TerriSTORY

Documentation TerriSTORY®

docs.terristory.fr

Notice de contribution

Notice de contribution

Le dépôt du projet TerriSTORY®

Le projet TerriSTORY® est partagé en ligne sur la plateforme **GitLab** à l'adresse https://gitlab.com/terristory/terristory/.

Généralités sur les outils de systèmes de version

Git

Git est un système décentralisé de gestion de versions libre et open source qui permet de :

- Travailler à plusieurs sur un même projet ;
- Suivre l'évolution du code source ;
- Suivre et gérer les changements apportés au code ;
- Garder une trace de chaque changement apporté au code ;
- Revenir en arrière et comparer les versions antérieures du code, si une erreur est commise.

GitLab

GitLab est une plateforme libre en ligne qui offre la possibilité, entre autres, d'héberger des dépôts **Git** afin de permettre aux développeurs de travailler à plusieurs sur un même projet et de récupérer les modifications apportées par les autres développeurs. Elle propose des fonctionnalités supplémentaires comme un système de suivi des bugs et des outils d'intégration continue. GitLab est basé sur le logiciel **git**.

Contribution au projet TerriSTORY®

Il existe deux façons de contribuer au projet **TerriSTORY®** :

1. sans connaissance technique, en simple utilisateur, en déposant une idée, en signalant un bug, en rédigeant des tickets, etc.
2. avec des compétences de développement pour contribuer au code source de l'application.

En tant qu'utilisateur

Pour contribuer au projet TerriSTORY® en tant qu'utilisateur, GitLab propose un **système de gestion de tickets** (ou *issues* en anglais).

Un ticket GitLab permet de :

- Poser des questions techniques ;
- Signaler des incidents ;

- Obtenir de l'aide ;
- Faire de nouvelles demandes ;
- Faire des retours ;
- tout en gardant l'historique des conversations.

Généralement un ticket se compose de 3 parties :

- Un message : le sujet du ticket
- Un responsable : à qui est affecté le ticket
- Un statut : *to-do*, *in-progress*, *review*, *done*, etc.

Plus le traitement du ticket avance et plus il s'enrichit, par exemple :

- En l'affectant à un jalon (ou *milestone*) et à des labels (ou *catégories*) ;
- En ajoutant des pièces jointes ;
- De commentaires relatifs à sa résolution ou aux développements associés (questions/réponses) ;

Chaque ticket relatif à TerriSTORY® suit le cycle de vie suivant :

Cycle de vie d'un ticket

- **Ouverture**

Création et rédaction du ticket.

- **Priorisation**

Estimation de la priorité en fonction de l'importance de la modification associée au ticket : dans le cas de corrections de bugs ou d'améliorations mineures, le développement peut être intégré au fil de l'eau. Si, au contraire, il s'agit de changements importants ou d'une nouvelle fonctionnalité, le développement nécessitera d'être planifié.

- **Planification**

Affectation du ticket à un jalon.

- **Développement**

Lancement des développements informatiques nécessaires pour traiter le ticket.

- **Test et validation**

À partir du moment où le ticket est résolu, la branche contenant le correctif ou l'amélioration doit être testée avant de l'envoyer pour validation à l'équipe de développement. Il est possible de se référer à la page dédiée aux tests pour

connaître les différents tests à effectuer pour s'assurer du bon fonctionnement de la version développée. Une fois les tests passés avec succès, la branche peut être envoyée à l'équipe de développement qui va relire et valider la branche.

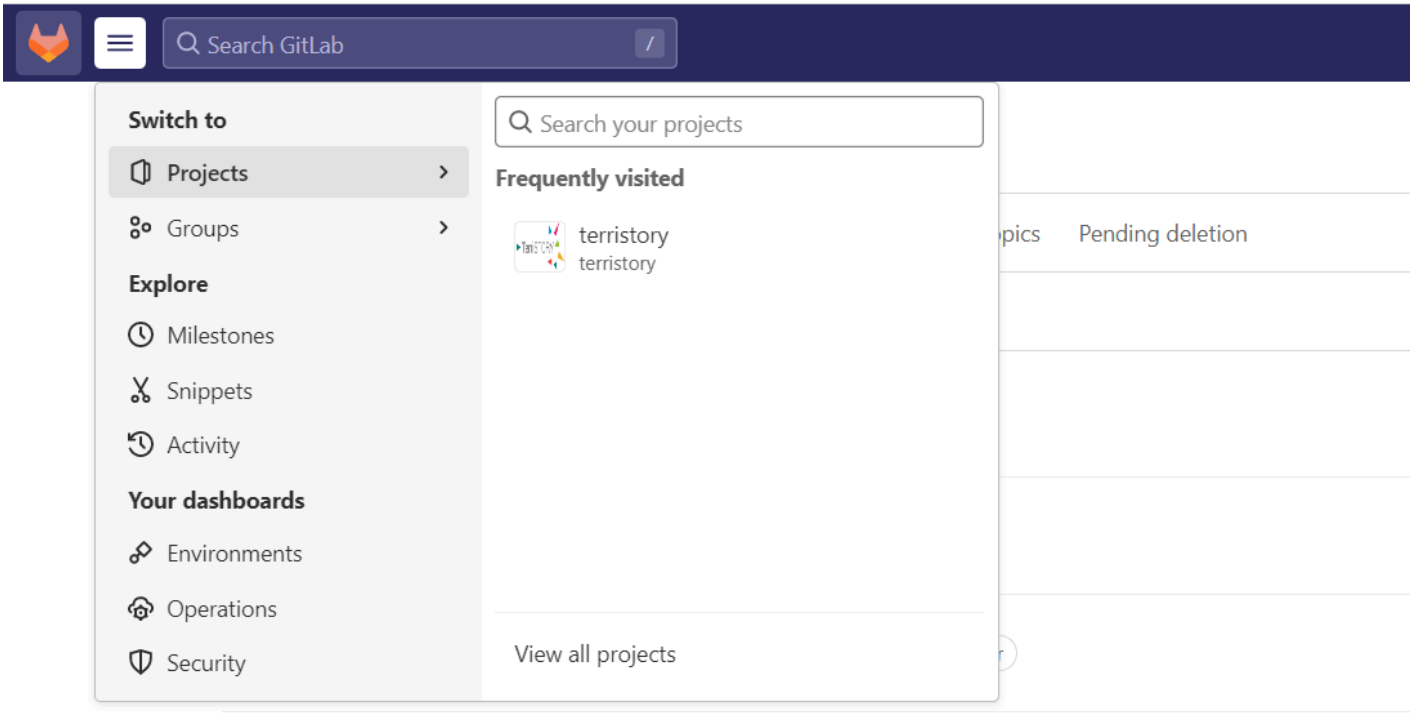
- **Fermeture**

Lorsque le ticket est validé, on peut ensuite le fermer. Il est tout à fait possible de le rouvrir en cas de besoin. Parfois, s'il manque certains éléments, si une idée supplémentaire a émergé ou si la nécessité d'un nouveau développement est apparue, on ferme le ticket initial dont on a répondu au besoin et on en ouvre un autre.

Créer des tickets par l'interface GitLab

- **Connexion** :

