LEI\_LDS2122\_GRUPO2

Configuration Management Plan

Versão 1.6

**14/01/2022**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Versão** | **Descrição** | **Autor(s)** |
| 27/10/2021 | 1.0 | Plano SCM base | José Fernandes  Diogo Pinto |
| 01/11/2021 | 1.1 | Alteração de quem aceita MR;  Adição de labels novas para requisitos | José Fernandes |
| 06/11/2021 | 1.2 | Alterações nos horários das reuniões  Alteração do Code Reviwer | José Fernandes |
| 12/11/2021 | 1.3 | Pequenas correções  Adicionadas Tags aos métodos de identificação  Os commits podem conter várias ações  Os commits para a main podem ser feitos para algo que o PO aprove  Melhoria de alguns tópicos do ponto 4. Metodologia | José Fernandes |
| 16/11/2021 | 1.4 | Adição de aprovers nos MR (nova funcionalidade GitLab) | José Fernandes |
| 02/12/2021 | 1.5 | Adição do role “Development Supervisor”  Reformulação das responsabilidades da DevTeam  Reformulação de como se fazem MR  Adição de novas labels  Adição de novas ferramentas | José Fernandes |
| 02/12/2021 | 1.5.1 | Descrição da label Code Review Rejected corrigida | José Fernandes |
| 14/01/2022 | 1.6 | Remoção de um passo acerca da DevTeam, adição do mesmo passo no PO e remoção de informação acerca da forma como é feita a revisão (Ponto 2).  Alteração da informação acerca da nomenclatura dos commits, remoção sobre a descrição dos MR, remoção de informação das versões do projeto, pequenas correções no commits do ponto 3.1.2, remoção do ponto acerca dos testes, pequenas alterações acerca da aceitação de MR e remoção do ponto sobre versões do projeto relacionadas com os hotFix’s e alteração acerca das release’s(Ponto 3)  Adição do recurso Docker nos recursos (Ponto 5) | Cristiano Rocha |

Conteúdo

[1. Introdução 4](#_Toc89378565)

[1.1 Objetivo 4](#_Toc89378566)

[1.2 Alcance 4](#_Toc89378567)

[1.3 Definições, Acrónimos e abreviações 4](#_Toc89378568)

[2. Software Configuration Management 5](#_Toc89378569)

[2.1 Organização, Responsabilidades e interfaces 5](#_Toc89378570)

[3. The Configuration Management Program 6](#_Toc89378571)

[3.1 Identificação da Configuração 6](#_Toc89378572)

[3.1.1 Métodos de Identificação 6](#_Toc89378573)

[3.1.2 Como fazer: 8](#_Toc89378574)

[3.1.3 Labels 10](#_Toc89378575)

[3.2 Configuration and Change Control 11](#_Toc89378576)

[3.2.1 Processamento e aprovação de Change Requests 11](#_Toc89378577)

[3.2.2 Change Control Board (CCB) 11](#_Toc89378578)

[3.3 Configuration Status Accounting 11](#_Toc89378579)

[3.3.1 Project Media Storage and Release Process 11](#_Toc89378580)

[3.3.2 Reports e Audits 11](#_Toc89378581)

[4. Metodologia 12](#_Toc89378582)

[5. Recursos e ferramentas 14](#_Toc89378583)

Configuration Management Plan

# Introdução

## Objetivo

O objetivo deste projeto consiste na criação de uma aplicação web turística que gera itinerários para visitar pontos de interesse numa determinada localidade.

A aplicação pretende facilitar a visita a uma localidade a qualquer pessoa que o pretenda gerando um itinerário automático altamente personalizável, este tem o objetivo de mostrar o maior número de pontos de interesse da cidade, ajudando a divulgar os negócios e a cultura local através, também, da sugestão de locais de refeição, repouso e lazer. Também é possível a criação de itinerários totalmente personalizáveis com o objetivo de planear viagens futuras.

## Alcance

Este documento é essencial para todo o desenvolvimento e planeamento do projeto uma vez que contém as regras a seguir para um trabalho de equipa organizado e uniforme, afetando não só o código em si, mas também todo o aspeto ágil do planeamento.

A metodologia utilizada é o SCRUM.

