

Projet Génie Logiciel

Gestion de projet

Manuel d'utilisation de l'espace Gitlab

Équipe GL2

Étudiants : Élina Houdé, Julien Pinto Da Fonseca,

Léa Solo Kwan, Yin Xu

Sommaire

Introduction	3
1. Configuration globale	4
1.1. Création et protection des branches	4
1.2. Configuration de la méthode de merge	11
2. Configuration du board	12
2.1. Board Development / Board Technical	12
2.2. Ajout d'un tableau de bord	13
2.3. Utilisation de label	15
2.4. Configuration des labels	16
2.5. Configuration des milestones	19
2.6. Configuration des templates par défaut des issues	20
3. Utilisation du board	22
3.1. Création d'une issue	22
3.2. S'attribuer une User Story	24
3.3. Gestion de la colonne Ready to Test/Testing	26
3.4. Utilisation d'un Burndown Chart	27
4. Configuration de l'intégration continue	29
4.1. Activation du runner	29
4.2. Initialisation de l'environnement	32
4.3. Le détail des différents stages	33
4.4. Spécification de chacun des stages	34

Introduction

Ce document décrit l'utilisation de l'outil Gitlab pour la gestion d'un projet, et plus précisément pour le projet Génie Logiciel. Plus spécifiquement, celui-ci est adressé aux équipes utilisant les méthodologies agiles Scrum/Kanban. Cependant, ce document décrit par ailleurs beaucoup de méthodes qui pourront être utilisées peu importe le type de projet réalisé.

1. Configuration globale

Afin d'éviter la régression du code et tout conflits, il peut être intéressant de posséder deux branches principales :

- **develop** : branche de développement ;
- **master** : branche principale que l'on utilisera pour merge la branche develop lors des différentes livraisons.

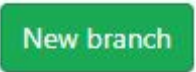
Par défaut, la branche develop est donc la branche utilisée par les développeurs.

1.1. Création et protection des branches

La démarche pour la configuration des branches est la suivante :

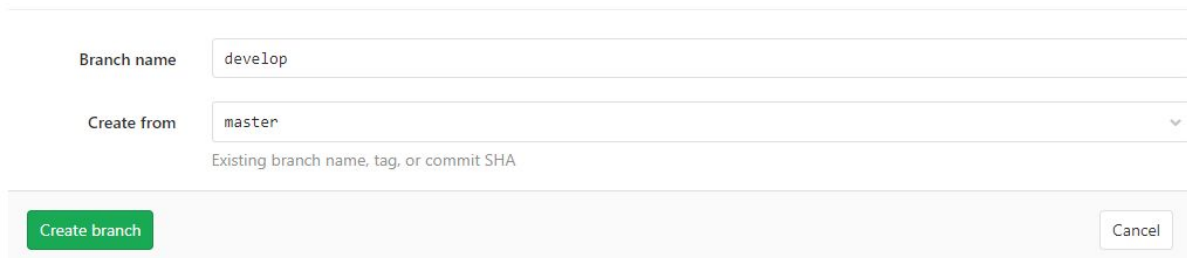
1. Création de la branche develop

Dans l'onglet *"Repository"* > *"Branches"*, sélectionner *"New branch"* :

A green rectangular button with the text "New branch" in white.

Saisir le nom de la nouvelle branche dans *"Branch name"* et sélectionner le bon champ *"create from"* :

New Branch

A form titled "New Branch" with two input fields. The first field, labeled "Branch name", contains the text "develop". The second field, labeled "Create from", contains the text "master" and has a dropdown arrow on the right. Below the second field is the text "Existing branch name, tag, or commit SHA". At the bottom of the form are two buttons: a green "Create branch" button on the left and a grey "Cancel" button on the right.

Branch name	develop
Create from	master
Existing branch name, tag, or commit SHA	
Create branch	Cancel

2. Configuration de la branche develop en tant que branche par défaut

Dans l'onglet *"Settings"* > *"Repository"*, section *"Default Branch"*, sélectionner l'option suivante :

Default Branch

Select the branch you want to set as the default for this project. All merge requests and commits will automatically be made against this branch unless you specify a different one.

Default Branch

develop

☒ **Auto-close referenced issues on default branch**

Issues referenced by merge requests and commits within the default branch will be closed automatically ?

Save changes

3. Protection des branches

Afin d'interdire les push dits "manuels" sur notre branche principale et notre branche de développement, il est possible de les protéger. L'objectif est ici de ne pouvoir effectuer que des merge sur ces branches, en utilisant le board et les merge requests depuis Gitlab.

Pour cela, il faut effectuer des modifications via l'onglet "Settings" > "Repository", section "Protected Branches".

La branche master est généralement protégée par défaut : de ce fait, il suffit dans un premier temps de seulement modifier sa configuration. Ainsi, on sélectionne "Developers + Maintainers" pour le champ "Allow to merge", "No one" pour le champ "Allowed to push" et on prend soin de désactiver l'option "Code owner approval" :

Branch	Allowed to merge	Allowed to push	Code owner approval	
master	Developers + Mai... ▾	No one ▾	<input type="checkbox"/>	Unprotect

Dans un deuxième temps, il est nécessaire de protéger la branche develop en inscrivant directement les règles souhaitées pour celle-ci (identiques à ce que l'on vient de modifier pour la branche master) :

Protect a branch

Branch: Wildcards such as `*-stable` or `production/*` are supported

Allowed to merge:

Allowed to push: Only groups that [have this project shared](#) can be added here

Require approval from code owners: ☒ Pushes that change filenames matched by the CODEOWNERS file will be rejected

Le résultat final doit correspondre à la capture d'écran suivante :

Branch	Allowed to merge	Allowed to push	Code owner approval	
develop <small>default</small>	<input type="text" value="Developers + Mai..."/>	<input type="text" value="No one"/>	<input checked="" type="checkbox"/>	<input type="button" value="Unprotect"/>
master	<input type="text" value="Developers + Mai..."/>	<input type="text" value="No one"/>	<input checked="" type="checkbox"/>	<input type="button" value="Unprotect"/>

4. Double approbation pour les merge request

Dans l'optique d'une phase obligatoire de test et de code review effectuée par un ou plusieurs membres de l'équipe, il est également possible de configurer une règle d'approbation pour les merge requests sur ces branches. On notera deux règles spécifiques :

- L'approbation obligatoire d'au moins deux membres de l'équipe (y compris le développeur principal) pour chaque merge request effectué sur la branche develop.