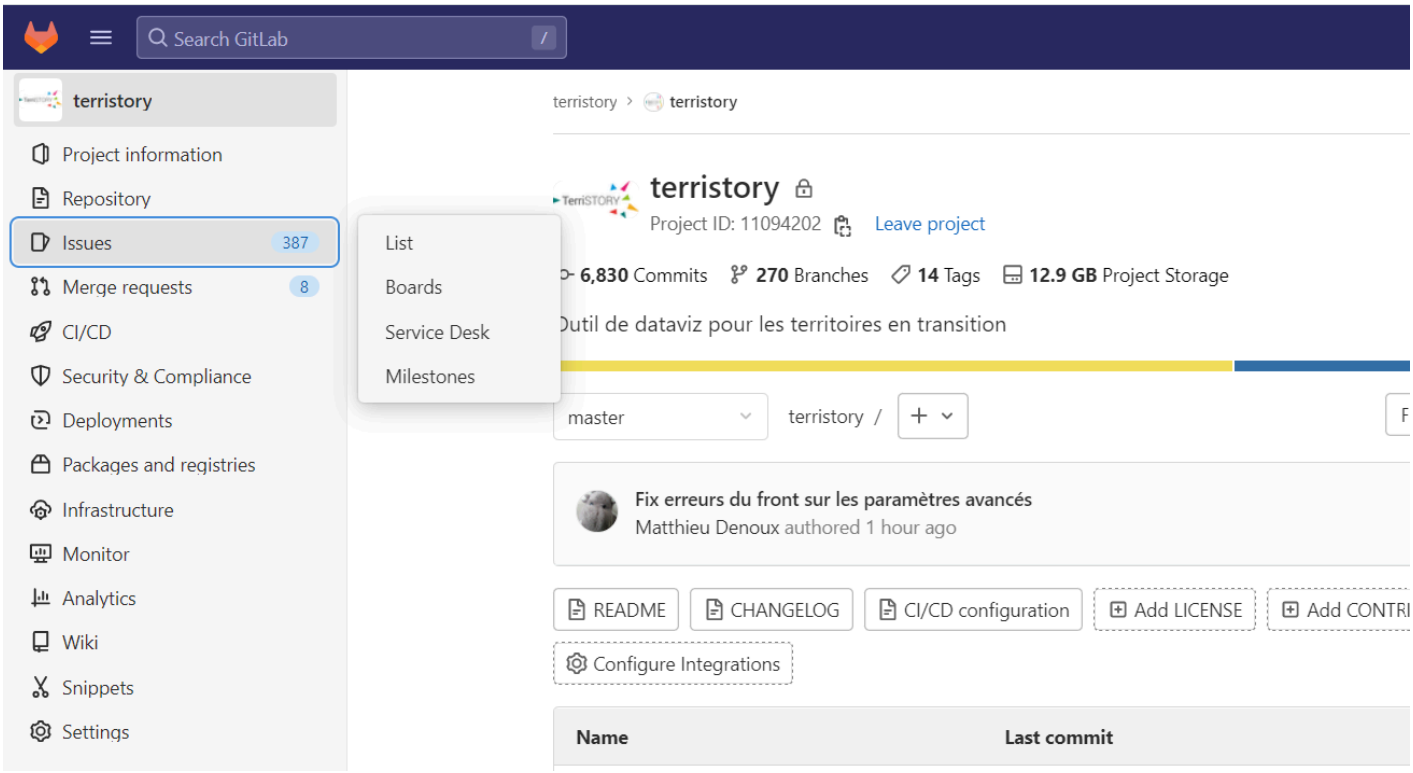
Se connecter à son compte GitLab à l'adresse gitlab.com et accéder au projet de son choix dans la section Projets.



Accéder au projet TerriSTORY® par exemple

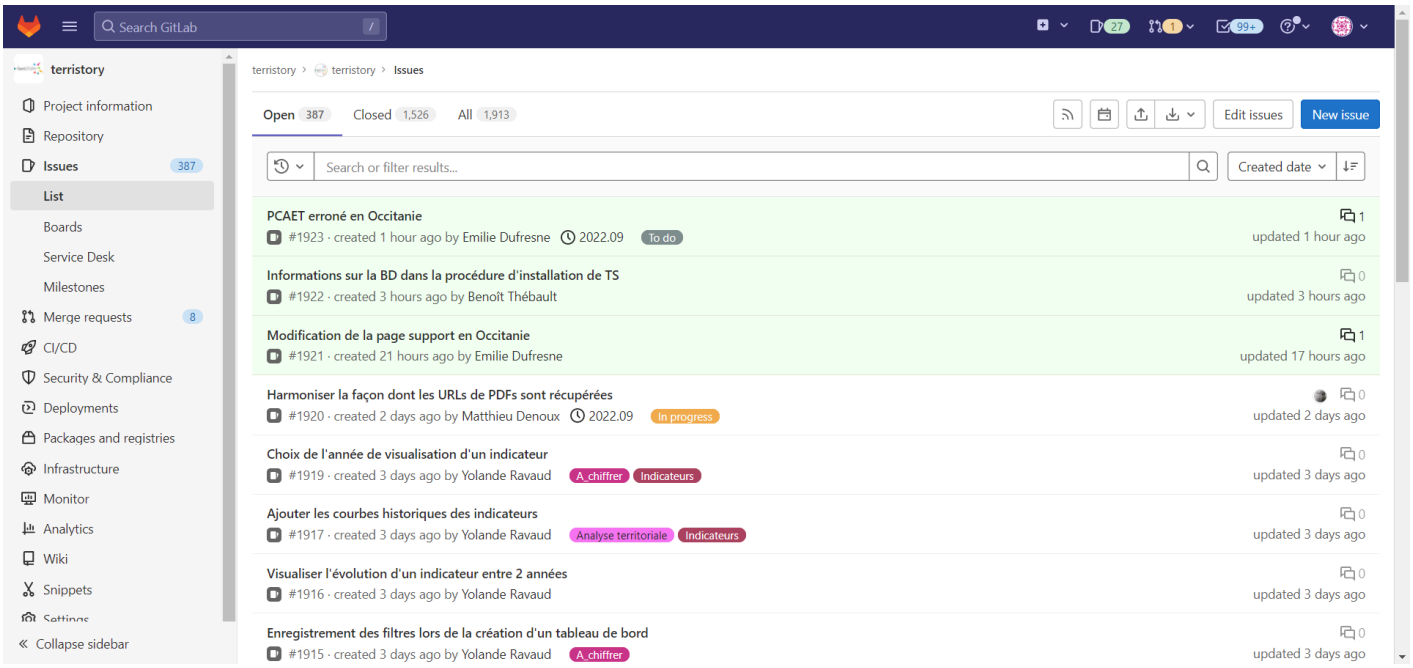
- **Visualiser tous les tickets existants dans le projet :**

Sur le tableau de bord de votre projet, accéder au menu Tickets et cliquer sur **List** pour lister tous les tickets ou **Boards** pour visualiser les tickets sous forme d'un tableau de bord dont il est possible d'organiser les colonnes et les filtres appliqués pour créer votre propre visualisation.



