# JO 2024 - Process & Documentation



---

## 1. Choix méthodologiques

#### Méthodologie Agile

Pour ce projet, j'ai choisi d'adopter la méthodologie **Scrum** plutôt que Kanban. Ce choix s'est imposé naturellement, car Scrum offre un cadre à la fois structuré et flexible, particulièrement adapté à un projet de développement mené seul mais avec des échéances précises.

L'un des principaux avantages de Scrum, c'est la **prévisibilité**. Le travail est découpé en sprints, des cycles de durée fixe qui permettent de planifier des objectifs clairs à court terme. Cette approche m'a aidé à garder une vision globale du projet tout en avançant pas à pas, avec des livrables concrets à la fin de chaque itération.

Au-delà de la planification, Scrum m'impose aussi de la réflexion. Les moments clés (cérémonies) comme la planification, revue et rétrospective me permettent de garder le rythme.

Un autre aspect que j'apprécie dans Scrum, c'est la mesure de la progression. En suivant la vélocité (nombre de story points complétés par sprint) je peux évaluer ma capacité réelle de travail et ajuster mon organisation.

À l'inverse, j'ai écarté **Kanban**, car ce cadre se prête davantage à un flux continu sans échéances fixes, par exemple pour de la maintenance ou du support. Dans mon cas, où le projet devait respecter un calendrier précis avec des livrables planifiés, Scrum m'a semblé plus pertinent.

#### Choix de l'outil

Côté outils, j'ai choisi de travailler avec **Jira**. Au-delà de sa notoriété, c'est un outil extrêmement complet et personnalisable. Il m'a permis de configurer mes propres workflows, types de tickets et champs, de façon à coller parfaitement à mon mode d'organisation. C'est aussi l'outil que j'utilise dans mon alternance.

Jira intègre nativement la logique Scrum : backlog produit, planification des sprints, tableaux de bord, burn-down charts, rapports de vélocité... Tout est prévu pour faciliter la mise en place d'un vrai cadre agile sans avoir à tout configurer soi-même.

Enfin, Jira s'intègre naturellement avec d'autres outils comme Confluence pour la documentation, GitHub pour le versioning du code. Cette interconnexion en fait une excellente base pour reproduire un environnement de travail professionnel complet, fidèle à ce qu'on trouve dans la majorité des équipes de développement.

### 2. Utilisation de l'espace de travail

#### Contexte de travail en solo

Bien que l'espace soit géré uniquement par moi, l'approche Scrum reste pertinente et nécessite des adaptations spécifiques :

**Auto-analyse et rigueur dans la méthodo:** Le travail en solo impose une discipline personnelle. Les cérémonies Scrum, même simplifiées, servent de barrière contre la dérive du projet. L'auto-analyse remplace les discussions d'équipe mais doit rester structurée et documentée.

**Documentation** : En l'absence d'équipe, chaque décision doit être documentée pour tracer le raisonnement et pouvoir justifier les choix lors de l'évaluation. Les commentaires Jira servent de journal de bord du projet.

#### Gestion du backlog

**Product Backlog** : Le backlog produit contient l'ensemble des Epics et Stories / Tasks identifiées pour le projet, classées par priorité selon la méthode MoSCoW (Must have, Should

have, Could have, Won't have) pour les Stories et classique (Highest, High, Medium, Low, Lowest) pour les Tasks. Ce backlog est régulièrement revu pour s'assurer que les éléments prioritaires sont suffisamment détaillés pour être inclus dans un sprint.

**Definition of Ready :** Avant d'être sélectionnée pour un sprint, une Story doit respecter les critères suivants :

- Le champ "DoR validée" (Definition of Ready) doit être coché
- L'estimation en story points doit être renseignée
- Les critères d'acceptation doivent être clairement définis
- Les dépendances techniques doivent être identifiées
- Les stories bloquantes doivent être en “Done”
- Les sub-taks doivent avoir été définies suite à l’analyse

**Backlog refinement :** Une session hebdomadaire (en dehors des sprints) permet de :

- Clarifier les Stories à venir
- Estimer les éléments non chiffrés
- Identifier les dépendances
- Réévaluer les priorités si nécessaire

