Internal Regulations

for

Leaflings

Prepared by João Pedro Santos

Reviewed by:

Hugo Guimarães

Pedro Pinho

Sónia Oliveira

ESTG

Grupo 10

02/10/2024

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
João Santos	03/10/2024	Added methods of evaluation, changed document structure.	1.0
João Santos	25/10/2024	Added methods of evaluation, changed document structure.	2.0

1. Introduction

The purpose of this document is to serve as a guideline for group members, when they want to get information when it comes to their role, what they are to do in order to contribute to the success of the development of the Leaflings system.

In this document, each member will be able to access their role, their responsibilities, the rules for attending meetings, the rules for communication, the protocol for meetings, the penalties for missing deadlines, and the strategies on to deal with such cases (delays).

For the sake of the success of this project, each member is meant to be communicative and as transparent and honest with the group as can be.

The reason for this is so we can all work collaboratively for the completement of the system before the final submission date. Since our timeline is only 3 months, we would like to enforce to every group member that as soon as they come across a problem, they should communicate it to the group as soon as possible so others can help.

2. Group roles and responsibilities

2.1 Group member roles

João Pedro Santos - SCRUM master, team manager, DEV team member;

Hugo Guimarães – Tool administrator, Product owner, DEV team member;

Sónia Oliveira – Lead Tester and reviewer, DEV team member;

Pedro Pinho - Configuration Analyst, DEV team member;

2.2 Responsibilities of each role

DEV team members - the Dev Team is attributed to all four members. They are entrusted with the tasks of creating a shippable product increment every sprint. The dev team is the one responsible for the actual coding.

Product Owner - They have the responsibility of ensuring the project itself is on track, they are the to divide the work into tasks and prioritizing said tasks so that all objectives are met.

SCRUM master – They're tasked with ensuring that the team is following a SCRUM approach to the work to be done. They are to play a coaching role when it comes to any doubts when it comes to the SCRUM process and help with any necessary steps in said process. With this said, they are responsible for making sure that all necessary meetings happen and that beats in the drum of the SCRUM methodology are in place.

Team manager – Tasked with sending meeting notices, scheduling meetings, filling out documents regarding meetings and uploading them into the GitLab.

Lead Tester and Reviewer – They're responsible for overseeing the design and executing of tests to validate the functionality and performance of the software. the design and execution of tests for each component of the software is to be made by each member for their own assigned tasks, and the lead tester is the one to green light the test outcases and test implementations.

Tool admin – Responsible for managing tools for configuration management and tracking changes made to the project's configuration.

Configuration Analyst – Responsible for verifying that all documents are in the correct version, saved correctly and up-to-date.

3. Attendance and Participation Rules

3.1 Penalties for missing meetings or missing deadlines

As meetings are the backbone of our teamwork as a group, it's expected that there are going to be penalties for missing meetings or deadlines; Those are outlined as follows:

. Missing Meetings: Missing a **first** meeting will result in a warning if only one member missed, **if more than 1 member misses a meeting**, **what was going to be discussed in that meeting will be decided by the rest of the present member board**.

Missing a **second meeting** will result in required meeting summary submissions, a document to be **filed by the missing member and delivered to the SCRUM master**; said summary should be written based on notes of other members about the meeting, and it should summarize what was discussed and decided in the meeting. Moreover, members that have **missed two or more meetings will see their power in decision making reduced, and will have deductions in internal evaluation.**

3.2 Procedures for handling delays

. Early Detection and Reporting: Team members should promptly report any foreseeable delays in their tasks to the group. This should be done during daily meetings or through designated communication channels (discord).

. Task Reallocation: If a team member is facing delays, tasks can be reallocated to others with more availability or who have finished their own work. This ensures work continues without significant impact on the timeline.

. The project timeline was thought with the possibility of delays being bound to happen, for that reason we have dedicated 1 sprint dedicated solely to solving possible delays. If no delays have happened until that point in the project, that sprint will then be replanned into anticipating work.