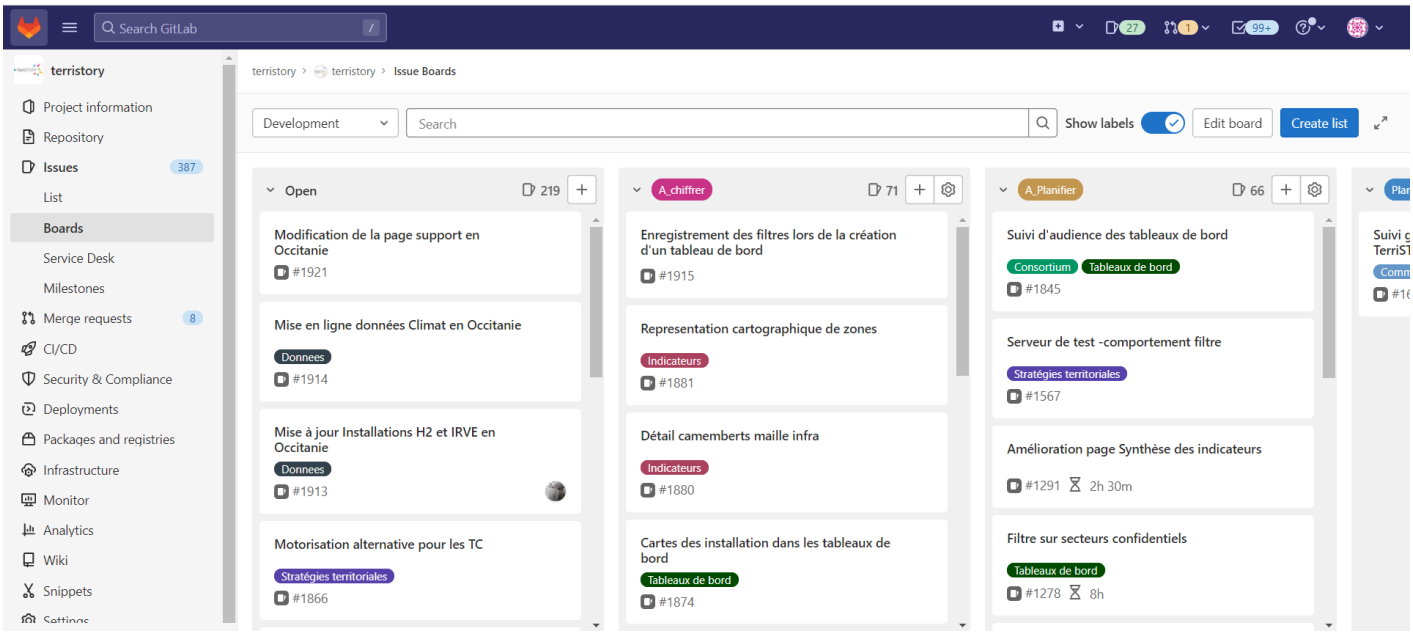
Lister les tickets

1. Visualiser les tickets sous forme d'une liste :



Lister les tickets - List

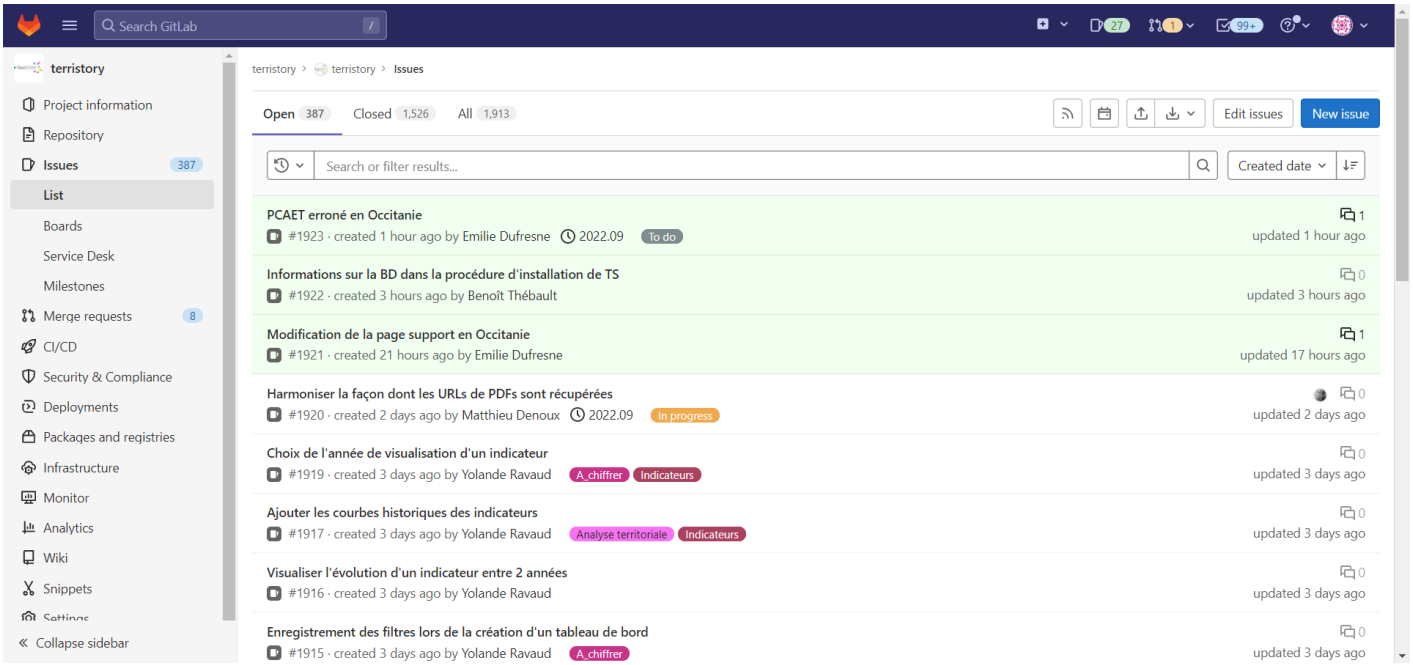
1. Visualiser les tickets sous forme d'un tableau de bord



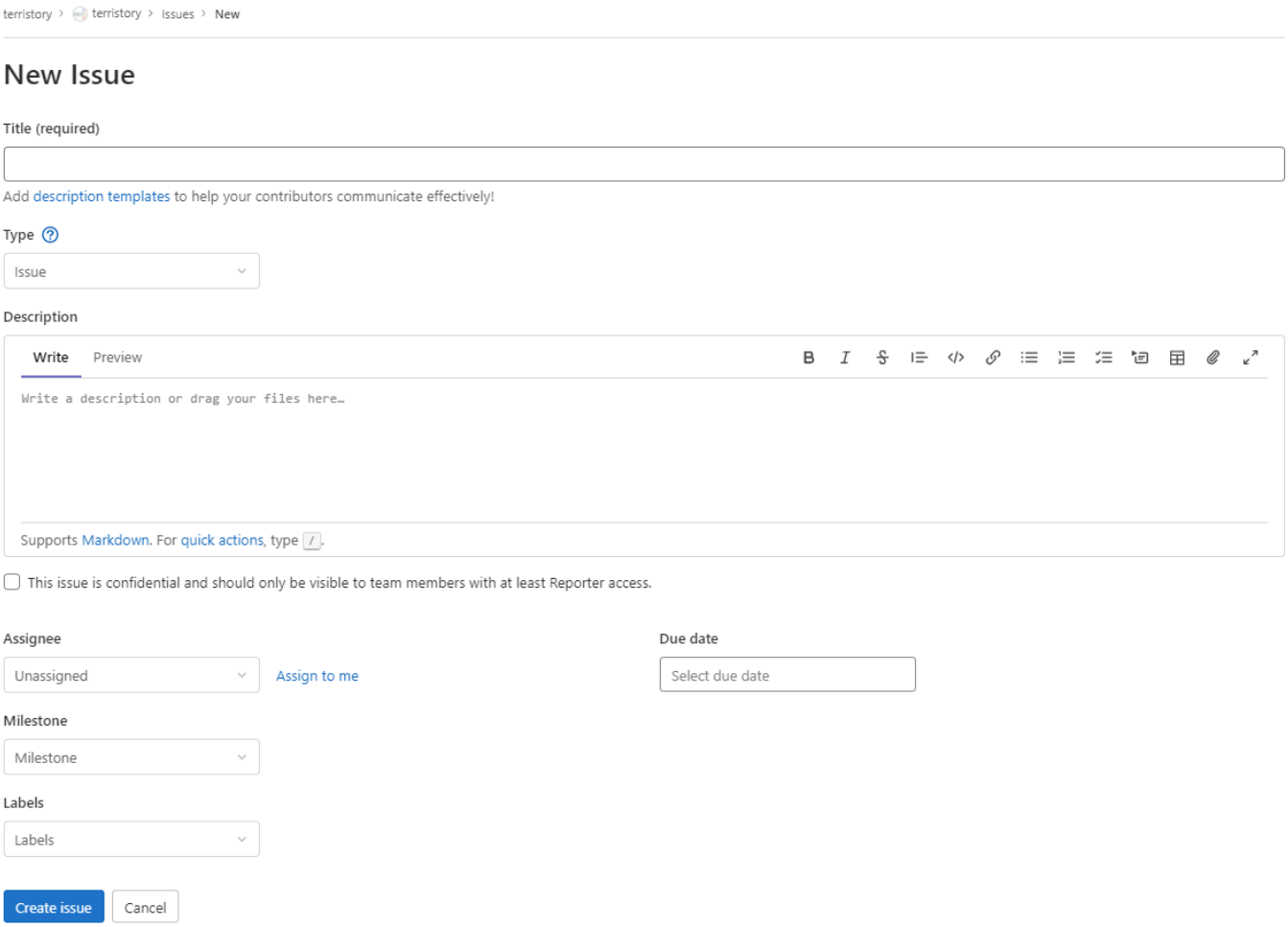
Lister les tickets - Board

- **Créer un ticket** :

Après avoir choisi un projet et cliqué sur *Issues* > *List* > *New Issue* (ou, en français, *Tickets* > *Liste* > *Nouveau ticket*) :



C'est l'interface de composition qui se présente.



*Ajouter un titre pour le ticket dans le champ « **Titre** ».*

Spécifier en quelques mots la demande dans le champ « **Description** ». Il est possible d'utiliser la barre de mise en forme pour insérer du code source (javascript, html, etc), joindre des fichiers et images, etc.

Une fois terminé, cliquer sur le bouton « **Create issue** » enregistre la demande, une nouvelle page s'affiche et confirme la création du ticket.

Bonnes pratiques pour créer un ticket

Un ticket avec un message clair et détaillé facilite son traitement et le rend plus rapide et efficace.

Pour ce faire, il faut suivre quelques recommandations :

- Une seule demande par ticket : cela permet de suivre clairement le traitement du ticket et de garder un historique propre.
- Une description simple, claire et efficace : cela permet d'économiser du temps d'échange. Par exemple, dans le cas d'un bug, il est utile de détailler les étapes pour reproduire le bug, de joindre des captures d'écran ainsi que tout autre élément pouvant permettre aux développeurs d'appréhender plus complètement la situation.
- Joindre une capture d'écran pour illustrer le problème / la demande.
- Indiquer comment reproduire le problème dans le cas d'un bug.
- Indiquer dans quelles circonstances le problème arrive.