## Definições, Acrónimos e abreviações

**SCM** – Software Configuration Management

**DevTeam** – Equipa de desenvolvimento

**CCB** – Change control Border

**PO** – Product Owner

**CR** – Change Request

**MR** – Merge Request

**SM –** Scrum Master

**DS** – Development Supervisor

# Software Configuration Management

## Organização, Responsabilidades e interfaces

**SCM Role:** DevTeam

**Atores:** Catarina Araújo, Cristiano Rocha, Diogo Pinto, Iuri Soares, José Fernandes, Rui Campos.

**Responsabilidades:**

* **Desenvolver o código**

1. Implementar e resolver o que é relatado nas issues;
2. Testar o código desenvolvido;
3. Dar assign a quem for rever a issue depois de a ter implementado e testado.
4. Rever as issues em que foi referenciado para tal.
5. Caso tenha sido referenciado para rever uma issue, fazer “*merge requests*” para a branch development depois de rever o trabalho desenvolvido.
6. Se rever código de alguém tirar-se o assign da issue.

**Nota**: Cada membro revê o código de outro membro da equipa.

**SCM Role:** SCRUM Master

**Ator:** José Fernandes

**Responsabilidades:** Gerir DevTeam

* **Gerir reuniões**

1. Organizar as reuniões de planeamento, revisões de sprint e retrospetivas;
2. Convocar e facilitar a reunião diária;
3. Gerir o comportamento da **DevTeam** nas reuniões;
4. Fazer relatórios das reuniões e atualizar a descrição do sprint.

* **Ajudar a equipa a seguir um desenvolvimento ágil**

1. Proteger a equipa de interrupções durante o sprint;
2. Ajudar a superar obstáculos;

**SCM Role:** Development Supervisor

**Ator:** Iuri Soares

**Responsabilidades:** Manter a integridade da branch “development”

* **Aceitar merge requests para a branch development** 
  1. Analisar se o merge request não põe em causa a integridade da branch “development”.
  2. Verificar se todas as pipelines passaram.

**Nota**: Este role apenas existe para issues de desenvolvimento de código frontend ou backend.

**SCM Role:** Product Owner

**Ator:** Catarina Araújo

**Responsabilidades:** Supervisão

* 1. Gerir produtos de backlog;
  2. Atribuir issues;
  3. Referenciar alguém para rever as issues resolvidas na descrição da issue.
  4. Priorizar tarefas;
  5. Avaliar progresso do projeto;
  6. Estabelecer baselines;
  7. Aceita **MR** para a branch **”main”**;
  8. Gere os change requests;
  9. Fechar User Stories;
  10. Fazer pequenas alterações antes de fechar o sprint caso ache necessário (por exemplo renomear um ficheiro, corrigir erros ortográficos, etc.)
  11. Aceita **MR** em fase pré desenvolvimento. (Especificação de requisitos / use cases / mockups).

# The Configuration Management Program

## Identificação da Configuração

### **Métodos de Identificação**

* **Métodos:**
* Todo o método devem usar a nomenclatura camelCase (ex: getVolume())
* **Commits:**
* Se for criado um método/classe o título do commit é recomendado ser “create method getVolume ”. Caso haja uma alteração a um método/classe “update method getVolume”. Caso algo seja eliminado o título deve de ser “delete method getVolume”. Caso tenha sido documentado algo o título deve de ser “Document method getVolume”. Caso sejam implementados testes o título deve de ser “Test method getVolume”. Apesar de se recomendar o uso das nomenclaturas anteriormente referidas, cabe a cada ator dos commits nomear o mesmo segundo o que sente ser mais adequado.
* Os commits podem conter várias ações desde que estas sejam especificadas corretamente na descrição do commit.
* Caso alguém se engane ou esqueça de colocar algo na descrição do commit pode usar a funcionalidade de comentar commits para tratar da situação.
* A descrição do commit, sempre que se achar necessário, deve conter detalhes sobre o que foi alterado, por exemplo eventuais erros que foram corrigidos ou um novo comportamento do método.
* **Branches:**
* A branch principal deve se chamar **”main”**;
* Existe uma branch secundária que deriva diretamente da branch **”main”** esta branch deverá chamar-se **“development”**;
* Podem existir outras branches secundárias que são denominadas **“hotFix”** (Problema urgente encontrado).
* Todas as outras branches criadas devem de ser criadas a partir da branch **“development”** e são utilizadas para resolver issues, o seu nome deverá de retratar com clareza a issue em questão (“Issue#**X**”, sendo “**X”** o número da issue) .
* **Merge requests:**
* Os merge requests devem ter um título com este formato **“Finalização da Issue #X**”, sendo “**X”** o número da issue tratada.
* **Sprints:**
* Os sprints têm o nome **“Sprint 1”**, **“Sprint 2”** e assim sucessivamente.
* A descrição do sprint deve conter o objetivo e os relatórios das reuniões de planeamento de sprint, de revisão e de retrospetiva.
* **Atas de reunião:**
  + As atas devem conter:
    - Um número identificador;
    - A data de início e fim;
    - Quem esteve presente;
    - O que foi tratado na reunião;
    - O local da reunião;
    - Um responsável por escrevê-la
    - Assinatura do PO
* **Tags:**
  + A cada Milestone é criada uma tag com o nome **“MilestoneX”**, sendo **“X”** o número da milestone.