#### Cérémonies Scrum pour un développeur solo

**Sprint Planning** : 1-2 heures en début de sprint

Objectifs :

- Définir le Sprint Goal
- Sélectionner les Stories / Tasks du product backlog en fonction de la vélocité passée
- Décomposer les Stories / Tasks en Sub-tasks techniques
- S'assurer que la charge de travail est réaliste

Livrables :

- Sprint backlog constitué (Stories + Tasks)
- Sprint goal documenté dans Jira
- Sub-tasks créées pour chaque Story

**Sprint Review**: 1 heure en fin de sprint

Objectifs :

- Vérification des fonctionnalités terminées (DoD respectée)
- Validation que le Sprint Goal est atteint

- Documentation

Livrables :

- Mise à jour du statut des Stories
- Notes sur ce qui a été livré vs prévu

**Sprint Retrospective** : 30-45 minutes après la Sprint Review

Objectifs :

- Identifier ce qui a bien fonctionné
- Identifier ce qui doit être amélioré
- Définir 1-2 actions concrètes pour le prochain sprint
- Analyser la vélocité

Livrables :

- Document de rétrospective dans Confluence
- Actions d'amélioration créées comme Bug dans Jira
- Mise à jour de la vélocité moyenne

**Stand-up quotidien adapté** En solo, le stand-up quotidien devient une revue personnelle de 5-10 minutes chaque matin :

- Qu’est-ce que j’ai accompli hier ?
- Que vais-je faire aujourd'hui ?
- Quels obstacles m'empêchent d'avancer ?

#### Durée et structure des sprints

**Durée standard : 3 semaines** Les sprints ont une durée fixe de 3 semaines de 4 jours(Jeudi au dimanche). Cette durée offre un bon équilibre entre :

- Flexibilité pour s'adapter aux changements
- Stabilité pour planifier et terminer des fonctionnalités complètes
- Rythme compatible avec le planning de cours

**Structure d'un sprint type**

Jour 1 :

- Sprint Planning (matin)
- Début du développement

Jours 2-11 :

- Développement quotidien
- Stand-up personnel
- Mise à jour continue de Jira

Jour 12 :

- Finalisation et tests
- Sprint Review (après-midi)
- Sprint Retrospective
- Préparation du prochain sprint

---

### 3. Architecture et découpage des éléments

La hiérarchie des éléments dans Jira suit une structure en trois niveaux, chacun ayant un rôle et une granularité spécifiques.

#### Niveau 1 : Epics

Les Epics représentent de grands ensembles fonctionnels ou des thématiques majeures du projet. Elles regroupent plusieurs Stories / Tasks qui contribuent à un objectif commun de haut niveau.

Caractéristiques :

- Durée de vie : Plusieurs sprints
- Granularité : Une Epic peut représenter 50-200 story points
- Portée : Une fonctionnalité majeure ou un module complet

Exemples concrets :

- "Système d'authentification utilisateur"
- "Module de gestion des offres"
- "Interface de reporting et tableaux de bord"
- "API REST pour intégrations externes"

Une Epic est créée quand :

- Le travail nécessite plusieurs Stories interdépendantes
- L'objectif couvre une valeur métier significative
- Le travail ne peut pas être livré en un seul sprint

#### Niveau 2 : Stories et Tasks

À ce niveau, on distingue deux types d'éléments selon leur nature.

#### Stories (User Stories) - Développement

Les Stories représentent des fonctionnalités ou des évolutions du point de vue utilisateur. Elles correspondent à du travail de développement créant de la valeur métier directe.

Elles suivent généralement le format : "En tant que [rôle], je veux [action] afin de [bénéfice]"

Caractéristiques :

- Durée de vie : 1 sprint (idéalement terminée dans le sprint où elle est prise)
- Granularité : 1-8 story points (méthode Fibonacci)
- Liée à : Une Epic parente
- Priorité : MoSCoW (Must, Should, Could, Won't)

L'estimation des story points suit la suite de Fibonacci (1, 2, 3, 5, 8, 13, ...) :

- 1-2 points : Tâche simple, bien comprise, peu de risques
- 3-5 points : Tâche moyenne, quelques inconnues
- 8 points : Tâche complexe, plusieurs incertitudes
- 13 points et + : Story trop grosse, à découper

Exemples concrets :

- "En tant qu'utilisateur, je veux pouvoir réinitialiser mon mot de passe par email afin de retrouver l'accès à mon compte" (3 points)
- "En tant qu'administrateur, je veux pouvoir exporter les données des offres au format Excel afin de les analyser" (5 points)

#### Tasks - Intégration et support

Les Tasks représentent du travail technique qui ne génère pas directement de valeur utilisateur visible, mais qui est nécessaire au projet : configuration, intégration, documentation technique, mise en place d'infrastructure.