Ci-joint deux exemples de tickets de signalement d'un bug et de proposition de nouvelle fonctionnalité.

terristory › terristory › Issues › #1529

Closed Issue created 10 months ago by **Vincent Wawrzyniak** Developer Reopen issue

Bug filtres changement territoire

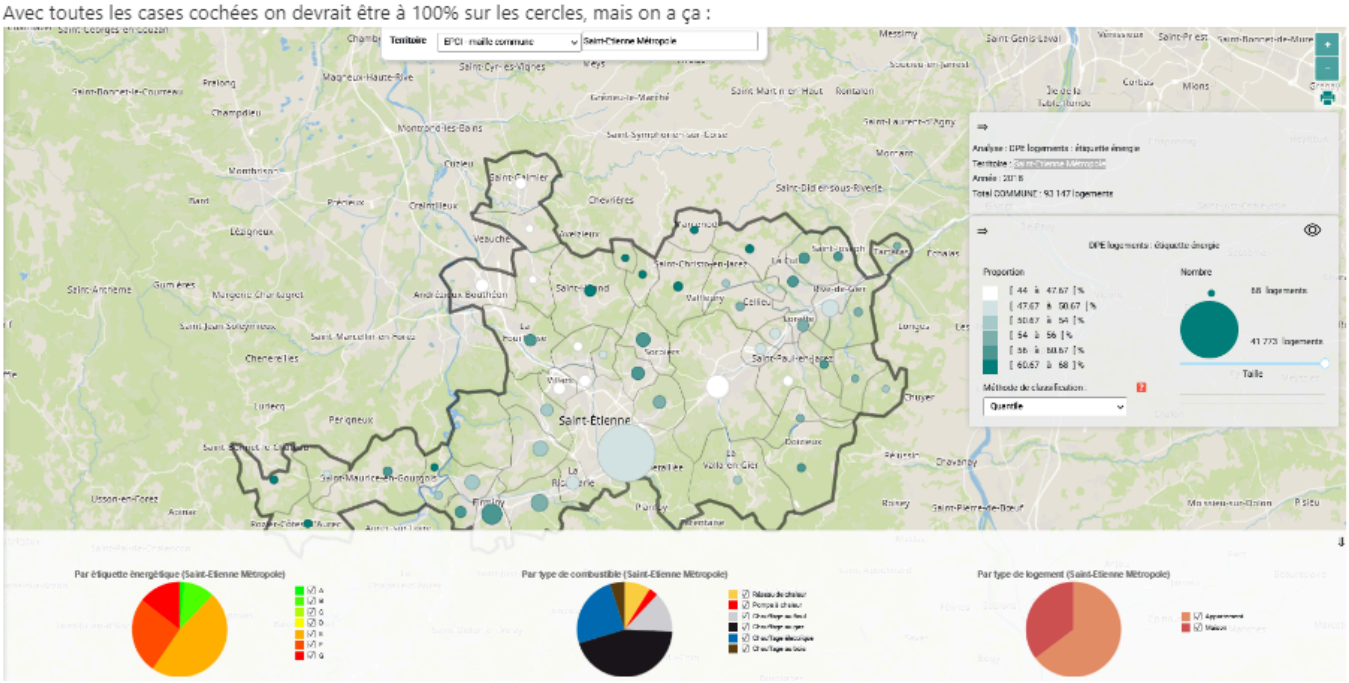
Avec @yravaud , on s'est rendu compte de quelques bugs avec les filtres lorsqu'on change de . On a l'impression qu'il y a un décalage entre l'affichage du filtre (front) et le filtre réel appliqué sur les données (back).

Je donne ici 2 exemples :

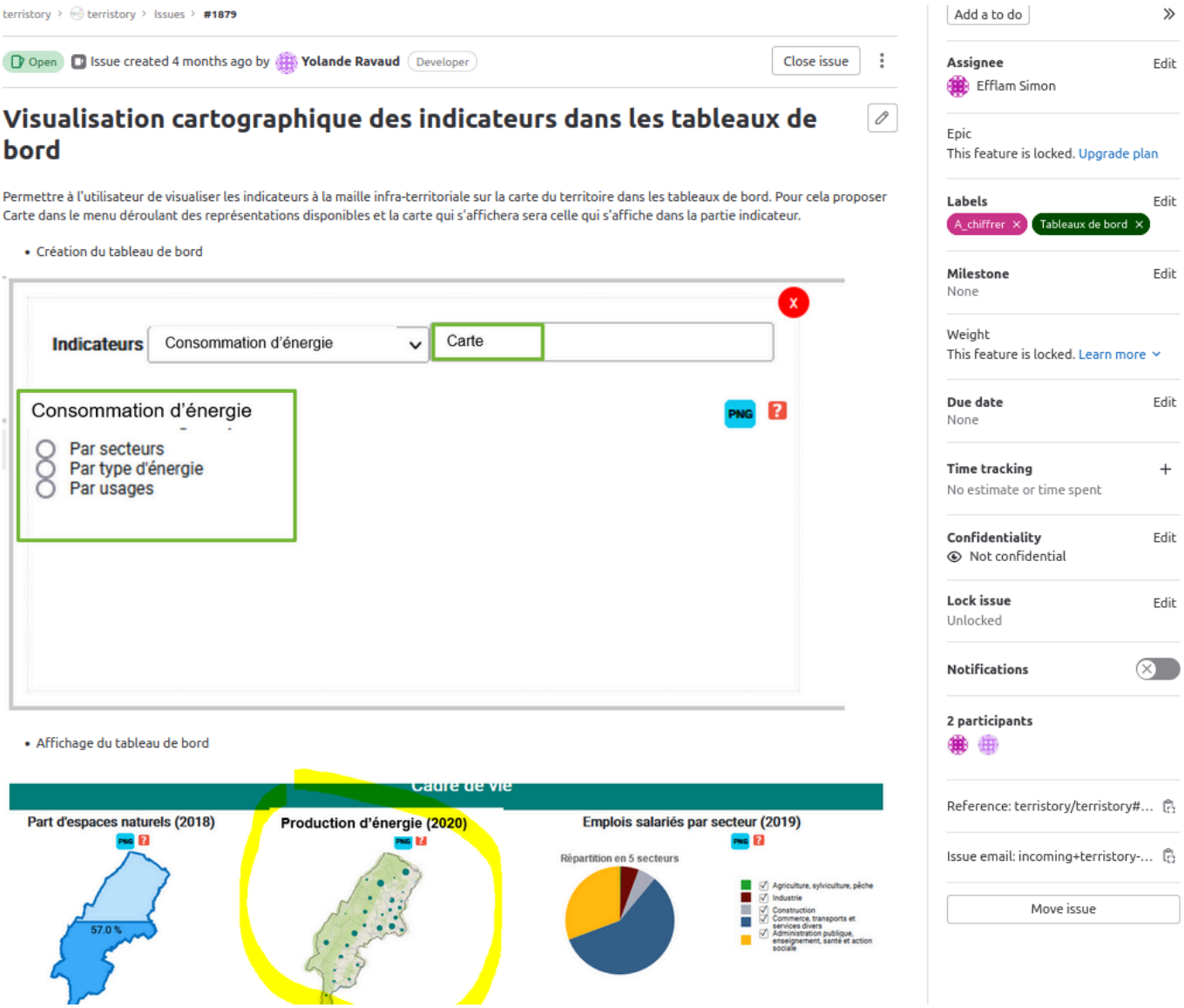
1. indicateur DPE :

- je vais sur CC Ambert Livradois Forez
- je sélectionne l'indicateur DPE logements : étiquette énergie
- je filtre pour enlever la classe D
- je vais sur Saint-Etienne Métropole -> en fait c'est la classe C qui est décochée (car sur Ambert il n'y a pas de classe A)
- je coche C -> ça fait baisser le nb de logements alors qu'en fait on en ajoute.

Avec toutes les cases cochées on devrait être à 100% sur les cercles, mais on a ça :



Exemple de ticket signalant un bug



Exemple de ticket de proposition de feature

En modifiant le code source

Pour contribuer au projet TerriSTORY® en tant que développeur, **GitLab** s'appuie sur les mêmes principes que les autres outils basés sur le système de gestion de versions décentralisé **Git**.

Le développeur, à la différence du simple utilisateur, modifie le code source de l'application. Cela peut se faire selon deux modalités différentes :

- Soit il ou elle fait partie de l'équipe de développement directement et il lui est alors possible de contribuer directement au code source du projet de référence.
- Dans le cas contraire, il est nécessaire de faire une copie du dépôt GitLab du projet TerriSTORY® (ou *fork*) et de toutes ses branches et de travailler indépendamment du code de référence.

