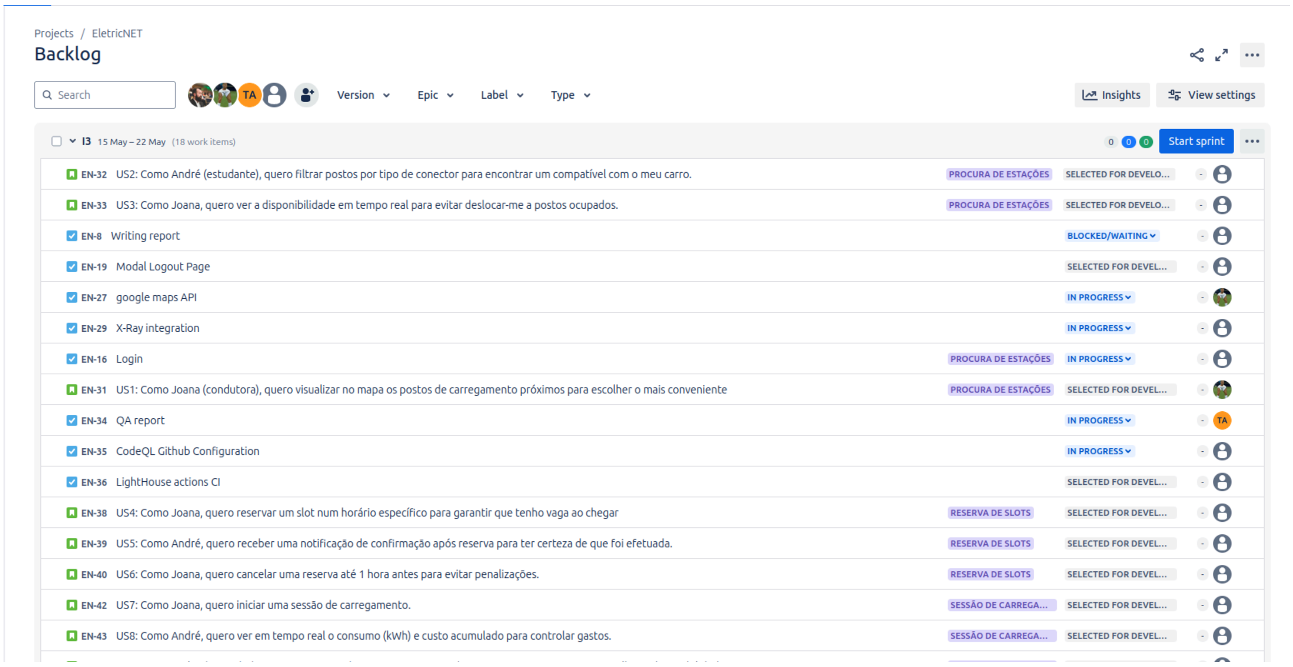
1.2 Backlog Grooming and Progress Monitoring

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

- **SELECTED FOR DEVELOPMENT:** Tasks prioritized for the current iteration.
- **BLOCKED/WRITING:** Tasks with unresolved dependencies.
- **IN PROCESS:** Tasks currently in progress.
- **IN REVIEW/TEST:** Tasks pending review by the rest of the team.
- **DONE:** Completed tasks.

Git and CI/CD Integration:

- **Branches:** Created from Jira 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).



Projects / EletricNET

Backlog

Search | Version | Epic | Label | Type | Insights | View settings

I3 15 May – 22 May (18 work items) | 0 0 0 | Start sprint

EN-32	US2: Como André (estudante), quero filtrar postos por tipo de conector para encontrar um compatível com o meu carro.	PROCURA DE ESTAÇÕES	SELECTED FOR DEVELO...	-
EN-33	US3: Como Joana, quero ver a disponibilidade em tempo real para evitar deslocar-me a postos ocupados.	PROCURA DE ESTAÇÕES	SELECTED FOR DEVELO...	-
EN-8	Writing report		BLOCKED/WAITING	-
EN-19	Modal Logout Page		SELECTED FOR DEVEL...	-
EN-27	google maps API		IN PROGRESS	-
EN-29	X-Ray integration		IN PROGRESS	-
EN-16	Login	PROCURA DE ESTAÇÕES	IN PROGRESS	-
EN-31	US1: Como Joana (condutora), quero visualizar no mapa os postos de carregamento próximos para escolher o mais conveniente	PROCURA DE ESTAÇÕES	SELECTED FOR DEVEL...	-
EN-34	QA report		IN PROGRESS	-
EN-35	CodeQL Github Configuration		IN PROGRESS	-
EN-36	LightHouse actions CI		SELECTED FOR DEVEL...	-
EN-38	US4: Como Joana, quero reservar um slot num horário específico para garantir que tenho vaga ao chegar	RESERVA DE SLOTS	SELECTED FOR DEVEL...	-
EN-39	US5: Como André, quero receber uma notificação de confirmação após reserva para ter certeza de que foi efetuada.	RESERVA DE SLOTS	SELECTED FOR DEVEL...	-
EN-40	US6: Como Joana, quero cancelar uma reserva até 1 hora antes para evitar penalizações.	RESERVA DE SLOTS	SELECTED FOR DEVEL...	-
EN-42	US7: Como Joana, quero iniciar uma sessão de carregamento.	SESSÃO DE CARREGA...	SELECTED FOR DEVEL...	-
EN-43	US8: Como André, quero ver em tempo real o consumo (kWh) e custo acumulado para controlar gastos.	SESSÃO DE CARREGA...	SELECTED FOR DEVEL...	-

