TQS: Quality Assurance manual

Gonçalo Lopes [107572], Rodrigo Graça [107634], Tiago Cruz [108615]

1 Project management

1.1 Team and roles

Team Coordinator: Tiago Cruz

Ensure that there is a fair distribution of tasks and that members work according to the plan. Actively promote the best collaboration in the team and take the initiative to address problems that may arise. Ensure that the requested project outcomes are delivered on ime.

QA Engineer: Gonçalo Lopes

Responsible, in articulation with other roles, to promote the quality assurance practices and put in practice instruments to measure que quality of the deployment.

Monitors that team follows agreed QA practices.

DevOps master: Rodrigo Graça

Responsible for the (development and production) infrastructure and required configurations. Ensures that the development framework works properly. Leads the preparing the deployment machine(s)/containers, git repository, cloud infrastructure, databases operations, etc.

Developer: All elements
Responsible for monitoring the pull requests/commits in the team repository.

1.2 Agile backlog management and work assignment

In the **MedConnect** project, our team employs a comprehensive Agile backlog management and work assignment process designed to optimize efficiency, adaptability, and team collaboration. The approach is fundamentally user story-oriented, ensuring that all functionality delivers tangible value to the users and aligns closely with their needs.

Backlog Management Practices

- **User Stories Creation:** All features and requirements are broken down into user stories that describe functionality from the perspective of an end-user. This helps keep the focus on delivering value directly observable by the users.
- **User Stories Grooming:** Regular backlog grooming sessions are held to refine user stories, ensuring they are clearly understood and actionable. This includes defining acceptance criteria, estimating effort, and identifying any dependencies or blockers.
- **Prioritization:** During the sprint, progress on user stories is tracked, allowing all team members to see the status of tasks in real time.

Work Assignment Practices

- **Team Roles:** Each team member has a defined role (e.g., developer, tester, UX designer) but is encouraged to take ownership of tasks across the spectrum to promote skill development and improve project understanding.

- **Task Ownership:** Tasks are not assigned by the project manager but chosen by team members themselves based on interest and expertise. This self-assignment method increases engagement and accountability.
- **Daily Stand-ups:** Daily stand-up meetings are conducted where team members discuss what they did the previous day, what they plan to do today, and any impediments they are facing. This facilitates quick resolution of blockers and keeps the team aligned.
- **Pair Programming:** For complex user stories or to foster knowledge sharing, pair programming is encouraged. This practice not only improves code quality but also helps in disseminating knowledge across the team.
- **Retrospectives:** At the end of each sprint, a retrospective meeting is held to discuss what went well, what could be improved, and how the team can adjust its practices to increase productivity and work satisfaction

Tools Used

- **JIRA:** For backlog management, sprint planning, and progress tracking. It allows for detailed reports and real-time collaboration.

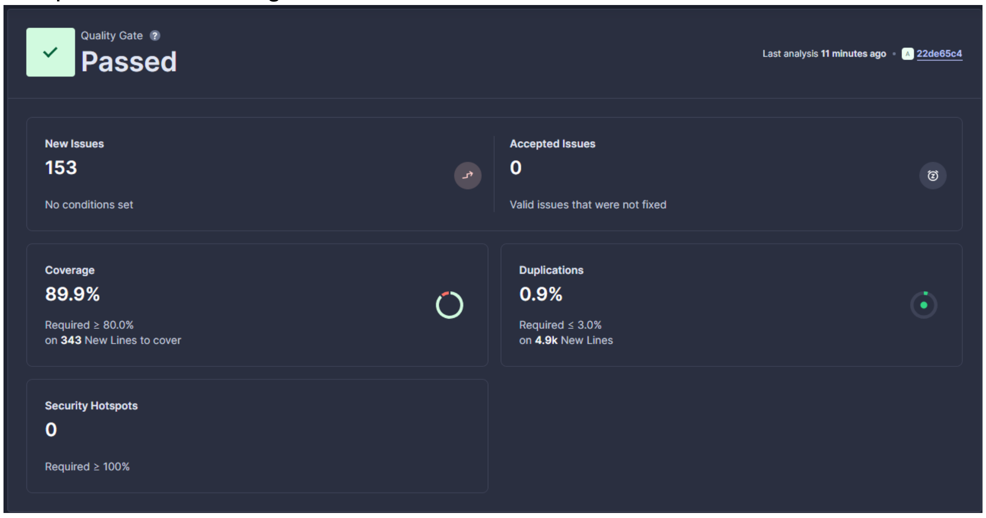
2 Code quality management

2.1 Guidelines for contributors (coding style)

Adopting the coding style guidelines of the AOS project to ensure consistency and readability across the codebase.

2.2 Code quality metrics and dashboards

Static Code Analysis: Using tools like SonarQube to perform static code analysis, identify code smells, and enforce quality gates.
Quality Gates: Defined based on the project's needs to ensure code quality, maintainability, and compliance with coding standards.



Quality Gate
Passed

Last analysis **11 minutes ago** · 22de65c4

New Issues
153
No conditions set

Accepted Issues
0
Valid issues that were not fixed

Coverage
89.9%
Required ≥ 80.0%
on **343** New Lines to cover

Duplications
0.9%
Required ≤ 3.0%
on **4.9k** New Lines

Security Hotspots
0
Required ≥ 100%

3 Continuous delivery pipeline (CI/CD)

3.1 Development workflow

- **GitFlow Workflow:** Adopted for branch management, mapping user stories to feature branches, and maintaining separate branches for development, testing, and production.

- **Code Review Practices:** Implementing mandatory code reviews via pull requests to ensure code quality and consistency.
- **Definition of Done:** A user story is considered done when it is fully implemented, tested, reviewed, and merged into the main branch.

3.2 CI/CD pipeline and tools

- **Continuous Integration:** Using Jenkins for automated builds and tests on every commit to the repository.
- **Continuous Delivery:** Using Docker for containerization and deployment, ensuring consistent environments across development, staging, and production.

3.3 System observability

- Implementing logging, monitoring, and alerting using tools like ELK Stack (Elasticsearch, Logstash, Kibana) and Prometheus to ensure system observability and quick incident response.

4 Software testing

4.1 Overall strategy for testing

In the MedConnect project, we have adopted a Test-Driven Development (TDD) methodology to ensure the quality and reliability of our software. We utilize tools such as JUnit for unit testing, Mockito for mocking dependencies, and Spring Test for integration testing. These practices help us to write tests first and then develop the code that fulfills the test requirements, ensuring robust and well-tested software components.

4.2 Functional testing/acceptance

Functional testing is performed from a user perspective using the Spring MVC Test framework. We define scenarios and acceptance criteria to validate that the implemented functionalities meet the user requirements. These tests simulate HTTP requests and verify that the system behaves as expected in real-world scenarios.

4.3 Unit tests

We use JUnit and Mockito for writing unit tests that focus on individual components of the system. Each unit of code is tested to ensure it performs as expected. Achieving high code coverage is a priority to detect potential issues early in the development process. Example tests ensure that services return expected results and controllers handle HTTP requests correctly.

4.4 System and integration testing

- **Integration Tests:** We ensure that different components of the system work together as expected using Spring Test and MockMvc. These tests validate the interactions between various modules, ensuring that they integrate seamlessly.
- **API Testing:** MockMvc is extensively used for testing our RESTful APIs. We verify that our APIs perform correctly under various conditions, including different input parameters and edge cases. These tests ensure that endpoints are correctly mapped, handle input validation, and return appropriate responses.