Quel que soit la méthode utilisée, lors vous contribuez au code source de TerriSTORY®, vous en acceptez la licence (voir la page dédiée à la licence) et vous engagez à ne pas inclure de code qui enfreindrait cette licence. Pour cela, deux documents vous seront demandés au cours de la procédure avant d'aboutir à la fusion de votre code. AURA-EE, propriétaire du code source de la plateforme, vous demandera en effet de signer :

- un document certifiant que vous êtes bien à l'origine du code et que vous disposez des droits nécessaires. Ce document, appelé *Developer Certificate of Origin* (DCO), est accessible à l'adresse suivante : ...
- un document dans lequel vous cédez la propriété de votre code au propriétaire de TerriSTORY®, AURA-EE. Ce *Contributor License Agreement* (CLA) permet à la structure associative qu'est AURA-EE de se prémunir contre tout risque légal vis-à-vis des contributions faites par des tiers. Ce document est accessible à l'adresse suivante : ...

Ces documents devront être envoyés à l'adresse contact@terristory.fr.

Ajouter un développement sur le dépôt de référence distant

Après l'installation de l'application TerriSTORY® sur votre machine en local, vous pouvez commencer le développement.

Voici quelques commandes de base de Git :

- **La configuration de git :**

La configuration de votre git est importante, elle va permettre, entre autres, de définir votre identité dans vos commits.

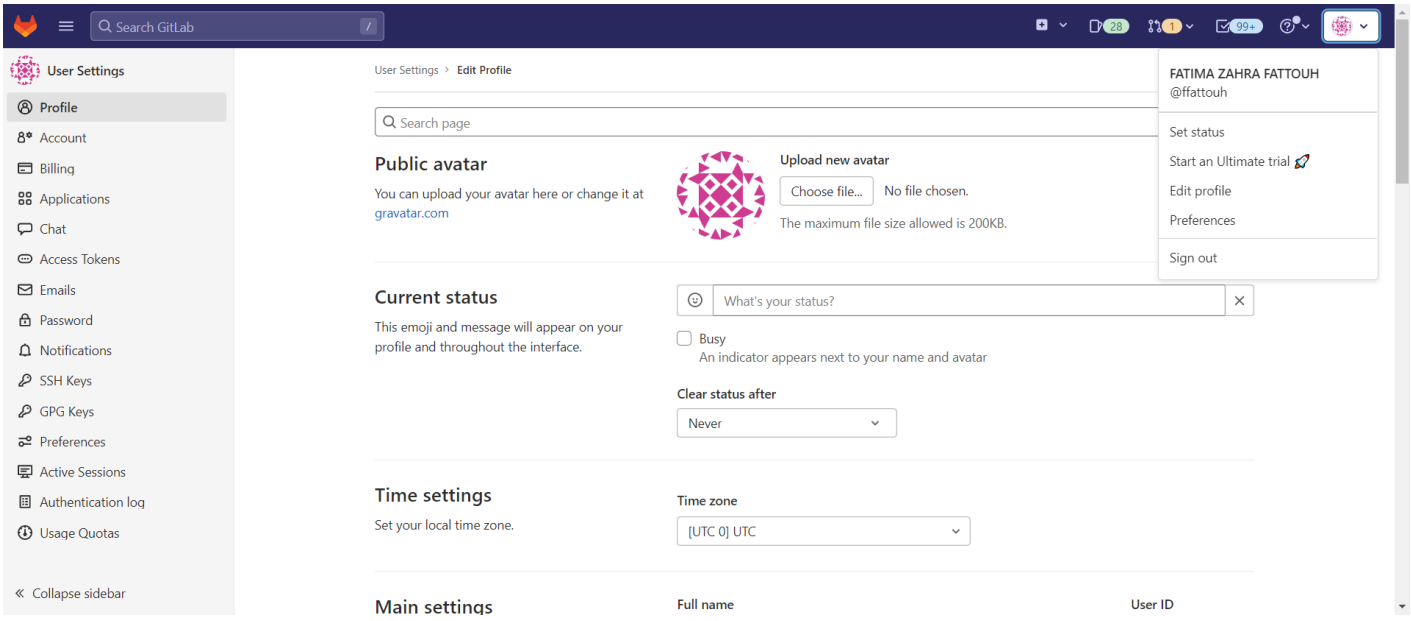
```
git config --global user.name "John Doe"
```

```
git config --global user.email johndoe@example.com
```

- **Ajouter une clé ssh**

Si vous souhaitez établir une connexion sécurisée à GitLab et pousser les branches sans taper votre identifiant/mot de passe, vous pouvez ajouter une clé ssh.

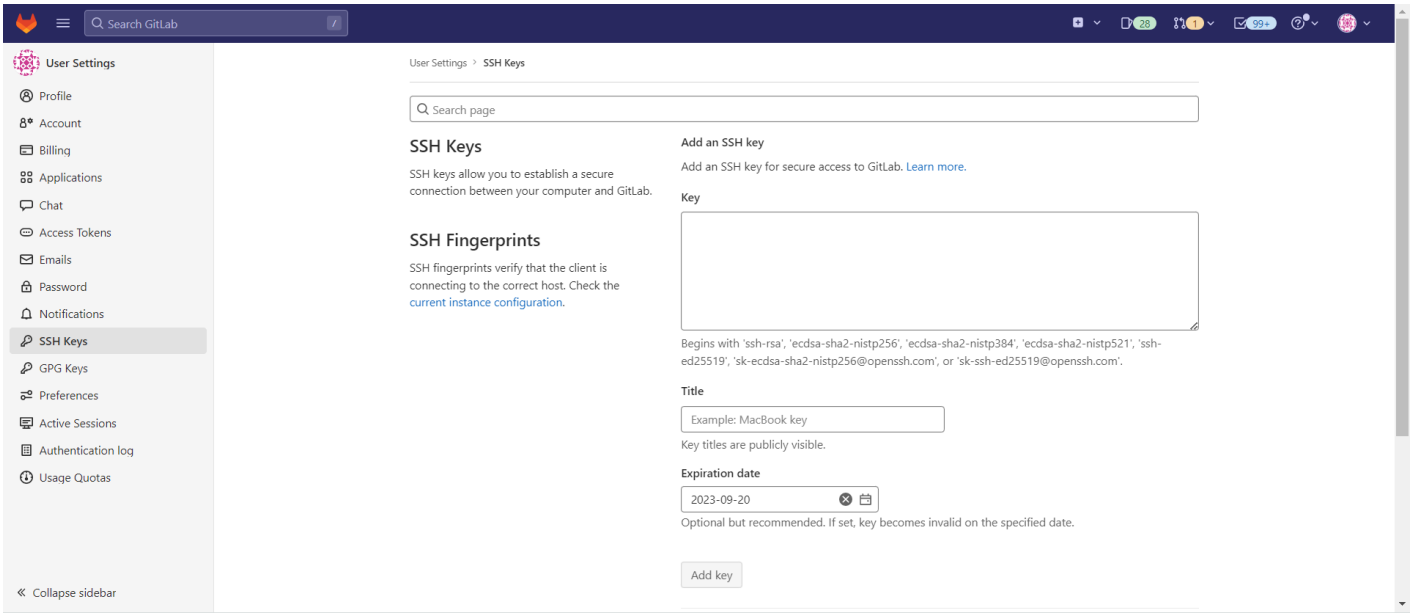
Pour ajouter votre clé ssh, Rendez-vous sur votre compte GitLab, cliquez sur votre avatar en haut à droite pour ouvrir le menu puis cliquez sur « **Edit profile** »



Page de configuration de profil

Collez votre clé ssh dans « **SSH Fingerprints** »

Une fois terminé, cliquez sur « **Add key** »



Ajouter une clé ssh

- **Créer une branche**

Créer une branche en local

```
git branch [nom_branche]
```

Se rendre sur cette nouvelle branche

```
git checkout [nom_branche]
```

Pour créer et se rendre directement sur une nouvelle branche, utiliser la commande suivante

```
git checkout -b [nom_branche]
```

Pour voir la liste des branches, taper la commande suivante

```
git branch
```

Pour voir l'état des modifications et la branche courante, taper la commande suivante

```
git status
```

- **Ajouter des modifications**

Une fois qu'un ensemble de modifications a été réalisé, il faut l'enregistrer dans l'historique Git. Pour cela nous devons utiliser les commandes git add et git commit qui permettent de rassembler les nouvelles versions de plusieurs fichiers dans un seul « paquet de modifications » (ou *commit*) qui marquera une nouvelle étape dans l'historique du projet :

```
git add [Nom_Fichier]
```

```
git commit -m « message du commit »
```

Pour afficher les différences entre vos modifications et votre dépôt, utiliser la commande suivante

```
git diff
```

Pour annuler les modifications sur un fichier, taper la commande suivante

```
git restore [Nom_Fichier]
```

- **Mettre à jour la branche courante avec le master ou une branche distante**

Lorsque vous travaillez avec une branche secondaire, Il peut arriver qu'entre temps, la branche initiale ait reçu plusieurs commits et ainsi, votre branche n'est plus synchronisée avec la branche d'origine.

