Documentation

[Jira 1](#_Toc167366147)

[GitHub 2](#_Toc167366148)

[Bonnes pratiques 3](#_Toc167366149)

[ENV 3](#_Toc167366150)

# Jira

Chaque colonne du Jira correspond à un état d’avancement d’une tâche.  
Il faut respecter l’ordre du tableau (de gauche à droite) et placer les tâches dans les bonnes colonnes à chaque étape.  
Ci-dessous, voici la liste des différentes colonnes et à quoi elles correspondent.   
  
N.B.  
Penser à bien respecter les indications pour chaque colonne pour que l’organisation du projet reste la même tout du long.  
  
Colonnes :

* **A spécifier**

C’est la première colonne du tableau, elle correspond donc à la création des tâches.  
C’est aussi à cette étape qu’on explicite la tâche (dans sa description), le travail qu’il faudrait réaliser, les détails spéciaux etc.  
Les tâches peuvent être créées par une seule personne cependant il faut qu’elles soient spécifiées par tout le groupe.  
Pour chaque tâche de dev ajoutée à cette colonne, il faut créer la branche correspondante sur le repository en respectant la nomenclature (cf. [GitHub](#_Github)) à cette étape et l’annoter dans la description.

* **A planifier**

Une fois que la tâche a été créée et spécifiée elle passe dans cette colonne pour qu’elle soit attribuée à quelqu’un. S’il reste des tâches à planifier dans cette colonne, n’importe qui peut s’en attribuer une, sans consulter le reste du groupe.

* **En attente d’exécution**

Comme son nom l’indique, cette colonne correspond aux tâches qui sont prêtes à être réalisées mais qui sont mise en attente, le temps de terminer les autres tâches déjà en cours.

* **En cours d’exécution**

Colonnes des tâches en cours

* **A merger / A recetter**

Une fois que la tâche est considérée comme « prête », elle doit être mise dans cette colonne.  
Une fois par semaine, il y aura une review du Jira pour faire un point sur les tâches et pour fusionner ou non, celles qu’on considère comme abouties, au projet principal.  
  
La personne qui se sera occupée de la tâche, devra alors la présenter au groupe ainsi qu’expliquer son code, dans le cadre d’une tâche de développement, pour que tout le monde puisse comprendre. Le groupe pourra alors tester les modifications apportées pour s’assurer que tout est ok.  
  
Enfin, pour que la tâche soit mergée et passée dans la dernière colonne, il faut qu’elle respecte certains critères :  
 - Un code commenté   
 - Un code factorisé  
 - Une nomenclature respectée (cf. [bonnes pratiques](#_Bonnes_pratiques))

* **Terminé**

Une fois que la tâche à été testée et mergée, on peut la passer dans cette colonne pour garder un historique.

# GitHub

Pour que le repository soit le plus clean possible et qu’on s’y retrouve, il faudra suivre certaines règles.  
  
Dans un premier temps, j’ai bloqué les push directs à la branche main car elle correspond à notre branche de production. Il faudra donc obligatoirement faire une pull request à chaque fois qu’une tâche sera terminée, de sa branche sur la branche main.   
  
Comme indiqué dans la partie Jira, une fois par semaine, nous passerons sur chacune des pull request pour revoir le code et l’accepter ou non.

Enfin, il est important de respecter une nomenclature commune pour nos branches, un peu comme un GitFlow pour bien les différencier.  
  
Il faudra que nos branches commencent par un préfixe qui l’a décrit le mieux parmi ces 2 préfixes:   
- feat (pour un ajout de fonctionnalité)  
- bugfix (pour la résolution d’un bug)  
Le tout suivi du nom de la branche en Snake Case et en anglais.  
ex : feat/add\_calendar\_provider  
  
N.B.   
Penser à bien merge la branche main dans la branche avant de créer une pull request pour éviter de perdre 30min pendant les réunions à gérer les conflits.

# Bonnes pratiques

Dans cette partie je vais vous décrire quelques bonnes pratiques à respecter pour que le code soit le plus compréhensible possible mais aussi pour faciliter la vie dans la rédaction de ce dernier.  
  
Afin de ne pas se perdre sur son propre code ou celui des autres il est important d’utiliser une nomenclature commune, de la même manière que pour les branches du GitHub.  
Elle sera à utiliser sur les variables, nom de fichiers, nom de fonction, etc.  
  
Dans un premier temps, le code doit être en anglais, c’est le plus important.  
Pour les variables et pour les fonctions il faut utiliser du Camel Case  
ex : $apartment / setApartment()

Pour ce qui est des noms de fichiers, il ne sert à rien de créer directement les fichiers à la main, car php artisan est fait pour ça.  
  
Il peut créer toute sorte de fichier directement avec la bonne nomenclature et déjà préremplie : controller, vue, model, migration…  
La commande la plus utile est surement : php artisan make:model Nomdumodel -mrc  
-mrc permet de créer aussi une migration, un controller et faire en sorte que ce soit un controller de ressource, c’est-à-dire avec toutes les fonctions utiles (CRUD) déjà présente.  
  
Il y a beaucoup d’autre commande pour se faciliter la vie, donc ne pas hésiter à regarder la documentation : <https://laravel.com/docs/11.x/artisan>

# ENV

<https://github.com/NathanPradelle/projet_annuel/blob/main/README.md>