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 in bug fixes when it took us too much time in our hands to deal with.

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.

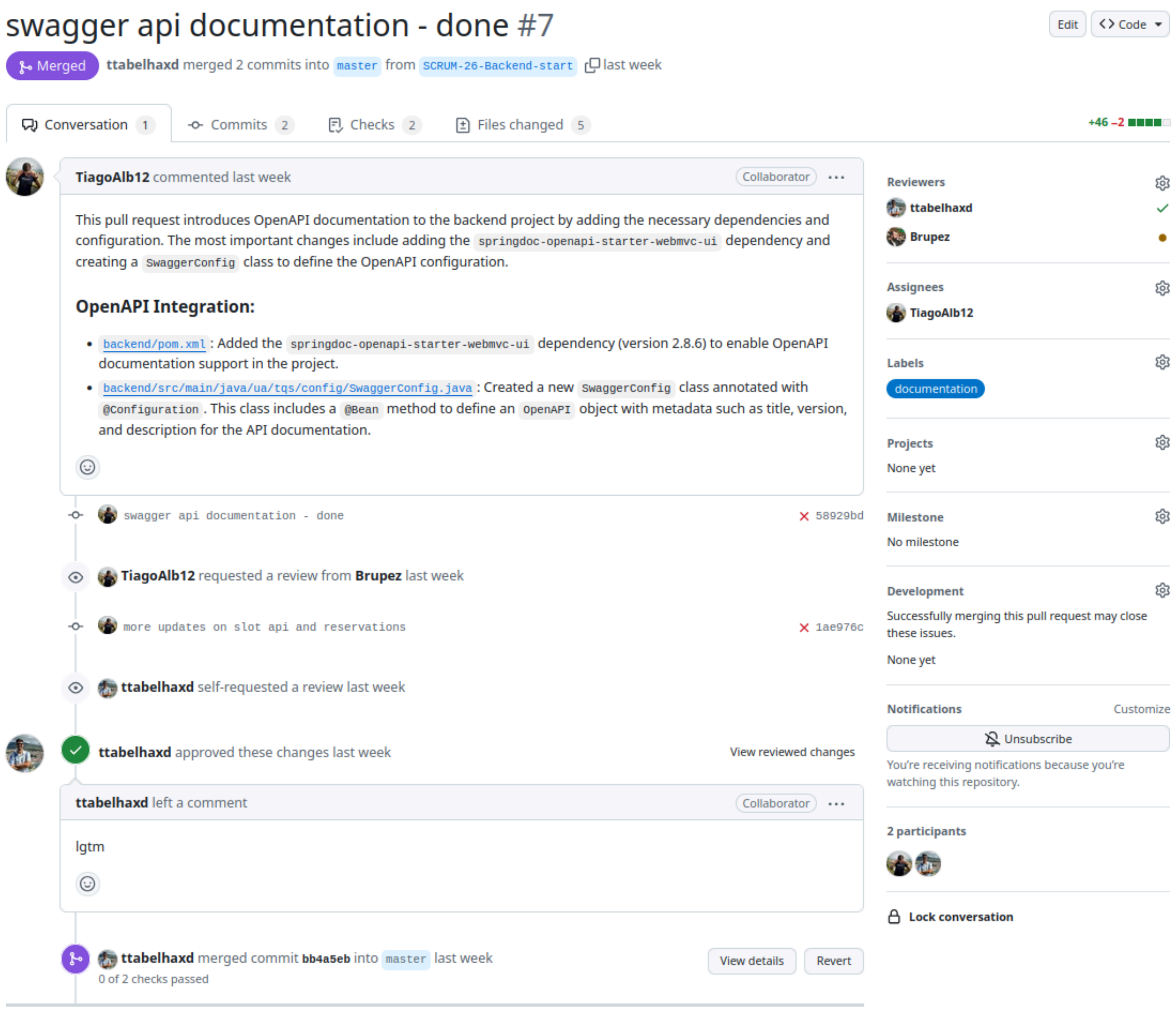
As this project was developed in Java, Typescript and Javascript programming languages after some thought we follow the **Google Style Guide**. For example, focusing on camelCase for variables and method, no line wrapping and use of optional braces on *if, else, for,..*