- B. L'approbation obligatoire du responsable d'intégration de l'équipe pour chaque merge request, généralement de la branche develop, sur la branche master. Cela signifie qu'il sera le seul à pouvoir réaliser cette opération.

Afin de mettre en place une nouvelle règle, il suffit d'aller dans l'onglet *"Settings"* > *"General"*, section *"Merge request approvals"*, et de sélectionner *"Add approval rule"* :

Approval rules

Approvers	Target branch	No. approvals required
Any eligible user ?	Any branch	<input type="text" value="0"/>

[Add approval rule](#)

Pour mettre en place la première règle "A", on effectue le paramétrage suivant (en prenant bien soin d'ajouter chacun des membres de l'équipe en tant que *"Approvers"*) :

Add approval rule ×

Rule name

Merge to develop

e.g. QA, Security, etc.

No. approvals required

2

Target branch


develop


▼


Apply this approval rule to any branch or a specific protected branch.


Approvers


Search users or groups


 Julien Pinto Da Fonseca



 Membre n°2



 Membre n°3



Cancel

Add approval rule

Afin de mettre en place la seconde règle “B”, on effectue le paramétrage suivant (en prenant bien soin de n’ajouter que le responsable d’intégration de l’équipe dans la liste des “Approvers”) :

Add approval rule ×

Rule name

e.g. QA, Security, etc.


No. approvals required

Target branch

▼

Apply this approval rule to any branch or a specific protected branch.

Approvers




 Julien Pinto Da Fonseca 🗑️

Cancel

Add approval rule

Une fois ces deux règles mises en place, le résultat doit correspondre à la capture d'écran suivante :


Approval rules


Name	Approvers	Target branch	No. approvals required	
Any eligible user 		Any branch	<input type="text" value="0"/>	
Merge to develop		develop	<input type="text" value="2"/>	<input type="button" value="Edit"/> <input type="button" value="Delete"/>
Merge to master		master	<input type="text" value="1"/>	<input type="button" value="Edit"/> <input type="button" value="Delete"/>

Il est également possible de modifier les différentes options disponibles. Plus spécifiquement, on peut s'intéresser à *"Can override approvers and approvals required per merge request"*. L'activation de cette option permet de garder la possibilité de modifier les règles d'approbation, lors de l'édition d'une merge request. Cependant, même s'il s'agit d'une mauvaise pratique qui faut éviter d'utiliser, cette fonctionnalité peut être utile dans des cas très particuliers (ex : ajout de correctifs de dernière minute avant une livraison). Dans le cas où cette fonction est désactivée, seules les règles d'approbation "A" et "B" ajoutées précédemment sont valables sur l'ensemble des merge requests.


Par ailleurs, il est recommandé de désactiver l'option *"Prevent approval of merge requests by merge request author"*, afin de permettre au développeur principal, auteur de la merge request, d'approuver celle-ci.


Une fois cette partie complétée, le résultat doit correspondre à la capture d'écran suivante (modulo la première option) :


The "Require approval from CODEOWNERS" setting was moved to **Protected Branches** 

☒ Can override approvers and approvals required per merge request 

☒ Remove all approvals in a merge request when new commits are pushed to its source branch

☐ Prevent approval of merge requests by merge request author 

☐ Prevent approval of merge requests by merge request committers 

☐ Require user password to approve 

1.2. Configuration de la méthode de merge

Lors du développement d'une fonctionnalité, il peut être intéressant de ne pas effectuer de merge request à chaque commit afin d'éviter d'interférer avec les tâches des autres développeurs. De cette manière, la fonctionnalité n'est partagée avec le reste de l'équipe qu'une fois qu'elle est totalement implémentée. Cela permet de limiter le risque de générer des erreurs sur les autres fonctionnalités en cours de développement.

Pour cela, il faut configurer la méthode de merge dans l'onglet *"Settings"* > *"General"* comme suit :

Merge method

This will dictate the commit history when you merge a merge request

- ☐ Merge commit
Every merge creates a merge commit
- ☐ Merge commit with semi-linear history
Every merge creates a merge commit
Fast-forward merges only
When conflicts arise the user is given the option to rebase
- ☒ Fast-forward merge
No merge commits are created
Fast-forward merges only
When conflicts arise the user is given the option to rebase

2. Configuration du board

Lors d'un projet, il est également important de bien cerner le travail à réaliser et de savoir le découper en tâches à effectuer. Dans cette optique, il est très intéressant d'utiliser certaines fonctionnalités comme :

- les tableaux de bord ;
- les labels ;
- les milestones ;
- les templates pour les US.

2.1. Board Development / Board Technical

Afin de bien différencier les différents types de tâches qui doivent être réalisées au cours d'un projet, il peut être intéressant de créer deux tableaux de bord:

- *Board Development* : dans ce tableau de bord sont affichées toutes les tâches de développement ;
- *Board Technical* : on retrouve ici toutes les tâches techniques, i.e. autres que de développement (ex: réalisation de la documentation technique, préparation des réunions de suivi, etc.).

1. Tableau de bord Development

Le tableau de bord *Development* comprend les colonnes suivantes :

- Ready for Development : cette catégorie regroupe les tâches qui peuvent être réalisées et dont les objectifs sont définis ;
- In Progress : dans cette colonne sont placée toutes les US en cours de développement par un membre de l'équipe ;
- Ready to Test/Testing : lorsqu'une tâche a été implémentée, elle est placée dans cette catégorie afin qu'un autre membre de l'équipe puissent effectuer diverses vérifications.

2. Tableau de bord Technical

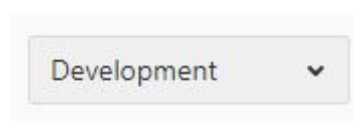
Le tableau de bord *Technical* comprend les colonnes suivantes :

- Ready for Development : cette catégorie regroupe les tâches qui peuvent être réalisées et dont les objectifs sont définis ;
- In Progress : dans cette colonne sont placée toutes les US en cours de réalisation par au moins un membre de l'équipe ;
- Ready to Test/Testing : lorsqu'une tâche technique n'est réalisée que par un membre de l'équipe, cette colonne permet d'attendre la vérification de la conformité de l'US par le reste de l'équipe si nécessaire.