Faire un rebase permet d'être à jour par rapport à la branche avec laquelle vous souhaitez travailler et ainsi d'éviter les conflits en cas de merge.

Pour faire un rebase, exécuter la séquence des commandes suivantes

```
git checkout master
```

```
git pull
```

```
git checkout [Nom_Branche_Courante]
```

```
git rebase master
```

- **Pousser la branche sur le dépôt distant**

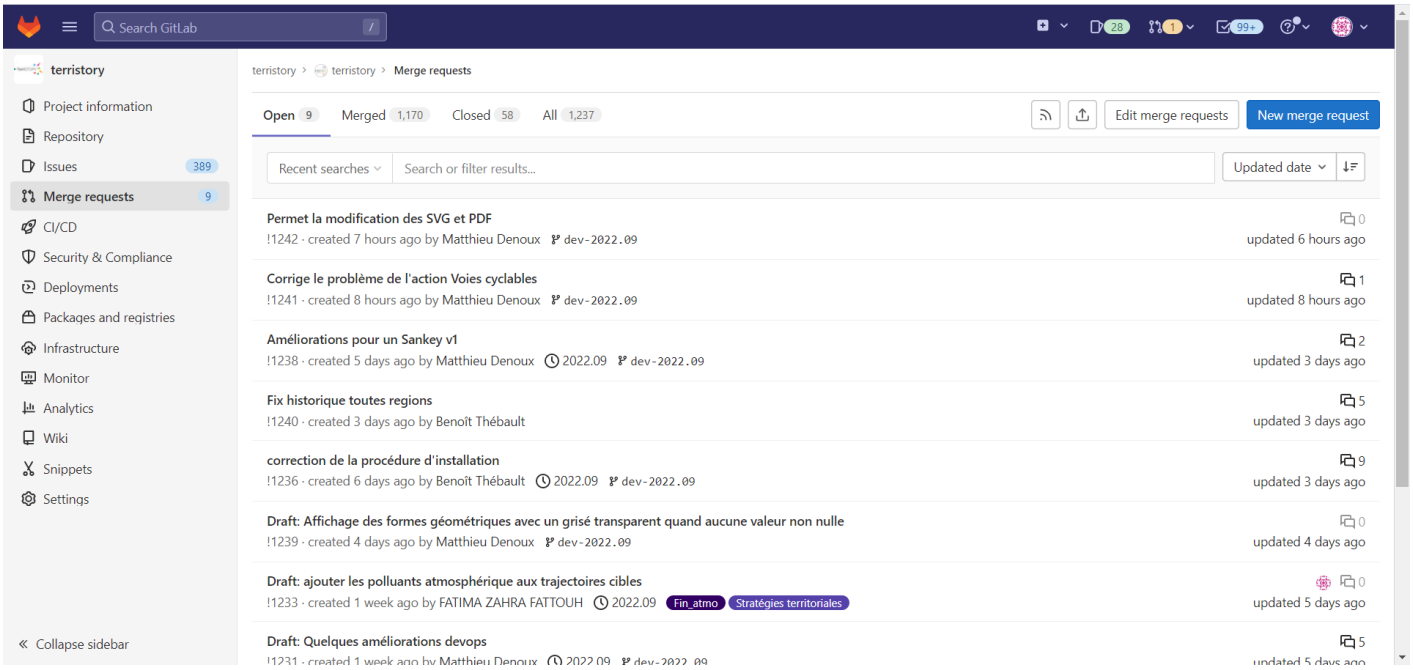
```
git push --set-upstream origin [Nom_Branche]
```

- **Fusionner une branche avec le master**

Fusionner une branche dans une branche de production ou la branche principale passe toujours par une **Merge Request** (MR).

Attention, avant de procéder à une telle demande, il faut s'assurer que notre version de l'application n'a pas créé de régression. Pour cela, un certain nombre de tests automatiques ont été mis en place. Il faut veiller à les exécuter avant de procéder à une merge request. Si les tests ne passent pas, la branche ne sera pas étudiée. On pourra se référer à la page dédiée aux tests pour plus d'informations.

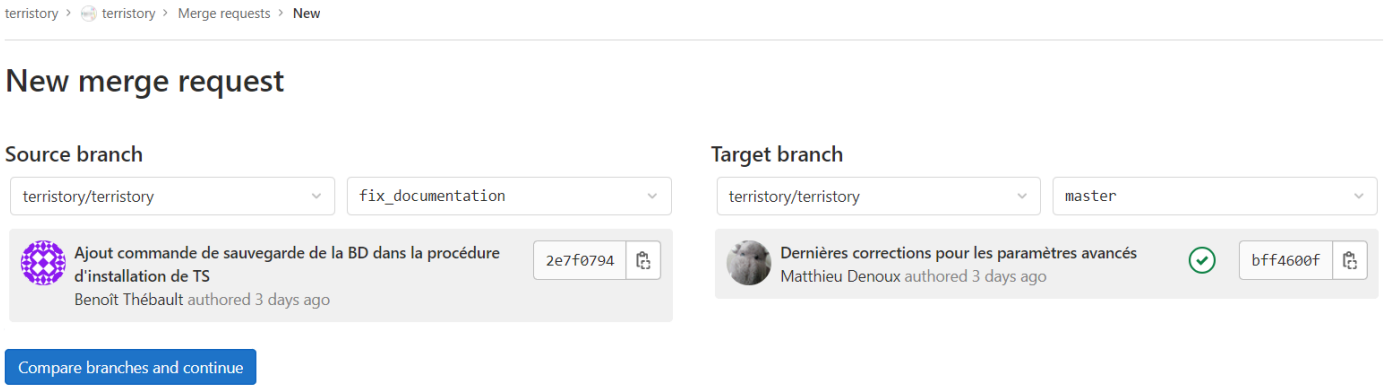
Pour créer une Merge Request, se rendre sur le tableau de bord de votre projet, accéder au menu « **Merge Requests** » et cliquer sur « **New merge request** »



Menu Merge Requests

Après avoir cliqué sur « **New merge request** », la page suivante s'affiche.

Renseigner le nom des branches *source* (*a [priori]{.underline}*celle sur laquelle vous avez travaillé) et *cible* (la branche que l'on souhaite mettre à jour) et cliquer ensuite sur « **Compare branches and continue** ».