Tools to use:

- Java - Checkstyle
- TypeScript - ESLint + @typescript-eslint
- JavaScript - ESLint

Code reviewing

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



swagger api documentation - done #7

ttabelhaxd merged 2 commits into `master` from `SCRUM-26-Backend-start` last week

TiagoAlb12 commented last week · Collaborator

This pull request introduces OpenAPI documentation to the backend project by adding the necessary dependencies and configuration. The most important changes include adding the `springdoc-openapi-starter-webmvc-ui` dependency and creating a `SwaggerConfig` class to define the OpenAPI configuration.

OpenAPI Integration:

- `backend/pom.xml`: Added the `springdoc-openapi-starter-webmvc-ui` dependency (version 2.8.6) to enable OpenAPI documentation support in the project.
- `backend/src/main/java/ua/tqs/config/SwaggerConfig.java`: Created a new `SwaggerConfig` class annotated with `@Configuration`. This class includes a `@Bean` method to define an `OpenAPI` object with metadata such as title, version, and description for the API documentation.

`swagger api documentation - done` ✕ 58929bd

TiagoAlb12 requested a review from **Brupez** last week

`more updates on slot api and reservations` ✕ 1ae976c

ttabelhaxd self-requested a review last week

ttabelhaxd approved these changes last week · View reviewed changes

ttabelhaxd left a comment · Collaborator

lgtm

ttabelhaxd merged commit **bb4a5eb** into `master` last week

0 of 2 checks passed

Reviewers: ttabelhaxd ✓, Brupez ●

Assignees: TiagoAlb12

Labels: documentation

Projects: None yet

Milestone: No milestone

Development: Successfully merging this pull request may close these issues. None yet

Notifications: Customize

You're receiving notifications because you're watching this repository.

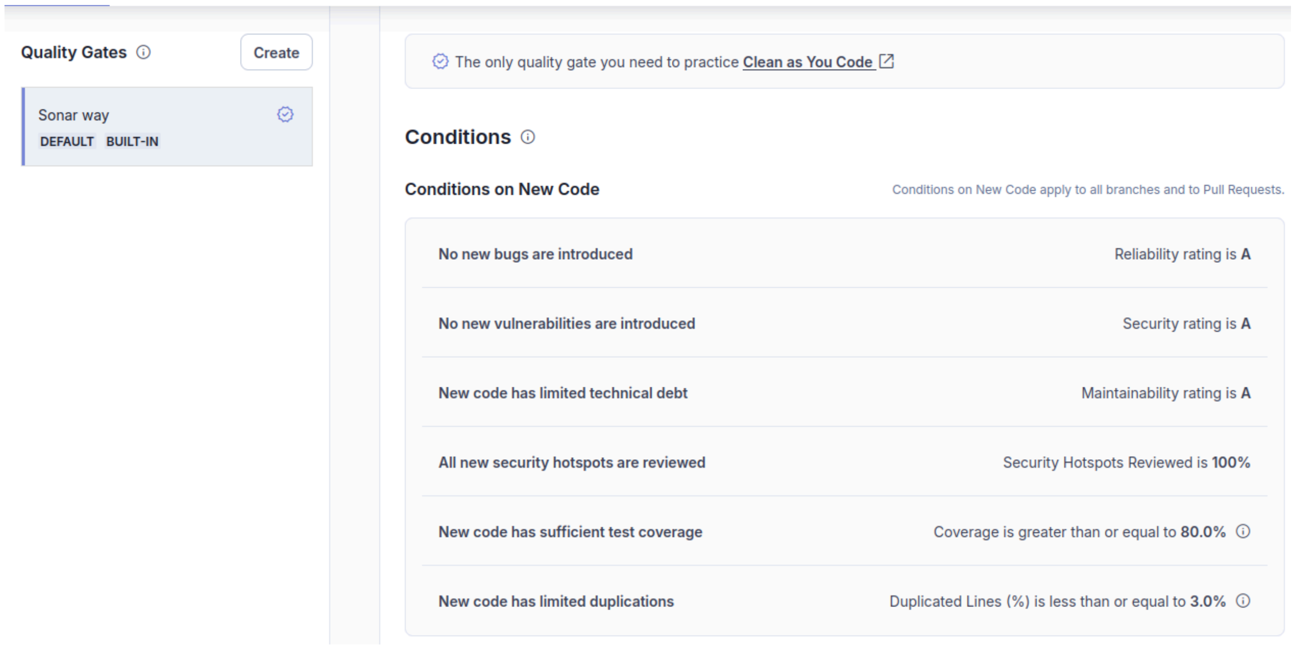
2 participants

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

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 and 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%.