### **Como fazer**:

* **Commits:**
* Os commits apenas são feitos em branches terciárias à **”main”** ou nas branches **“hotFix”**;
* Os commits preferencialmente representam mudanças em apenas uma classe ou método, mas pode ser optado por fazer várias alterações dentro do mesmo commit.
* Um commit para branches terciárias pode ser feito nas seguintes situações:

1. Quando um método é implementado;
2. Quando um método é testado (todos os testes são feitos);
3. Quando alguma documentação é feita;
4. Quando alterações são feitas e existe a necessidade de dar commit (por exemplo alteração de um método previamente criado).

* Nas branches **“hotFix”** os commits podem ser feitos apenas quando a alteração estiver completa.
* Podem ser feitos na branch **”main”** apenas para atualizações do plano SCM, atualização da pipeline ou outras situações que sejam justificáveis e aprovadas pelo **PO**.
* **Branches:**
* Existem vários tipos de branches:
* Branch principal **”main”** para a qual é feito um merge request (a partir da branch development) no final de cada sprint, este merge request é efetuado pelo **DS** e aceite pelo **PO**;
* Branch secundária **“development”** onde os sprints são desenvolvidos, é a partir desta branch que se fazem os **MR** para a branch **”main”** no final de cada sprint;
* Branches terciárias onde cada issue será resolvida, esta branch deriva da branch **“development”** e qualquer membro da **DevTeam** que reveja o código de outro membro da equipa pode fazer **MR** para a branch **“development”** sendo este **MR** aceite ou recusado pelo **Development Supervisor**. Sendo esta uma das poucas branches que se fazem commits, eles não são fundidos (não há squash dos commits) quando se faz o **MR**.
* As branches HotFix servem para corrigir erros urgentes detetados na branch **”main”** que não podem esperar para serem resolvidos no próximo sprint.
* **Merge request**
* Um **MR** para a branch **“development”**, deve apenas ser feito quando a issue estiver implementada e testada, pela pessoa que fizer a revisão do código de outro membro da equipa.
* Pode ocasionalmente ser feito um **MR** para ter um programa funcional, este merge request não precisa de finalizar uma issue e normalmente é feito na fase final do sprint onde algo precisa de estar funcional para mostrar o que foi desenvolvido mesmo estando incompleto.
* Os **MR** devem especificar quem está responsável por aceitar o **MR** (assignee).
* Em fase pré-desenvolvimento o responsável por aceitar e aprovar o **MR** é o **PO**, apenas deve ser necessária uma aprovação.
* **MR** de issues relacionadas com o desenvolvimento de código devem de ser aceites pelo **Development Supervisor.**
* **Aceitar MR:**
* Em fase pré-desenvolvimento todos os **MR** são aceites pelo **PO**, ou seja, enquanto se está a fazer a especificação de requisitos, use cases e a elaboração de mockups.
* Os merge requests para a branch **“development”** (em fase de desenvolvimento) são aceites pelo **DS**. Antes de aceitar o **DS** analisa o resultado da pipeline, verificando se o merge request não trará nenhum conflito com o projeto já desenvolvido. Cabe ao **DS** interagir com a devTeam de forma a resolver conflitos ou problemas que ele detete ao fazer a revisão. Caso o **MR** seja recusado é atribuída à issue a label ***“Review Rejected”*** junto com uma descrição a explicar a rejeição, cabe ao desenvolvedor retirar esta label e colocar a issue com a label correta (Plan/Doing/Testing) para resolver o problema.
* Os merge requests para a branch **”main”** são feitos pelo **DS** e aceites pelo **PO** que deverá tomar os devidos cuidados para assegurar a integridade desta branch, devendo analisar o resultado da pipeline e o impacto do merge request no projeto. Nesta fase o **PO** deve fechar as **User Stories** que foram implementadas.
* Ninguém pode aceitar os seus próprios **MR** (além do **DS**), há sempre alguém que irá estar encarregue desta função.
* A cada **MR** para a branch **“development”** a branch de origem deve ser eliminada.
* **Atas de reunião:**
  + Cada reunião deve gerar uma ata escrita e/ou aprovada pelo **PO**.
  + As atas de planeamento do sprint são escritas pelo **PO**.
  + As atas de revisão e retrospetiva são escritas pelo **SM**.