Nouvelle demande de fusion

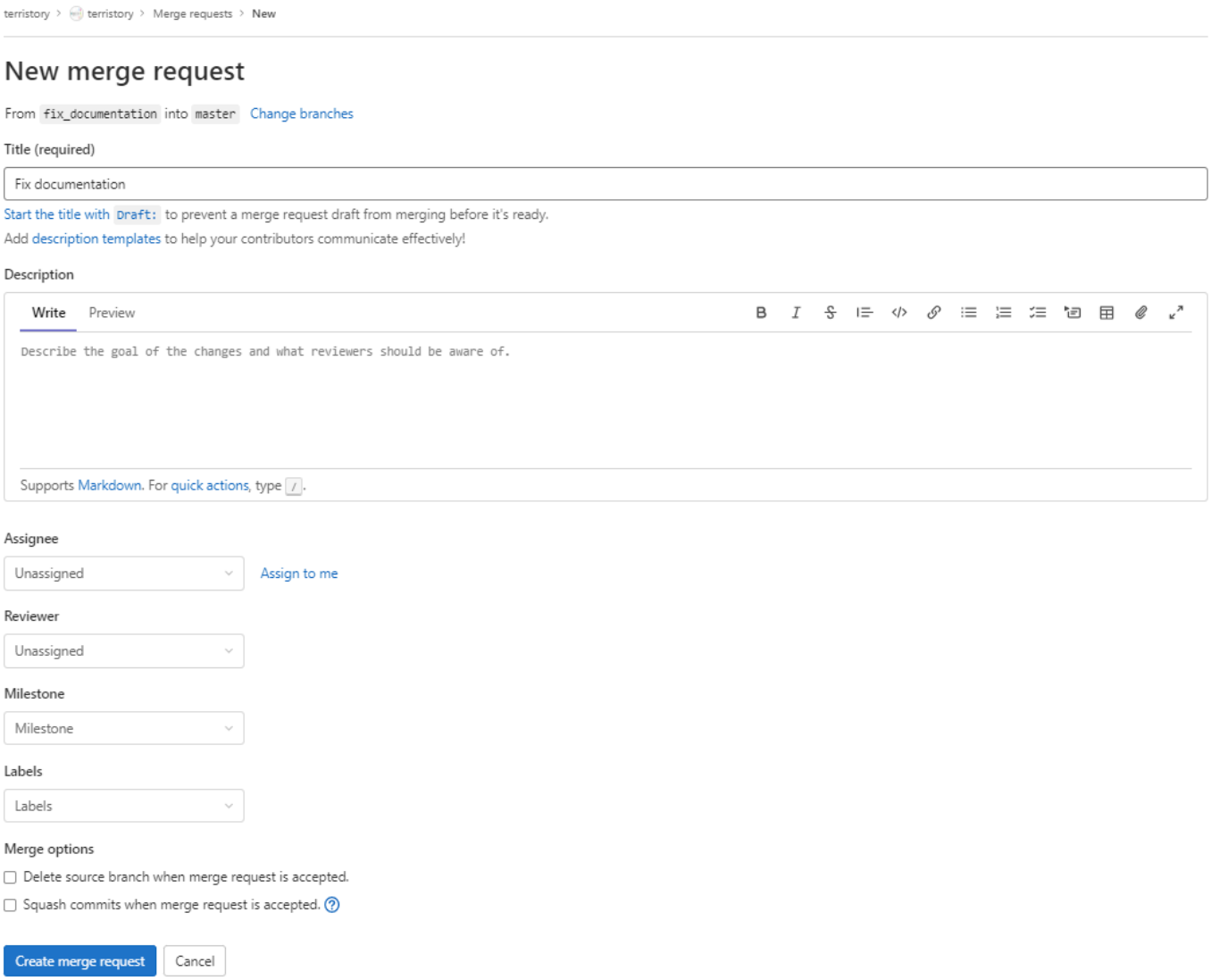
Cela ouvre la page de création d'une demande de fusion (Figure 11).

Ajouter un titre pour la merge request dans le champ « **Title** »

Spécifier en quelques mots, l'objet de la MR dans le champ « **Description** ».

Utiliser la barre de mise en forme, pour insérer du code source (javascript, html, etc), joindre des fichiers et images, etc.

Une fois terminé, cliquer sur le bouton « **Create merge request** », une nouvelle page s'affiche pour confirmer la création de la Merge Request (Figure 12).



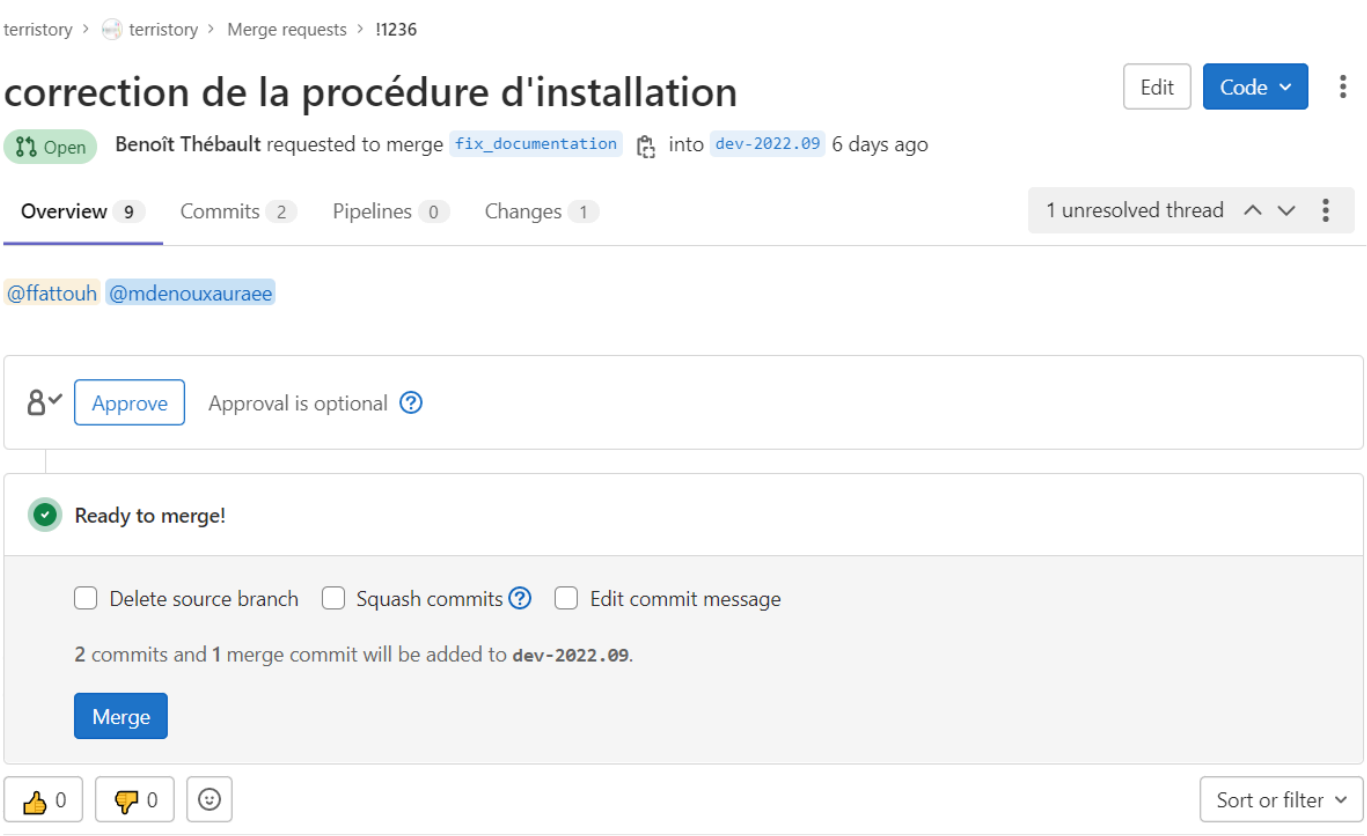
Page de composition d'un Merge Request

- **Révision et fusion du code**

Avant de fusionner la branche avec la branche principale, il faut d'abord attendre la relecture et le test du code par les autres développeurs concernés.

Une revue de code permet souvent de prévenir l'apparition de bugs, d'éviter les problèmes fonctionnels (mauvaise compréhension de la fonctionnalité demandée), de s'assurer de la conformité du code proposé par rapport à l'application globale, dans l'esprit (principes de développement respectés, fonctionnalité ajoutée en adéquation avec les besoins, etc.) et dans la lettre, et de tenir au courant les autres développeurs de l'évolution du logiciel.

Une fois que la branche a été validée par les autres développeurs, cliquer sur « **Merge** » pour fusionner la branche avec le code principal du projet.



Page de fusion de code

Bonnes pratiques et conventions sur Git

- Une branche par ticket
- Donner des noms significatifs aux branches
- Lorsque vous travaillez sur une branche secondaire en local, n'oubliez pas de faire fréquemment un rebase avec la branche master pour éviter les conflits au moment de fusion.
- Ajouter des messages des commits clairs et concis qui décrivent exactement les changements apportés.
- Faites des commits régulièrement (un commit = un problème résolu)
- Ne faites pas de commit d'un fichier de configuration ou d'un fichier contenant des données personnelles, des mots de passe ou des informations confidentielles.

Les conventions sont les suivantes sur TerriSTORY® :

- Les messages des commits doivent être en anglais.
- Les noms des branches doivent être en anglais.
- Les noms des fichiers doivent être en anglais.
- Les noms des branches doivent commencer par « **fix-xxx** » s'il s'agit d'une résolution d'un bug ou par « **feature-xxx** » s'il s'agit d'une nouvelle fonctionnalité.
- Les noms des branches « **dev-xxx** », « **test-xxx** » et « **prod-xxx** » correspondent à des noms réservés aux branches de développement utilisées pour la mise en production des versions de TerriSTORY®.
- La branche « **master** » est également réservée.

- La mise à jour des branches *dev/test/prod/master* est protégée et réservée à l'équipe de développement de TerriSTORY®. Aucun autre membre contribuant au projet ne pourra effectuer ces opérations.