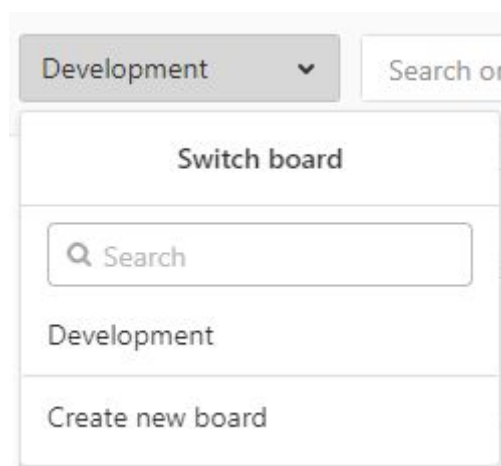
2.2. Ajout d'un tableau de bord

Afin de procéder à la création d'un tableau de bord, généralement appelé board, on se positionne tout d'abord sur la vue par défaut via l'onglet "*Issues*" > "*Boards*".

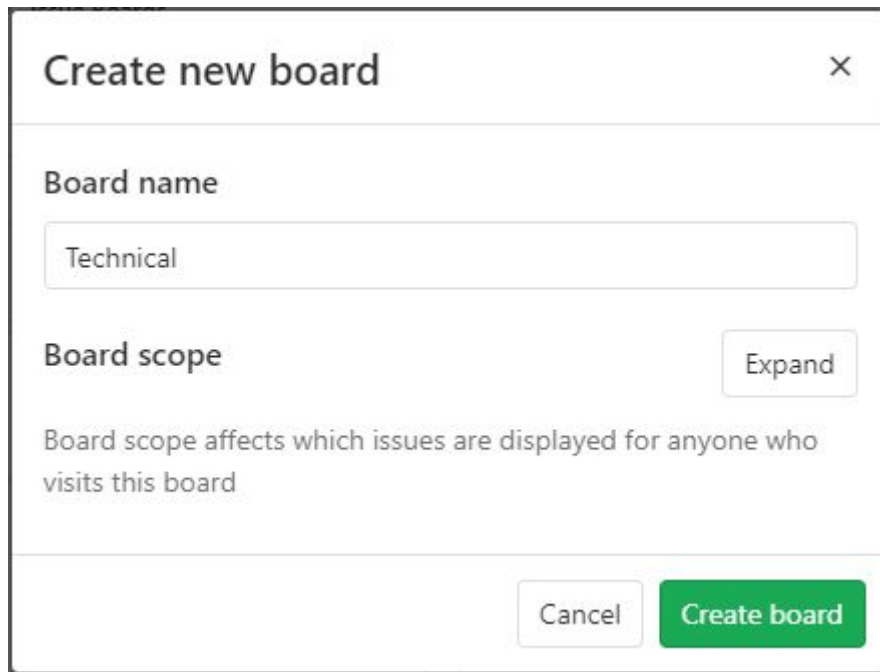
On sélectionne ensuite le board actuel comme suit :



Puis, on sélectionne "Create new board" :

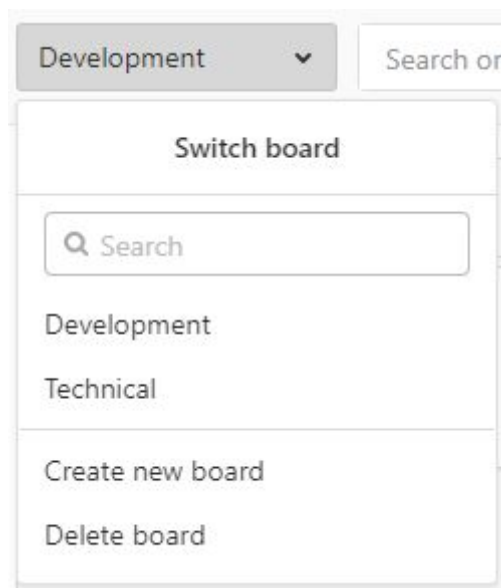


Il suffit ensuite de saisir le nom du nouveau board (par exemple "*Technical*") puis de valider en sélectionnant "*Create board*".



The screenshot shows a modal dialog titled "Create new board" with a close button (X) in the top right corner. Inside the dialog, there is a section for "Board name" with a text input field containing the word "Technical". Below this is a section for "Board scope" with an "Expand" button. A descriptive text below the "Board scope" section states: "Board scope affects which issues are displayed for anyone who visits this board". At the bottom of the dialog, there are two buttons: "Cancel" and "Create board" (which is highlighted in green).

Pour passer d'une vue à une autre et donc sélectionner un board en particulier, on répètera l'action "sélectionner le board actuel" effectuée précédemment :



The screenshot shows a dropdown menu titled "Switch board" that appears below a "Development" button with a downward arrow. The dropdown menu contains a search input field with a magnifying glass icon and the text "Search". Below the search field, there are four options listed: "Development", "Technical", "Create new board", and "Delete board".

2.3. Utilisation de label

Les labels (ou étiquettes) permettent de regrouper les User Stories par thématique (thème, dev, déploiement), statut (à faire, en cours, à déployer), ou toute autre classification que vous aurez envie de définir. Cela permet de classer les US dans les tableaux de bord et de les filtrer selon ces étiquettes.

Certains labels seront souvent nécessaires, peu importe la nature de votre projet de développement :

- "Problem" : une fois associée à une US de la colonne *Ready to Test/Testing*, cette étiquette permet de signaler au(x) développeur(s) de la tâche qu'un des testeurs a rencontré un problème lors de la vérification de la fonctionnalité ;
- "Blocked" : ce label indique aux autres membres de l'équipe que le développement d'une fonctionnalité est interrompue suite à un problème rencontré par le(s) développeur(s). En positionnant ce label sur l'US, ce(s) dernier(s) requier(en)t l'aide des autres membres de l'équipe ;
- "Tests OK" : ce label permet de signaler au responsable d'une US que celle-ci a été validée par un autre membre de l'équipe et qu'il peut donc, s'il le souhaite, effectuer une merge request. Néanmoins, il existe d'autres solutions alternatives à l'utilisation de cette étiquette. Par exemple, on peut tout simplement vérifier avant d'effectuer une merge request que l'US a été approuvée par un membre tiers de l'équipe. L'utilisation de ce label est donc laissée à votre appréciation.