### **Labels**

* Dev::Backlog – Backlog do sprint.
* Dev::Plan – Fase pré-desenvolvimento em que a issue está a ser planeada pelo developer. A descrição da issue deve de ser atualizada nesta fase.
* Dev:Doing – Fase de implementação.
* Dev::Testing – Fase de testes ao código implementado.
* Dev::Reviewing – Fase de revisão da implementação da issue.
* Dev::Done – Fase em que o merge request é feito, a issue deve ser revista pelo CDR.
* Dev::Review Rejected - Significa que o merge request proveniente da issue foi recusado.
* Dev::Code Review Rejected - Significa que o código não passou na revisão.
* Low Priority – Prioridade de desenvolvimento baixa.
* Critical – Prioridade de desenvolvimento alta (tem de ficar feito naquele sprint).

(caso não seja assumida nenhuma prioridade, considera-se uma prioridade intermédia).

* Request::New Request – Atribuida a um novo **CR**.
* Request::Analysis – Análise do **CR**.
* Request::Rejected – Atribuída quando um **CR** é rejeitado.
* Request::Approved – Atribuída quando um **CR** é aprovado.
* User Story – Identifica user stories.
* User Storie::Ready – Identifica user stories que estão bem definidas.
* Bug – Label para identificar bugs.
* Fix – Label para identificar issues com o objetivo de corrigir algo.
* Hotifx – Label para identificar algo que tem que ser corrigido urgentemente.

## Configuration and Change Control

### Processamento e aprovação de Change Requests

* Qualquer membro da equipa pode fazer um **CR**, este deverá ser criado através de uma nova issue com uma label específica denominada “Request::New Request”;
* O **PO** está encarregue de aprovar/recusar os pedidos, caso seja aprovado, após a análise do pedido são empregues os meios necessários para efetuar as alterações necessárias, podendo até ser preciso recolher novas User Stories.

### Change Control Board (CCB)

* É neste quadro que os Change Requests são geridos.

## Configuration Status Accounting

### Project Media Storage and Release Process

* Caso seja necessário voltar atrás no projeto, deve-se recuar até à última versão estável do programa que vá de encontro ao problema. Este processo de “rollback” é feito através do repositório Git presente no GitLab.
* As releases acontecem no final de cada milestone.

### Reports e Audits

* Os *Reports* são feitos utilizando o gitLab no final de cada sprint, neste reports a equipa analisa as user stories e as issues que foram implementadas, aquilo que ficou por implementar e o que retornou para o backlog.
* Estes reports são feitos no final do sprint, todas as considerações retiradas deste report irão ser analisadas e transmitidas para uma parte da descrição do sprint que retrata a reunião de revisão de sprint.

# Metodologia

A metodologia usada é o SCRUM, a nossa metodologia define-se por:

* **Fase Inicial:**
  + Criar um repositório vazio;
  + Criar dois *Boards*, um para change requests e outro para desenvolvimento;
  + Criar labels que serão atribuídas às issues;
  + Definir Requisitos, use cases e mockups;
  + Definir ferramentas e recursos;

Nota: Esta fase tem a duração estimada de 3 semanas, durante estas duas semanas existem várias reuniões, formais ou informais para perceber e definir o que é preciso fazer e como fazer.