Pour corriger des bugs qui empêchent le fonctionnement normal de l'application TerriSTORY® sur le code de production. On doit créer une branche pour corriger le problème qui part de master et commence par le mot clé **fix-** ou **hotfix-** avant de contenir la dénomination de la branche corrigée (par exemple, *hotfix-prod-2022.08*).

Conventions de code

Voici quelques conventions pour un code clair, homogène et lisible :

EN PYTHON

- **Le code de TerriSTORY® tente de respecter la nomenclature PEP8**
- Ainsi, les noms de variables/fonctions et les noms des fichiers doivent être définis en **snake_case** : en lettres minuscules, avec les mots séparés par des caractères _ (underscore). Exemple : `nom_fonction(nom_variable)`
- L'indentation est faite en utilisant 4 espaces.
- La comparaison avec *None* se fait avec *is* / *is not* et non == / !=
- Privilégier `isinstance()` à une comparaison entre `type()`
- Il faut ajouter des commentaires de type string Documentation au début des fonctions Python, avec la convention Numpy comme décrit dans cette documentation : numpydoc.readthedocs.io/en/latest/format.html.
  - Une ou plusieurs lignes décrivant la fonction et son utilisation
  - Le titre "Parameters", suivi de chaque paramètre avec son type et une description éventuelle
  - Le titre "Returns", suivi du type de retour et une description éventuelle

```
def get_user_by_login(login):
    """Récupère un utilisateur/utilisatrice en fonction de son login.

    Parameters
    ----------
    login : str
        Login de l'utilisateur / utilisatrice, i.e. son email

    Returns
    -------
    User
        L'objet contenant les informations utiles sur l'utilisateur.trice
    """
    query = "select mail, password, actif from utilisateur where mail = $1"
    rset = await fetch(query, login)
    if rset and rset[0]["actif"]:
        return User(login=rset[0]["mail"], password=rset[0]["password"])
    return None
```

- il est recommandé d'installer l'outil black pour formater automatiquement le code source python. Il est généralement possible, avec un bon éditeur de texte, que le formatage soit fait automatiquement à chaque enregistrement.
- de même, il est fortement conseillé d'ajouter la librarie pre-commit à votre environnement python pour que le code source que vous souhaiteriez ajouter soit bien formaté avant chaque commit.

EN JAVASCRIPT

- Un nom de variable/fonction en Javascript doit respecter la convention **lowerCamelCase**: les mots sont collés ensemble en en mettant en majuscule la première lettre de chaque mot, sauf le premier. Exemples : `nomFonction(nomVariable)`
- Les noms des fichiers JavaScript et les noms de classes doivent respecter la convention **UpperCamelCase**, en commençant le nom par une majuscule ainsi que chaque mot dont il est composé. Exemples : *NomFichier.js*, `NomClasse`
- Laisser un espace avant et après les opérateurs (+, -, =, ===, !==).
- Ajouter une ligne vide entre les fonctions.
- L'indentation est faite en utilisant 4 espaces.
- Ajouter des commentaires de documentation en dehors des fonctions Javascript entre `/**` et `*/`, en utilisant la convention JSDoc documentée ici : jsdoc.app. Les tags `@param` et `@returns` sont utilisés respectivement pour les paramètres et la valeur de retour, suivi du type entre accolades `{}`, du nom du paramètre si c'en est un et enfin d'une description.

```
/**
 * Convert a string to a normalized string lowercase without unusual characters.
 *
 * @param {String} str - Input string.
 * @returns {String} The string without unusual characters.
 */
function normalizeString(str) {
    return str
        .toLowerCase()
        .normalize("NFD")
        .replace(/\p{Diacritic}/gu, "");
}
```

- Il est recommandé d'installer l'outil prettier pour formater automatiquement le code source javascript et CSS. Il est généralement possible, avec un bon éditeur de texte, que le formatage soit fait automatiquement à chaque enregistrement.
- Privilégier les opérateurs `===` et `!==` plutôt que `==` et `!=`, pour éviter les conversions de type non désirées.
- Ne pas utiliser le mot-clé `var` : utiliser `const` quand c'est possible et `let` lorsque nécessaire, afin de contrôler la portée des variables.

DANS TOUS LES LANGAGES

- Les noms de variables/fonctions/types doivent être en anglais et significatifs : pas de variable d'une seule lettre, pas d'initiales hors sigles connus, pas de nom vide de sens (data, value...).
- Itérer sur les valeurs des tableaux/dictionnaires plutôt que sur leurs indices quand c'est possible. Exemples :
  - (python) `for element in array` plutôt que `for index in range(len(array))`
  - (javascript) `for (const element of array)` plutôt que `for (const index in array)`, ou pire `for (let i = 0; i < array.length; i++)`
- Privilégier les early-return aux if-else pour limiter l'indentation. Exemple : Utiliser

```
def my_function(args):
    if not check_validity(args):
        return default_value
    # important part of the function
    ...
```

plutôt que

```
def my_function(args):
    if check_validity(args):
        # important part of the function
        ...
    else:
        return default_value
```

Organisation des contributions et des développeurs

Par défaut, nous invitons toute personne à contribuer par la création de tickets et l'utilisation des *forks* et des *pull requests*.

L'intégration à l'équipe gitlab de développement de TerriSTORY® ainsi que l'attribution de rôles particuliers (*Reporter*, *Developer*, *Maintainer*, *Owner*, cf. la page de la documentation gitlab) sont pour l'instant réservés à l'équipe de développement du propriétaire de TerriSTORY® (AURA-EE) et aux acteurs du Consortium encadrant le développement de TerriSTORY®. Dès lors qu'un ticket touche à des données confidentielles ou à des problématiques sensibles, il est donc possible que ce ticket soit rendu confidentiel au moment de la création ou par la suite afin de garantir la confidentialité des échanges qui lui seront associés. Une fois un ticket rendu confidentiel, seule la personne l'ayant créé, toutes les personnes mentionnés dans le ticket ou associés au ticket (assignées, reviewers, etc.) ainsi que les utilisateurs enregistrés comme membres de l'équipe TerriSTORY® dans le projet gitlab et ayant un droit d'accès suffisant, pourront le consulter.

Dans le cas d'une personne contribuant régulièrement, il est possible pour elle de demander à être intégrée dans l'équipe d'utilisateurs ayant des droits sur le dépôt TerriSTORY® sur gitlab. Cette demande sera étudiée par AURA-EE, propriétaire de TerriSTORY®. Dans le cas éventuel où cette requête rencontre un accueil favorable du propriétaire de TerriSTORY®, il sera potentiellement envisagé que la personne demandeuse doive signer un accord de non divulgation (ou NDA) pour garantir la confidentialité des échanges auquel son nouveau statut lui donnera accès.

L'état des tickets

Les tickets sont classés dans différentes catégories:

- « Open » dans le cas des tickets destinés à faire avancer une réflexion,
- « Planned » à définir,
- « To do » pour ceux qui doivent faire l'objet de développements,
- « In progress » lorsque ces développements sont en cours,
- « Pour passage sur dev/test/prod » lorsque les développements sont à prêts à avancer dans le processus de déploiement,
- « Vérification sur dev/test/prod » lorsqu'une validation doit être faite sur un des serveurs de déploiement et enfin,

- « Closed » lorsqu'ils sont achevés et validés.

Il est également possible d'être notifié par courriel lorsqu'une opération est effectuée sur le projet. Il existe plusieurs niveaux d'abonnement en fonction de la manière dont vous souhaitez être tenu informé.e. Pour accéder aux tickets, sélectionnez «issues» puis «Boards».

Les tickets changent ensuite de statut au cours du processus de développement. En effet, lorsque les dernières modifications destinées à résoudre un ticket sont prêtes à être intégrées à la branche principale (c'est-à-dire que la branche est en *Code review*), on déplace le ticket en question dans « *Pour passage sur dev* ». Une fois l'accord d'autres développeurs obtenu, on fusionne le code dans la branche principale et on déplace le ticket dans la colonne « *Vérification sur dev* ». On procède de même pour les serveurs de test et de prod successivement. Lorsque des vérifications ont permis de valider la résolution du ticket, on le ferme (colonne « *Closed* »).

Notez qu'il est possible de notifier un contributeur par mail en mentionnant dans n'importe quelle partie de la page (description de tickets, ou fil de discussions) son nom d'utilisateur GitLab précédé du signe `@` comme dans `@le_pseudonyme_de_la_personne` par exemple.

Documents de contribution

- Certificat d'origine à l'usage des développeurs (Developer's Certificate of Origin)
- Accord de licence de contributeur