deti universidade de aveiro departamento de eletrónica, telecomunicações e informática

TQS: Quality Assurance manual

***Alexandra de Carvalho [934465]*, *Anthony Pereira [93016]*, *Hugo Ferreira [93093]*, *João Soares [93078]*, Wei Ye [93442]**

v2020-05-25

1 Project management

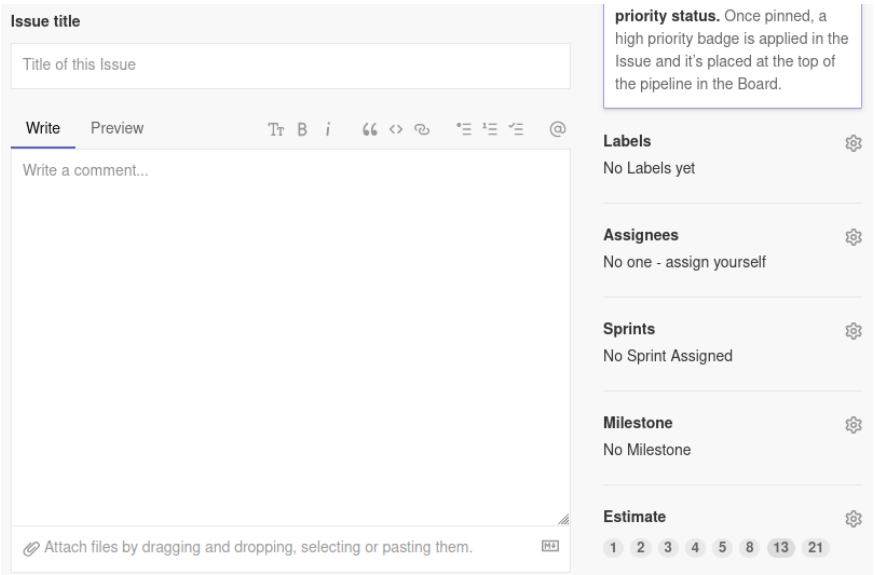
1.1 Team and roles

Role	Responsibilities
Team Coordinator João Soares	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 in due time.
Product Owner Alexandra Carvalho	Represents the interests of the stakeholders. Has a deep understanding of the product and the application domain; the team will turn into the Product Owner to clarify the questions about expected product features. Should be involved in accepting the solution increments.
QA Engineer Anthony Pereira Hugo Ferreira	Responsible, in articulation with other roles, to promote the quality assurance practices and put in practice instruments to measure the quality of the deployment.

DevOps Master Wei Ye	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 Alexandra Carvalho Anthony Pereira Hugo Ferreira João Soares Wei Ye	Responsible for all the development tasks.

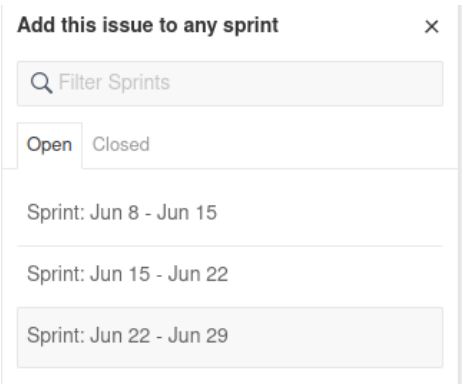
1.2 Agile backlog management and work assignment

ZenHub is being used for backlog management. This tool allows the Team Coordinator to add new issues, mapped by user stories based functionalities, as we can see in the image below.



The custom labels in use are “Backend”, “Frontend”, “ProudPapers”, “EasyDelivers” and “Rider’s App”.

As shown below, the sprints are weekly, and the estimates are made by agreement with all team members.



When an issue needs to be developed on the actual sprint, it is moved to the “Sprint Backlog” column. When an assigned developer starts to work on an issue, they pass it from the “Sprint Backlog” column to the “In Progress” column. Then, when coding and testing are finished, it goes into the “Review/QA” column” and, when it is accepted by the reviewers, it goes into the “Done” column. If it is rejected, it goes back to the “In Progress” column.