The only quality gate you need to practice Clean as You Code

Conditions

Conditions on New Code

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%**

Figure 1. Sonar Quality Gate

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
├── docs
├── frontend
├── monitoring
└── proxy
```

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

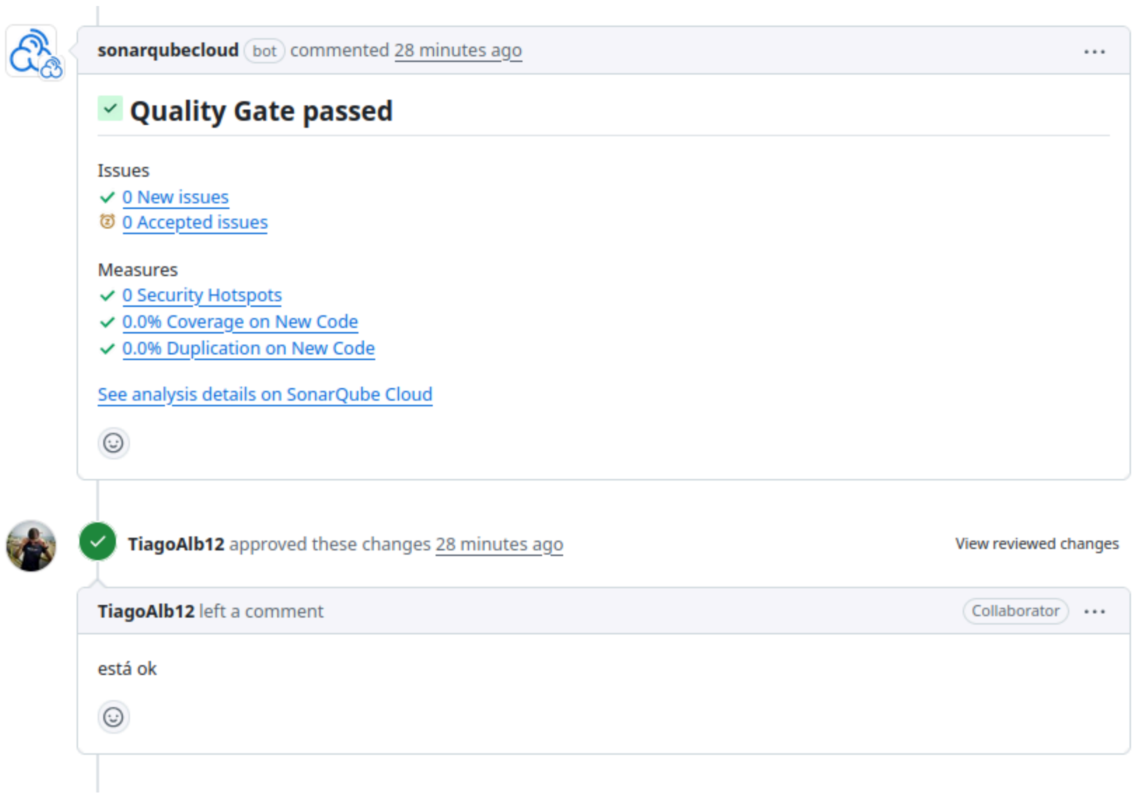
Several CI/CD pipelines were implemented on GitHub Actions with the backup of SonarCloud quality tests.

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
  - Playwright and Cucumber tests for the frontend side
  - Upload of results to Sonar Cloud (check section 2.3)
  - Upload of results to Xray, testing user stories
  - GitHub's CodeQL analysis

- Frontend
  - SonarCloud analysis
  - Lighthouse performance and metrics evaluation
  - GitHub's CodeQL analysis

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



sonarqubecloud bot commented 28 minutes ago

Quality Gate passed

Issues
✓ 0 New issues
0 Accepted issues

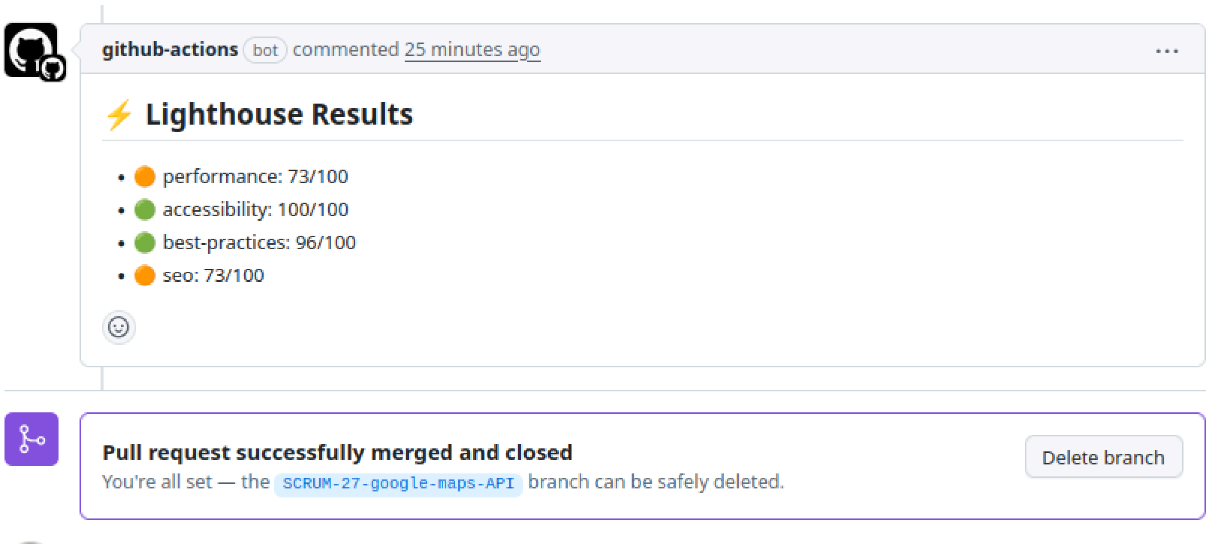
Measures
✓ 0 Security Hotspots
✓ 0.0% Coverage on New Code
✓ 0.0% Duplication on New Code

See analysis details on SonarQube Cloud

TiagoAlb12 approved these changes 28 minutes ago — View reviewed changes

TiagoAlb12 left a comment — Collaborator

está ok


github-actions bot commented 25 minutes ago

⚡ Lighthouse Results

- 🟠 performance: 73/100
- 🟢 accessibility: 100/100
- 🟢 best-practices: 96/100
- 🟠 seo: 73/100

Pull request successfully merged and closed

You're all set — the `SCRUM-27-google-maps-API` branch can be safely deleted.

3.3 System observability

Docker health checks are implemented for both the database of PostGres and Spring Boot backend, Spring Actuator, Prometheus and Grafana.