* **Fase de desenvolvimento:**
  + É feita uma reunião de sprint na qual uma ou mais user stories são selecionadas e divididas em várias issues;
  + As issues são atribuídas a diferentes membros da equipa;
  + As issues vão circulando pelo *Board* de desenvolvimento e sendo fechadas à medida que são implementadas, testadas e o merge request é aceite.
* **Fase pós desenvolvimento:**
  + A equipa reúne-se uma primeira vez para perceber o que aconteceu durante o sprint, tomando nota daquilo que ficou feito e do que ficou por fazer;
  + Uma segunda reunião é feita para apurar o que podia ter sido feito para melhorar a metodologia utilizada, nesta fase é normal haver a alterações ao **SCM** plan.
  + Um **MR** é feito para a branch **”main”** assim que o **PO** achar oportuno, toda a equipa pode estar presente nesta situação, mas apenas o **PO** interage com o merge request.
* **Milestones:**
  + Existirão 3 Milestones;
  + Os objetivos de cada Milestone estão descritos no enunciado fornecido.
* **Horário dos sprints**
  + Até ao dia 12/11/2021 os Sprints são de sexta-feira a sexta-feira, começa na aula laboratorial e acaba às 0:00 da sexta-feira seguinte.
  + O sprint de dia 12/11/2021 começa na aula prática desse dia, mas acaba na quarta-feira seguinte às 0:00.
  + A partir do dia 17/11/2021 todos os sprints são de quarta-feira a quarta-feira, começando depois da reunião de planeamento de sprint e acabando às 0:00 da quarta-feira seguinte.
* **Sprint Planning meeting:**
  + O objetivo do sprint é definido e selecionadas user stories;
  + As user stories são divididas em issues (tarefas mais pequenas) e atribuídas aos elementos da equipa;
  + O backlog do sprint fica definido e é o ponto de partida da equipa.
* **Daily Scrum meeting:**
* Esta reunião é informal, mas muito importante, sendo feita preferencialmente de pé, mas podendo também ser feita através de vídeo chamada;
* O objetivo é manter a equipa focada e atualizada sobre o que está a acontecer durante o sprint;
* Cada desenvolvedor explica brevemente o que fez e os problemas que encontrou;
* Não é uma reunião de extrema importância, se houver um dia que não se faça não há um grande problema, mas convém fazer uma todos os dias.
* Tem uma duração de aproximadamente 15 minutos.
* **Sprint Review meeting:**
* Todo o interveniente no sprint é obrigado a estar presente nesta reunião que demora cerca de meia a uma hora;
* No último dia do sprint ou no dia seguinte é feita uma revisão a tudo o que foi feito e ficou por fazer, ficando tudo documentado na descrição do sprint;
* São tomadas decisões sobre próximos sprints;
* É especificado se o objetivo foi ou não cumprido, podendo ser alterado o curso do projeto se necessário;
* Esta reunião pode originar novos produtos para o backlog e/ou fazer com que alguns que estavam presentes neste sprint retornem ao backlog.
* **Sprint Retrospective meeting:**
* Esta reunião tem o propósito de avaliar a metodologia a ser utilizada para encontrar falhas ou possíveis melhorias.
* Tudo aquilo que correu bem e o que correu mal é documentado e novas ideias são discutidas;
* No final desta reunião o plano **SCM** poderá ser atualizado adaptando-se às necessidades da equipa.
* Com esta reunião é possível refinar cada vez mais o processo de desenvolvimento, fazendo com que o mesmo seja melhorado no final de cada sprint.
* Todo o interveniente no sprint é obrigado a estar presente nesta reunião que demora cerca de meia a uma hora;
* **Horário de reuniões**
  + Até dia 12/11/2021
    - Reunião de planeamento de sprint: sexta-feira das 12:00 às 13:00
    - Reuniões de revisão e retrospetiva de sprint: sexta-feira das 11:00 às 12:00
  + Semana 12/11/2021 (sexta-feira) a 17/11/2021 (quarta-feira)
    - Reunião de planeamento de sprint: sexta-feira das 9:00 às 11:00
    - Reuniões de revisão e retrospetiva de sprint: quarta-feira das 16:20 às 17:20
  + Depois de dia 17/11/2021
    - Reunião de planeamento de sprint: quarta-feira das 17:30 às 18:30
    - Reuniões de revisão e retrospetiva de sprint: quarta-feira das 16:20 às 17:20
* **Nota: As reuniões diárias ocorrem espontaneamente na ESTG ou podem ser marcadas dinamicamente pelo Scrum Master.**

# Recursos e ferramentas

* GitLab – <https://gitlab.com/>
* Git – <https://git-scm.com/>
* Visual Studio – <https://visualstudio.microsoft.com/>
* Visual Studio Code – <https://code.visualstudio.com/>
* C# – <https://docs.microsoft.com/en-us/dotnet/csharp/>
* Swagger – <https://swagger.io/>
* Angular – <https://angular.io/>
* Mockflow – <https://www.mockflow.com/>
* Docker – <https://www.docker.com/>