A cette liste s'ajoutent les labels spécifiques à votre projet. Dans le cas du projet Logiciel, vous pouvez donc ajouter les étiquettes suivantes :

- Shell : ce label va concerner toutes les tâches de réalisation de script Shell (ex : basic-synt.sh, etc.) ;
 - Gestion de projet : tâches concernant tout ce qui touche à la gestion de projet, que ce soit la documentation interne à l'équipe, la préparation de réunions SCRUM ou encore la réalisation de compte-rendus ;
 - Test : réalisation des tests systèmes avant la réalisation des étapes A, B et C ;
 - Etape A ;
 - Etape B ;
-

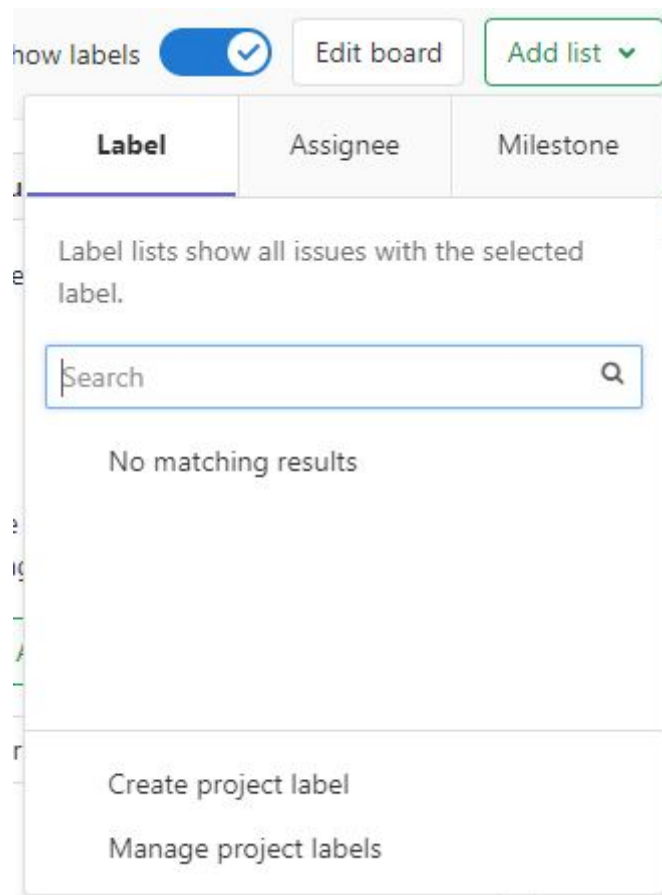
- Etape C ;
- Documentation : ce label est destiné à toutes les tâches pour la réalisation de la documentation externe du projet, i.e. la documentation à fournir à l'utilisateur, etc. ;
- Bibliothèque standard : tâches concernant l'implémentation des extensions.

2.4. Configuration des labels

Afin de procéder à la création d'un label, on se positionne tout d'abord sur la vue de notre board par défaut : onglet "Issues" > "Boards". On sélectionne ensuite "Add list",

A green rectangular button with the text "Add list" and a small downward-pointing chevron icon.

puis "Manage project labels" (ou éventuellement "Create project label" qui est un raccourci de création rapide),



et enfin "New Label" :

Labels can be applied to issues and merge requests to categorize them.

You can also star a label to make it a priority label.

New label

Generate a default set of labels

On peut ensuite procéder à la création du label souhaité, dans notre exemple il s'agit du label nommé "Documentation" dont la description est "Technical" car celui-ci ne sera utilisé qu'au sein du board Technical :

New Label


Title

Use `::` to create a scoped label set (eg. `priority::1`)

Description

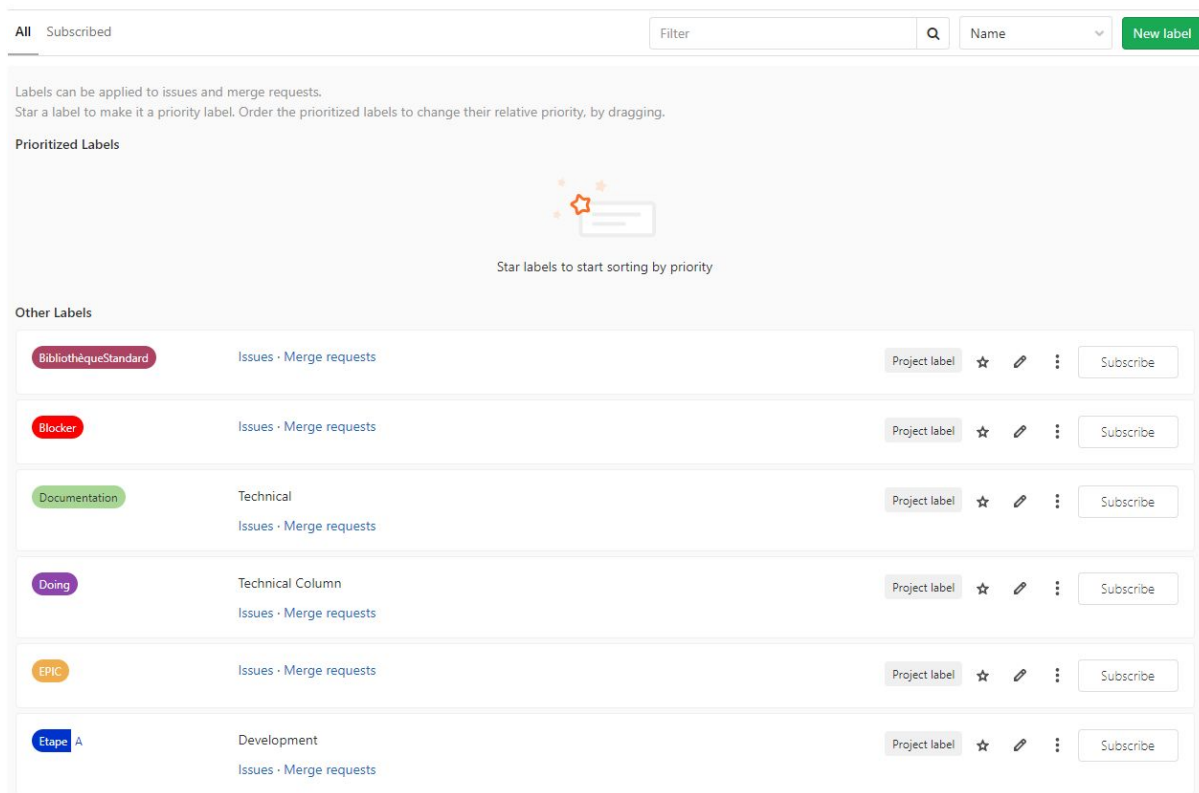
Background color

Choose any color.
Or you can choose one of the suggested colors below



Pour créer plusieurs labels, il suffit de répéter l'opération précédente ou alors de re-sélectionner "New label".