Caractéristiques :

- Durée de vie : Variable (peut être dans ou hors sprint)
- Granularité : Plus flexible que les Stories, estimée en temps (Original estimate)
- Liée à : Peut être liée à une Epic ou autonome
- Priorité : Highest, High, Medium, Low, Lowest

Différence avec les Stories :

- Les Tasks ne créent pas de valeur utilisateur directe
- Elles ne sont pas estimées en story points mais peuvent avoir une estimation en temps
- Elles ont un système de priorité différent (5 niveaux au lieu de MoSCoW)

Exemples concrets :

- "Configuration du container de production"
- "Documentation de l'API REST dans Swagger"
- "Configuration BDD avec PostgreSQL"
- "Mise en place de l'environnement de développement"
- "Résolution d'une dette technique : refactoring du module de logging"

Utiliser une Story quand : Le travail produit une fonctionnalité visible par l'utilisateur final / Utiliser une Task quand : Le travail est technique, sur l'infra ou de support

#### Niveau 3 : Sub-tasks (Sous-tâches)

Les Sub-tasks représentent le découpage technique granulaire d'une Story ou d'une Task. Elles correspondent aux étapes concrètes de réalisation.

Caractéristiques

- Granularité : Très fine, une Sub-task doit être réalisable en une session de travail
- Liée à : Attachée à une Story ou Task parent
- Estimation : Temps estimé (Original estimate en heures)

Une Sub-task est créée pour représenter :

- Une étape technique distincte (front-end / back-end / tests)
- Un composant spécifique à développer
- Une phase d'implémentation (conception / code / tests / revue)

**Exemples concrets** Pour une Story "Réinitialisation de mot de passe" :

- Sub-task 1 : "Créer l'interface utilisateur du formulaire de réinitialisation" (2h)
- Sub-task 2 : "Développer l'API endpoint POST /api/reset-password" (3h)
- Sub-task 3 : "Implémenter l'envoi d'email avec token de réinitialisation" (2h)
- Sub-task 4 : "Créer les tests unitaires du service de réinitialisation" (2h)
- Sub-task 5 : "Tester le flux complet en environnement de développement" (1h)

**Règles de découpage**

- Maximum 8-10 Sub-tasks par Story
- Chaque Sub-task doit être testable indépendamment si possible
- Les Sub-tasks doivent couvrir toutes les phases : développement, test, documentation

---

## 4. Description des champs par type d'élément

#### Champs des Stories

**Description** → Définit précisément le comportement utilisateur attendu avec le format "En tant que [rôle], je veux [action], afin de [bénéfice métier]"

**Acceptance Criteria** → Définit précisément quand la Story est considérée comme terminée avec une liste de conditions vérifiables sous forme "Given... When... Then..." par scénario

**Priority** → estimer l'ordre des tâches avec ces valeurs :

- **Must have** : Fonctionnalité critique, sans laquelle le projet ne peut pas être livré
- **Should have** : Fonctionnalité importante mais non bloquante
- **Could have** : Fonctionnalité souhaitable si le temps le permet
- **Won't have** : Fonctionnalité mineure exclue de cette itération