4. Meeting Protocol

Before each meeting, a member will receive a meeting notice sent by the **team manager** through email; They will also be notified through discord.

Every meeting notice will be sent 12 hours before the meeting at a minimum. If a notice is sent **after the 12-hour mark**, the **penalties for missing a meeting lined out in section 3.1 no longer apply**, and the team member will have points deducted in the internal evaluation.

5. Communication and Documentation

5.1 Official communication channels

The main communication channel used by the group will be discord. Although it's an informal communication channel, it's a platform every member has vast amounts of experience using, and therefore is considered to be the best option for communication between members for this project as it prevents members from having to adapt to another platform in short notice.

There is a discord server where every member is in, and in said server that are channels for holding meetings, delivering documents for consideration and peer review, general discussion, reporting delays and problems that may come up.

5.2 Approval for documents

A document is only considered to be approved for delivery after every member of the group has read it, and given their signed approval.

6. Sprint Planning

Specify deadlines for submitting work, signing off on meeting minutes, and completing tasks. Indicate how deadlines will be communicated and what happens if they are missed.

. Sprint outlines: We will be holding weekly sprints; Each sprint is to follow the following schedule:

. Sunday to Wednesday – These days of the sprint period, each member is to take their assigned user stories and implement them in the respective branch.

. Wednesday to Saturday – Wednesday is the last day dedicated to implementing, it is also the day where the team manager will distribute the test by each member. The test will be distributed from member who implemented to member who implemented (e.g. what Hugo implemented, João tests, and what João implemented, Hugo will test). As soon as all tests are done, members are to report any corrections necessary to their test peer.

. Saturday and Sunday – by the end of Saturday members should have their code and tests documented and ready to be delivered. Sunday the team will hold a sprint retrospective and will assign new user stories to members.

7. Technical Strategies

7.1 Branching

In the project, we'll use an organized structure of branches to ease the process of development and collaboration between the team. Branches are to follow the the following naming conventions:

Main Branches:

- **Master**: Dedicated for stable and ready to deliver code, containing the most recente version of the project.

- **Dev**: Used to integrate new features and functionalities or updates, receiving merges from support branches. This branch is to be tested to a deep level before allowing any merges to the Master Branch.

Support Branches:

Support branches are temporary and used for the development of features or specific corrections to code. Each member of the team works in an independent support branch, so that multiple aspects of the system can be developed simultaneously. Each support branch is supposed to be deleted (**with permission of the other 3 group members**) after a merge to Dev.

- **Naming Conventions:**

For Support Branches:

1. **Category Prefix**: Nature of the work. The allowing prefixes are the allowed ones:
   - ***feat/:*** To develop new features.
   - ***bug/:*** For correcting bugs.
   - ***doc/:*** For alteration in documentation of code.
   - ***junk/*:** Temporary or experimental code.
   - ***wip/*:** For features still in progress ("Work in Progress").

2. **Descriptive name**: Following the prefix, add a short and clear name for the point of the branch. Use Kebab case (lowcase-separated-through-hifens).

Examples of branch names:
- feat/add-user-authentication
- bug/fix-login-issue
- doc/update-readme
- wip/ui-improvements

7.2 Merge Requests

To maintain efficient collaboration and guaranteeing quality code, every merge request in the project is to follow the following instructions. These rules aim to patternize the processes of revision and ease the integration of code changes into code deliverable code.

General rules:

- **Each MR has to have a clear purpose**: Only one functionality, Correction or change per MR. Multiple changes are to be separated for multiple MRs.

- **Detailed Description**:
  - The title of a MR must contain the branch name that it comes from.
  - Include in the description:
    - **What was changed**: Explain what changes or features were added.
    - **Motive**: The problem that the change solves, or the new feature's functionality towards the system's purpose.
    - **Test Instructions**: If applicable, guidelines for testing.

- **Reference issues (when possible)**: If the MR solves an existing issue, reference the corresponding issue, using the following syntax : **Solves#<Issue-Number>;**

- **Before MR**: Please revise the code before a MR, make sure that you avoid:
  - Obvious code errors, or depricated code.
  - Tests are included and passed.