Un fois le premier label créé, une page d'aperçu répertoriant les différents labels existants est mise à disposition :



Note : si le nom du label est précédé d'un mot-clé, suivi de "::" : cela permet de rendre les différents labels précédé par ce même mot-clé incompatible entre-eux. Par exemple, imaginons qu'on ait les labels suivants :

- step::A
- step::B
- step::C

Si l'on assigne le label step::A à une issue, puis que l'on assigne ensuite le label step::C à cette même issue, le label step::A sera automatiquement remplacé par le label step::C. Au contraire, si nos labels étaient nommés simplement A, B et C, le fait d'affecter le label A à une issue puis le label C à cette même issue aurait pour effet de cumuler les deux labels pour cette même issue.

2.5. Configuration des milestones

Dans le but de visualiser les différents rendus à effectuer ou se représenter des sprints ou des versions à venir, il est également possible de regrouper plusieurs US au sein d'un milestone.

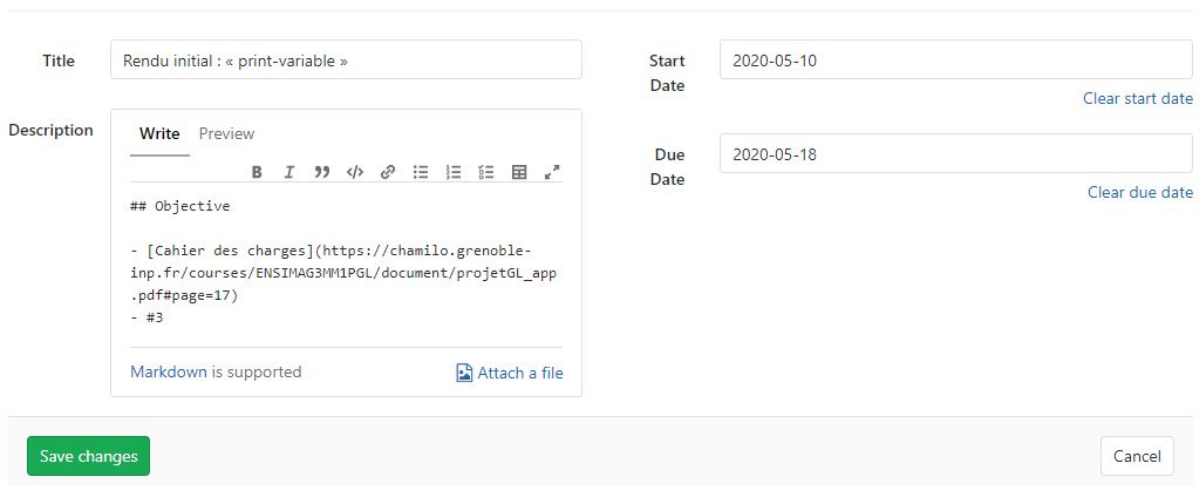
Afin de procéder à la création d'un milestone, on se positionne tout d'abord sur la vue des différents milestones ouverts par défaut via l'onglet *"Issues"* > *"Milestones"*.

On prendra ensuite le soin de sélectionner *"New milestone"*,

A green rectangular button with the text "New milestone" in white.

puis de saisir directement les renseignements souhaités tels que le montre l'exemple suivant :

Edit Milestone

A screenshot of the "Edit Milestone" form in GitLab. The form has a light gray background and contains several input fields and a rich text editor. At the top, the title "Edit Milestone" is displayed. Below it, there are three main sections: "Title", "Description", and "Dates". The "Title" section has a text input field containing "Rendu initial : « print-variable »". The "Description" section has a rich text editor with a "Write" tab selected, showing a list of tasks: "## Objective", "- [Cahier des charges](https://chamilo.grenoble-inp.fr/courses/ENSIMAG3M1PGL/document/projetGL_app.pdf#page=17)", and "- #3". The "Dates" section has two date input fields: "Start Date" with the value "2020-05-10" and "Due Date" with the value "2020-05-18". There are links "Clear start date" and "Clear due date" next to the date fields. At the bottom of the form, there are two buttons: "Save changes" (green) and "Cancel" (gray).

Title: Rendu initial : « print-variable »

Start Date: 2020-05-10 [Clear start date](#)

Due Date: 2020-05-18 [Clear due date](#)

Description: **Write** **Preview**

Objective

- [Cahier des charges](https://chamilo.grenoble-inp.fr/courses/ENSIMAG3M1PGL/document/projetGL_app.pdf#page=17)
- #3

Markdown is supported [Attach a file](#)

[Save changes](#) [Cancel](#)

2.6. Configuration des templates par défaut des issues

Afin de détailler tous les points importants concernant la réalisation d'une tâche, il est possible de définir un template par défaut pour vos User Stories. De cette manière, dès qu'une User Story est créée, sa description sera pré-remplie.

Cette option est configurable dans l'onglet "Settings" > "General" :

Default issue template

Set a default template for issue descriptions.

Default description template for issues ?

```
## Epic definition
—

## Story statement
**En tant que** __
**Je souhaite** __
**Afin de** __
```

Description parsed with [GitLab Flavored Markdown](#)

Vous pouvez par exemple utiliser le template suivant :

```
## Epic definition
—

## Story statement
**En tant que** __
**Je souhaite** __
**Afin de** __

### Additional information
---
```

—

Acceptance criteria

- [] First task

- [] Second task

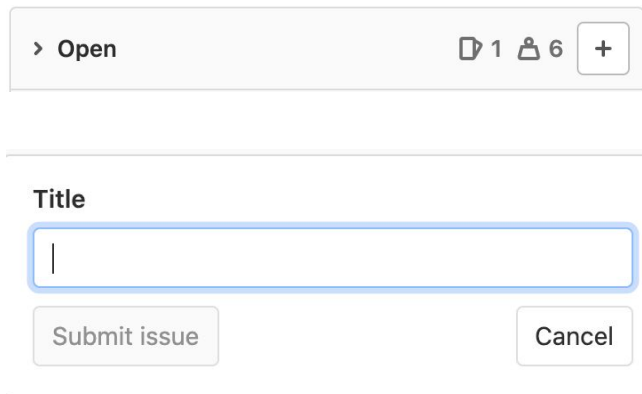
Ce template vous permettra de :

- définir le but de l'US dans la partie *"Story statement"* ;
- d'indiquer les ressources ou informations utiles pour la réalisation de la tâche dans la partie *"Additional information"* ;
- préciser les critères d'acceptation pour la validation de la tâche dans la partie *"Acceptance criteria"*.