2 Code quality management

2.1 Guidelines for contributors (coding style)

For this project, we are following some common Java rules:

- exceptions are being thrown
- methods are kept relatively short
- naming conventions are being followed
- spaces are being used for indentations, to make the code clearer
- braces’ style used is the standard
- line length is tried to be kept under 100 characters
- tests’ names are understandable

```java
public Admin getAdminByEmail(String email) throws AdminNotFoundException {
    Admin user = adminRepository.findAdminByEmail(email);

    if (user == null) {
        throw new AdminNotFoundException("Admin not found.");
    }

    return user;
}
```

```java
@Test
public void whenPostValidAdmin_thenVerifyAdmin( ) throws Exception, EasyDeliversService.AdminNotFoundException {
    Admin alex = new Admin( first_name: "alex", last_name: "conte", email: "alex@deti.com", password: "pass",
            position: "CEO", description: "You can fall only if you fly");

    //given(service.save(Mockito.any())).willReturn(alex);
    when(service.getAdminByEmail("alex@deti.com")).thenReturn(alex);
    when(service.getAdminByEmail("wrong_email")).thenReturn(null);

    mvc.perform(post( urlTemplate: "/dashboard").contentType(MediaType.APPLICATION_JSON)
            .content(JsonUtil.toJson(alex))).andExpect(status().isOk())
            .andExpect(jsonPath( expression: "$.name", is( value: "alex")));

    verify(service, times( wantedNumberOfInvocations: 1)).getAdminByEmail(Mockito.any());

}
```

2.2 Code quality metrics

For static code analysis, it was used SonarCloud integrated with JaCoCo to generate reports about the code coverage. Sonar Cloud facilitated the development of tests and code Reliability. Another tool used to analyse the code was SonarLint to quickly identify and fix issues during development.

The defined Code Quality Gate was the following:

CustomQG

Conditions

Conditions on New Code

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

Metric	Operator	Value
Condition Coverage	is less than	30.0%
Duplicated Lines (%)	is greater than	3.0%
Maintainability Rating	is worse than	A
Reliability Rating	is worse than	A
Security Hotspots Reviewed	is less than	100%
Security Rating	is worse than	A

3 Continuous delivery pipeline (CI/CD)

3.1 Development workflow

Gitflow WorkFlow is being followed in this project, with each branch feature/* representing a functionality, mapped by a user story, as explained before. The review of code in a branch is usually assigned to more than one colleague so that the code is inspected thoroughly.

The team's definition of done (DOD) is when the unit tests and integration tests have been written, executed and passed.

3.2 CI/CD pipeline and tools

Github Actions is being used as a CI/CD tool. There were defined four configuration files, which were placed at the root project, inside the .github/workflows directory - one for SonarCloud, and the others to run all tests from each maven or android project.
The CI for ProudPapers, EasyDeliversAdmin and EasyDeliversMobileApp run all tests in the project repository when a pull request is made to the develop branch or whenever a push is made.
Both EasyDelivers and ProudPapers services are deployed into different and specialized docker containers, using two similar docker-compose files. While EasyDelivers uses the port 8080 to run the Java Maven project and the 3306 to run the MySQL database, ProudPapers uses respectively the ports 9000 and 3309.

4 Software testing

4.1 The overall strategy for testing

The overall test Development Strategy was TDD.

For the controllers we used WebMvcApproach, which scans only the isolated controller and mocks the rest of the app, and used Rest-Assured to behave as a headless client and access, through customized requests, the RESTful server. This allowed us to test a wide range of requests while validating their responses.
We also used TestContainers to test the behaviour of each Repository Class, and Mockito to perform unit tests for multiple service methods that need pre-determined data from a Repository.

While developing all these tests we continuously checked the SonarCloud results, which helped us maintain the quality of our code.

4.2 Functional testing/acceptance

Functional tests were developed using Selenium and JUnit5, in a closed-box paradigm.

4.3 Unit tests

Unit tests were developed using Junit5 and Mockito, in an open-box paradigm.

4.4 System and integration testing

Integration tests and API tests were developed using WebMvc and SpringBootTest.