As Automatic Alerts we are notified by email of alerts in the CI/CD Pipeline.

- Health Checks:
  - Backend: Endpoint /health (Spring Actuator) checks:
    - Database connection;
    - Status of external APIs.

  - Frontend: Basic monitoring (e.g. HTTP 200 response on endpoint /).

- Automatic Alerts:
  - Docker: Notifications if containers fail;
  - CI/CD Pipeline: Alerts on Slack/Email for:
    - Failed builds;
    - Unfinished deployments.

4 Software testing

4.1 Overall testing strategy

To ensure coverage and efficiency, we adopted several strategies:

- **TDD (Test-Driven Development):** For critical components, unit tests were written before the code;
- Unit Tests;
- Integration Tests;
- **BDD with Cucumber:** Automated functional tests;
- **TestContainers** for integration with PostgreSQL;
- **Integration in CI:** All tests are executed in GitHub Actions. Merging is only allowed if they pass.

4.2 Functional testing and ATDD

- Tests focused on user behavior (black box).
- **When:** For each user story with clear acceptance criteria
- **Tools:**
  - Cucumber + Playwright (frontend);
  - Xray to link tests to USs in Jira.

4.3 Developer facing tests (unit, integration)

Unit Tests:

- Goal: Cover 80%+ of the code with JUnit.

Integration Tests:

- Goal: Validate communication between services.
- Tools: SpringBootTest + TestContainers.

4.4 Exploratory testing

Manual Testing with Real Users:

- Carried out by team members;

- Focus on scenarios not covered by automation, such as:
  - Usability;
  - Edge cases;
  - Compatibility.

4.5 Non-function and architecture attributes testing

Performance:

- Spring Actuator => Prometheus => Grafana
- Lighthouse for frontend metrics.

Security:

- SonarQube
- Github CodeQL