- **Peer Review**: The author of the MR must have at least one peer review their code, ideally someone with more experience in the code area;

- **Conflict resolution**: In case of conflicts with the Dev branch, the MR author is responsible for solving said conflicts before soliciting peer review.

- **Where to make MRs:** Merge requests are to be done, from support branches to other support branches, or from support branches to the dev branch. **The only branch that can make a MR to the Master branch is the Dev branch.**

Review Process:

- **Detailed Review**: The reviewer has to make sure that code is correct, follows good practices and doesn't introduce bugs, to the best of their ability.

- **Approval**: A merge request can only be approved after going through review and every criticism being solved. Merge requests that alter the System in a big manner might need more than one member's approval.

Title and Description Example for a Merge Request:

- **Title**: **feat/add-user-authentication**

- **Description**:

    - **What was changed:** Adicionada a funcionalidade de autenticação de utilizadores utilizando JWT.
    - **Motive:** Implementar uma camada de segurança que permita o login de utilizadores e proteja rotas específicas da aplicação.
    - **Test Instructions:** Testar o login e verificação de tokens JWT utilizando a rota /login.
    - **Solves** #15

7.3 Commits

To maintain legibility and organization of the repository, it is essential that commits are done in a homogeneous and well explained manner. For this project, we will be complying with the following rules for commits:

- **Prefixes:** Each commit must start with a prefix that categorizes its nature. The allowed prefixes are as follows:
    - ***Feat* –** for new features/functionalities.
    - ***Fix* –** for bug fixing.
    - ***Doc s-*** For altering or adding documentation.
    - ***Refactor -*** For code refactoring to improve code without changing its behavior.
    - ***Test* –** Adding or modifying tests.
    - ***Style* –** Correcting good coding patterns in case necessary.

- **Specify the targeted code:** after the prefix, within parenthisis, write in what part of the system your code works for. This makes it easier and quicker to identify where changes are happening. Area exemples are as follows: API, WebClient, MobileClient, DataBase, etc.
    - Exemplo:
        - If implemented a new feature in the authentication API: *feat(API)-add-jwt-authentication*
        - If corrected a bug in the client login: *fix(Web)-correct-login-error*

- **Small and simple description:** After the previous rules, include a short description, ideally with no more than 50 characters, make it clear and concise. In case of correction of an issue, a commit description can be a simple: #<issue-number>
- **Small and Frequent Commits:** commits should be small and only address one specific task. Avoid stacking multiple alterations that are not related in one commit.
- **Write in presente tense: Commit meesages should be written in the present; e,g " add authentitacion " instead of " I added authentication in this commit ".**

7.4 Assigning user stories

User stories assignment is to be done during Sunday sprint planning meetings; During these meetings, user stories will be distributed between members in a collaborative manner. The assigning is to be agreed upon by every team member **present at the meeting.**

8. Methods of evaluation

We Will be following a very simple method of evaluating each team member.

Each group member is supposed to keep an excel document to be given out by the team manager where they will note anything they consider to be important to the evaluation.

8.1 For Attendance

Out of the 20 points each student can have, 5 will be dedicated to complete attendance of all meetings. **Missing a meeting unjustifiably** will result in losing 1 point, losing a second meeting will result in losing 2 points, and losing a third meeting will result in losing all 5 points dedicated to this portion of the grade.

8.2 For Consistency

Out of the 20 points each student can have, 5 will be dedicated to the consistency of their deliveries. Missing a deadline, or having your deliveries considered not good enough by the group will result in losing 1 point. Doing it a second time will result in losing 2 points, and likewise, doing it a third time will result in getting a total of 0 points in this category.

8.3 For Personal Evaluation

Out of the 10 points dedicated to the other fields, each member can grade their group mates from a 0 to 10 when it comes to the project and their efforts. The final grade will be an average of each's group member grade given to other members.