3. Utilisation du board

Afin de bien utiliser les fonctionnalités précédemment configurées, les méthodes décrites ci-dessous peuvent être utilisées.

3.1. Création d'une issue



La création d'une issue se fait selon les étapes suivantes :

1. Création de l'issue.

Pour cela, il suffit de cliquer sur l'icône '+' et, ensuite de donner un titre à l'issue.

2. Définition du but de l'issue

Afin de comprendre quelle tâche effectuer, il est judicieux de compléter le corps de l'issue (clic droit sur le nom de l'issue). Cela peut être réalisé facilement à l'aide du template défini précédemment (cf 2.6).

3. Configuration des paramètres

En effectuant un clic gauche sur l'issue, un panneau s'affiche sur la droite. Vous pourrez y définir les paramètres suivants :

- le poids de l'issue (weight) : il s'agit d'une estimation de la proportion de travail à effectuer ;
- le milestone ;
- les labels associés à la tâche.

The diagram illustrates the configuration of a GitLab issue. On the left, the initial state shows a basic issue configuration. On the right, the final state shows a more detailed configuration with an assignee, labels, and weight.

Initial State (Left Panel):

- US_example #151** (Close icon)
- 0 Assignees** (Edit) - None - assign yourself
- Epic** (Edit) - None
- Milestone** (Edit) - None
- Time tracking** (?) - No estimate or time spent
- Due date** (Edit) - None
- Labels** (Edit) - None
- Weight** (Edit) - None
- Notifications** (Toggle) - On

Final State (Right Panel):

- US_example #151** (Close icon)
- Assignee** (Edit) - Yin Xu (@xuyi)
- Epic** (Edit) - None
- Milestone** (Edit) - None
- Time tracking** (?) - No estimate or time spent
- Due date** (Edit) - None
- Labels** (Edit) - Etape A
- Weight** (Edit) - 6 - remove weight
- Notifications** (Toggle) - On

A la fin de cette procédure, la nouvelle issue ressemblera à l'image ci-dessous :

The screenshot shows a GitLab issue card in a list view. The card is titled "US_example" and has a label "Etape A". It also shows the issue number "#151" and a notification icon with the number "6".

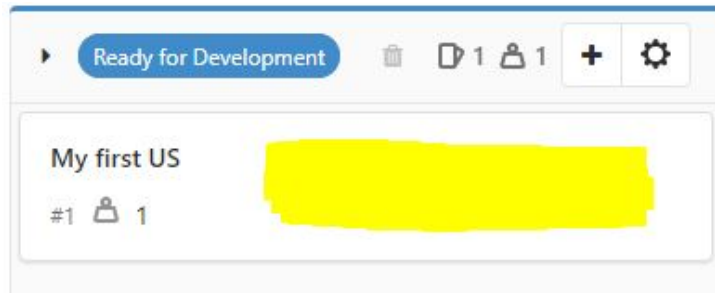
Issue Card Details:

- > Open** (Dropdown)
- 1** (Document icon)
- 6** (User icon)
- +** (Add icon)
- US_example**
- Etape A** (Label)
- #151** (Issue number)
- 6** (Notification icon)
- User Avatar** (Profile picture)

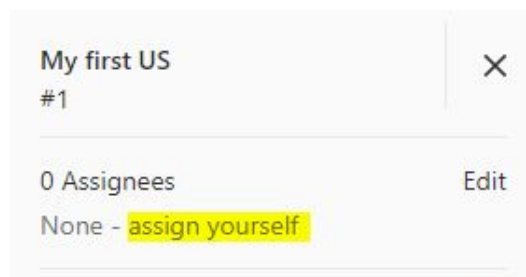
3.2. S'attribuer une User Story

Il est possible de s'attribuer une US seulement si celle-ci se trouve dans la colonne *Ready for Development* :

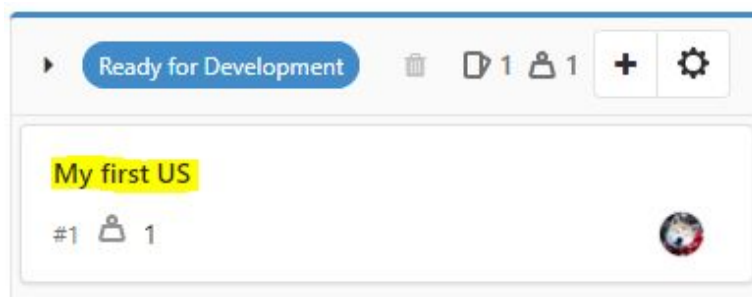
1. J'effectue un clic gauche sur l'US (zone hors titre) :



2. Je clique sur "assign yourself" dans la section "Assignees" du panneau à droite :



3. Je clique sur le titre de l'US pour voir celle-ci en détail :

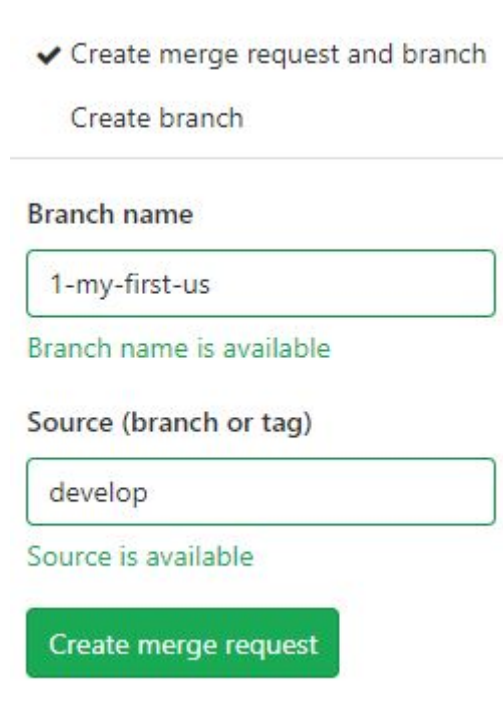


Je peux ainsi prendre connaissance du but de l'US et de ces critères de réussite.

4. Si besoin de développement, je clique sur le menu juste à côté du bouton "Create merge request" :



5. Je vérifie que les paramètres sont corrects et je clique sur *"Create merge request"* :



Attention : la source doit être la branche develop !

6. J'effectue la commande *"git fetch"* sur mon terminal afin de visualiser les merge request ouvertes :

```
Julien@SkyDesktop MINGW64 /c/Users/Julien/Documents/Git/testproject (develop)
$ git fetch
From gitlab.ensimag.fr:pintodaj/testproject
* [new branch]      1-my-first-us -> origin/1-my-first-us
```

7. J'effectue la commande *"git checkout NOM_BRANCHE"* sur mon terminal afin de me positionner sur une branche :

```
Julien@SkyDesktop MINGW64 /c/Users/Julien/Documents/Git/testproject (develop)
$ git checkout 1-my-first-us
Switched to a new branch '1-my-first-us'
Branch '1-my-first-us' set up to track remote branch '1-my-first-us' from 'origin'.
```

8. Je peux désormais développer la fonctionnalité indépendamment des autres User Stories en cours de développement.

3.3. Gestion de la colonne *Ready to Test/Testing*

Afin d'effectuer quelques vérifications sur les fonctionnalités implémentées, la procédure suivante peut être utilisée lorsqu'une US est placée dans la colonne Ready to Test/Testing.

Cette procédure est tout particulièrement destinée aux équipes ayant choisi la double approbation (cf. 1.1.4.) mais est aisément adaptable à d'autres stratégies de gestion de projet.

Selon que vous ayez réalisé la tâche ou que vous souhaitiez la vérifier, la procédure à suivre n'est pas la même :

1. Procédure pour le responsable de la tâche, i.e. celui qui a réalisé la tâche

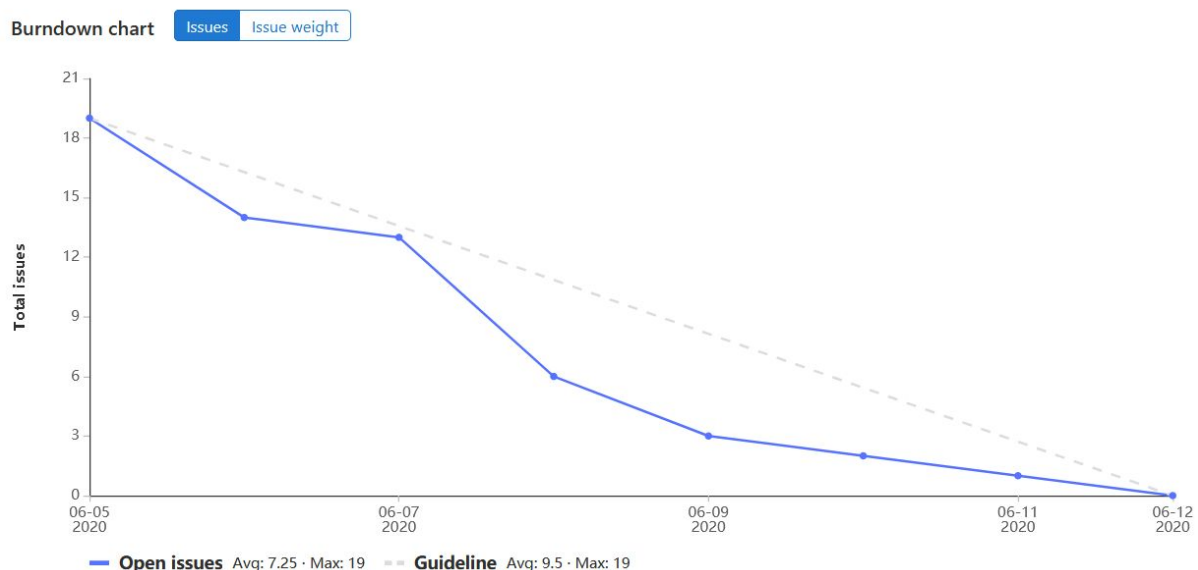
- Lorsque je place ma tâche dans la colonne *Ready to Test/Testing*, je reste assignée à la tâche.
- Si ma tâche comporte le label "Problem" :
 - Un testeur a rencontré un problème. Je peux alors consulter le commentaire qu'il a laissé.
 - Je repasse la tâche dans "en cours" pour traiter le problème
 - Une fois le problème traité, j'enlève le label "Problem" et je peux repasser la tâche dans la colonne *Ready to Test/Testing*.
- Si ma tâche comporte le label "Tests OK" :
 - Je vérifie qu'il n'y a pas de commentaires non traités/lus.
 - J'approuve la merge request.
 - Je *rebase* si nécessaire.
 - Je passe la tâche dans le colonne *Ready to merge*.
 - J'effectue la demande de merge (Merge Request).

2. Procédure pour les testeurs

1. Je m'assigne à la tâche que je souhaite tester.
2. J'effectue/écris mes tests.
3. En cas de questions, je laisse un commentaire sur l'US et/ou me rapproche du développeur de la tâche.
4. Si je ne rencontre pas d'erreur :
 - J'ajoute le label "Tests OK" à l'US.
 - J'approuve la merge request.Si je rencontre une erreur :
 - J'ajoute le label "Problem" à l'US.
 - J'ajoute un commentaire à la tâche expliquant le problème.
5. Je me désassigne de la tâche.

3.4. Utilisation d'un Burndown Chart

Pour chaque milestone, il est possible de visualiser certaines statistiques dans l'onglet "Issues" > "List".



Le Burndown Chart permet de visualiser l'avancée d'un milestone en fonction du nombre d'US ouvertes ou du poids des US ouvertes. De plus, si une deadline est définie sur le milestone, le graphique permet de visualiser une estimation de la quantité de travail à réaliser quotidiennement pour la respecter.

Attention : afin d'avoir de bonnes estimations, il est conseillé de créer toutes les US du milestone lors du premier jour.

4. Configuration de l'intégration continue

L'intégration continue est un ensemble de pratiques utilisées en génie logiciel consistant à vérifier à chaque modification de code source que le résultat des modifications ne produit pas de régression dans l'application développée.

Le processus d'intégration continue (pipeline) permet de lancer un ensemble de tâches à effectuer, qui forment des jobs.

Il est possible, sous Gitlab, de mettre en place un système d'intégration continue en effectuant les différentes étapes décrites ci-dessous.

4.1. Activation du runner

En vu d'utiliser le système d'intégration continue disponible via Gitlab, il est nécessaire d'activer ce qu'on appelle un "runner". Il s'agit d'un processus constamment disponible qui se chargera de lancer les nouveaux jobs en attente.

Pour cela, il suffit d'aller dans l'onglet *"Settings"* > *"CI / CD"*, section *"Runners"* puis de sélectionner *"Enable shared Runners"*, ce qui permet d'activer le runner partagé mis à disposition par l'Ensimag spécialement pour cet usage :

Shared Runners

GitLab Shared Runners execute code of different projects on the same Runner unless you configure GitLab Runner Autoscale with MaxBuilds 1 (which it is on GitLab.com).

[Enable shared Runners](#) for this project

Available shared Runners: 1

● xxC35N1f

Ensimag's shared runner

#24

[docker](#)

Par défaut, le système d'intégration continue nécessite d'inscrire le détail de la configuration désirée au sein d'un fichier **.gitlab-ci.yml**, directement situé à la racine du projet en question.

Exemple de configuration (contenu du fichier **.gitlab-ci.yml**) :

```
1  # Gitlab ci/cd file configuration
2
3  default:
4    image: "maven:3-openjdk-8"
5
6  variables:
7    MAVEN_OPT_1: "-Dmaven.repo.local=.m2/repository"
8    MAVEN_OPT_2: "-Dmaven.exec.skip=true"
9    MAVEN_OPT_3: "-Pcobertura"
10
11  cache:
12    paths:
13      - .m2/repository/
14      - target/
15
16  before_script:
17    - export PATH=$PATH:$CI_PROJECT_DIR/global/bin"
18    - export PATH=$PATH:$CI_PROJECT_DIR/src/main/bin"
19    - export PATH=$PATH:$CI_PROJECT_DIR/src/test/script"
20    - export PATH=$PATH:$CI_PROJECT_DIR/src/test/script/launchers"
21
22  stages:
23    - build
24    - test
25    - cobertura
26
27  build:
28    stage: build
29    script:
30      - mvn clean compile $MAVEN_OPT_1
31
32  test:
33    stage: test
34    script:
35      - mvn test $MAVEN_OPT_1
36
37  cobertura:
38    stage: cobertura
39    script:
40      - cobertura-dump.sh $MAVEN_OPT_1 $MAVEN_OPT_3
41
```

Au sein de ce fichier de configuration, on distingue plusieurs parties :

- A. L'initialisation de l'environnement, lignes 1 à 21 ;
- B. Le détail des différents stages, lignes 22 à 26 ;
- C. Le contenu et la configuration de chacun des stages, lignes 27 à 41.

4.2. Initialisation de l'environnement

Pour la première partie, on commence tout d'abord par spécifier les dépendances dues au moteur de production utilisé au sein du projet. Dans notre cas, il s'agit de Maven :

```
3  default:
4  image: "maven:3-openjdk-8"
```

On peut ensuite définir d'éventuelles variables qui seront disponibles dans notre fichier de configuration : ceci est généralement utile pour souligner l'usage d'options lors de certains stages.

```
6  variables:
7    MAVEN_OPT_1: "-Dmaven.repo.local=.m2/repository"
8    MAVEN_OPT_2: "-Dmaven.exec.skip=true"
9    MAVEN_OPT_3: "-Pcobertura"
```

La section "*cache*" permet de gagner un temps considérable lors de l'exécution de plusieurs stages consécutifs. En effet, ce code rend possible la mise en cache des différents fichiers maven ainsi que des différents fichiers produits post-build (dossier target) :

```
11 cache:
12   paths:
13     - .m2/repository/
14     - target/
```

(Attention : dans le cas de la mise en cache des différents fichiers maven, le répertoire sera alors à spécifier en argument de chaque commande.)

Enfin, la section *"before_script"* contient un listing des différentes commandes qui seront exécutées avant le lancement des différents stages. On notera ici l'importance de cette section, notamment pour la mise à jour des différentes variables d'environnement nécessaires au bon fonctionnement de la plupart des scripts utilisés au sein des différents stages :

```
16  before_script:
17    - export PATH=$PATH:"$CI_PROJECT_DIR/global/bin"
18    - export PATH=$PATH:"$CI_PROJECT_DIR/src/main/bin"
19    - export PATH=$PATH:"$CI_PROJECT_DIR/src/test/script"
20    - export PATH=$PATH:"$CI_PROJECT_DIR/src/test/script/launchers"
```

Note : la variable *"\$CI_PROJECT_DIR"* est une variable prédéfinie. L'ensemble des variables prédéfinies est disponible via la page de documentation officielle Gitlab https://docs.gitlab.com/ee/ci/variables/predefined_variables.html.

4.3. Le détail des différents stages

Concernant la seconde partie, on notera seulement la présence de la section *"stages"*. Cette section contient le listing des différents stages, dans leur ordre d'exécution. Chaque stage contient les spécifications d'un job et ses éventuelles conditions d'exécution (dans notre cas : aucune) qui seront détaillées dans la partie suivante.

```
22  stages:
23    - build
24    - test
25    - cobertura
```

4.4. Spécification de chacun des stages

Enfin, la dernière partie spécifie directement la configuration de chacun des stages. Dans notre cas, par exemple, nous avons trois stages bien distincts. On peut donc constater que notre processus d'intégration continue sera divisé en trois étapes :

- **Build** : effectue la commande *mvn clean compile* pour vérifier que la phase de build du projet se déroule sans erreur, dans le cas contraire, le job échoue.

```
27 build:
28   stage: build
29   script:
30     - mvn clean compile $MAVEN_OPT_1
```

- **Test** : effectue la commande *mvn test* pour vérifier que l'ensemble des tests systèmes et unitaires se déroulent correctement ; la vérification du score des tests systèmes s'effectue manuellement en vérifiant les logs. Si une erreur est lancée ou qu'un test unitaire ne passe pas, le job échoue.

```
32 test:
33   stage: test
34   script:
35     - mvn test $MAVEN_OPT_1
```

- **Cobertura** : effectue un rapport cobertura récapitulant le taux de couverture des tests unitaires. Celui-ci peut être visible à tout moment via les logs sur le commit correspondant.

```
37 cobertura:
38   stage: cobertura
39   script:
40     - cobertura-dump.sh $MAVEN_OPT_1 $MAVEN_OPT_3
```

Ce processus d'intégration continue, également appelé ***pipeline***, est donc appelé à chaque fois qu'un commit est réalisé ou qu'une branche est "merge" sur les branches *develop* ou *master*. Une fois ce pipeline appelé, celui-ci procédera à l'exécution automatique des différents **jobs** associés, correspondant aux stages précédemment définis.