**Story Point Estimate** → Mesure la complexité relative de la Story grâce à ces critères :

- Complexité technique
- Incertitude / risque
- Effort requis
- Temps estimé

**Fix version** → Indique à quelle release/version la Story est ciblée (exemple: "v1.0.1", "v1.2.0")

**DoR validée** → Indique que la Story respecte tous les critères pour être prise en sprint

**DoD validée** → Confirme que tous les critères de "terminé" sont respectés

**Status** → Indique l'état d'avancement actuel de la Story

**Due Date** → Date limite pour terminer la Story

**Parent** → Rattache la Story à son Epic parent

**Sprint** → Indique dans quel sprint la Story est planifiée

**Labels** → Pour la catégorisation des éléments et le filtrage dans l'interface de Jira

**Assignee** → Identifie qui est responsable de la Story (dans ce contexte ça sera toujours moi)

#### Champs des Tasks

Les Tasks partagent la plupart des champs avec les Stories, avec quelques spécificités :

**Différences par rapport aux Stories**

**Priority →** Échelle différente :

- **Highest** : Bloquant, arrêt de production ou sécurité critique
- **High** : Impact significatif, à traiter rapidement
- **Medium** : Priorité normale
- **Low** : Peut attendre
- **Lowest** : Nice to have, très faible urgence

**Absence de Story Points** → Les Tasks ne sont pas estimées en story points car elles ne contribuent pas directement à la vélocité métier. Remplacé par une estimation de temps.

**Time Tracking** → Suivi précis du temps investi sur les tâches techniques

#### Champs des Sub-tasks

Les Sub-tasks ont une structure simplifiée car elles représentent un travail très granulaire. Elles reprennent la plupart des champs des stories ou tasks avec quelques ajouts / modifications :

**Original Estimate** → Estimation du temps nécessaire pour cette Sub-task entre 30m à 8h (si plus, découper en plusieurs Sub-tasks)

**Fix version** → Versionning plus précis qu'entre les stories (hotfix, minor release)

**Commit** → Référence au commit Git associé à cette Sub-task

---

### 5. Workflow de développement

Le workflow représente le cycle de vie des éléments depuis leur création jusqu'à leur livraison complète.

#### Workflow des Stories

**États du workflow**

1. **Backlog**

- **Description** : État initial, la Story est identifiée mais non planifiée
- **Critères d'entrée** : Story créée avec un Summary

2. **Ready**

- **Description** : La Story est sélectionnée pour le sprint en cours
- **Critères d'entrée** :
  - DoR validée (case cochée)
  - Story estimée
  - Assignée à un sprint

3. **In Progress**

- **Description** : Le développement actif a commencé
- **Critères d'entrée** : Au moins une Sub-task est en cours

4. **In Review**

- **Description** : Le code est développé et attend une revue
- **Critères d'entrée** :
  - Toutes les Sub-tasks de développement sont terminées
  - Code poussé sur la branche de feature Git
  - Tests unitaires passants

5. **Testing**

- **Description** : Tests fonctionnels et validation des critères d'acceptation
- **Critères d'entrée** :
  - Code review terminée et approuvée
  - Code mergé sur la branche de développement
  - Déployé en environnement de test
- **Résultats possibles** :
  - ✅ Tous les tests passent → Transition vers "Done"
  - ❌ Bugs détectés → Retour vers "In Progress"

6. **Done**

- **Description** : La Story est complète et livrable
- **Critères d'entrée** (Definition of Done) :
  - Tous les critères d'acceptation validés
  - DoD validée (case cochée)
  - Code développé, revu et testé
  - Tests unitaires et d'intégration passants
  - Documentation technique mise à jour
  - Déployé en environnement de test
  - Aucun bug bloquant ou critique ouvert

**Règles de transition strictes**

- Une Story ne peut pas passer de "Backlog" à "In Progress" directement (doit passer par "Ready")
- Une Story ne peut pas passer à "Done" sans que DoD soit validée
- Tout retour en arrière doit avoir un ticket de bug créé

#### Workflow des Tasks

Le workflow des Tasks est légèrement simplifié :

**États** : Backlog → To Do → In Progress → Done

**Particularités** :

- Les Tasks peuvent être autonomes (hors sprint) pour les travaux d'infrastructure
- Pas de state "Testing", “In Review” systématique (dépend du type de Task)
- La validation se fait plutôt sur des critères techniques (configuration fonctionnelle, documentation complète)

#### Workflow des Sub-tasks

**États simplifiés** : To Do → In Progress → Done

**Règles** :

- Une Sub-task est créée avant que sa Story parent passe à "Ready"
- Toutes les Sub-tasks doivent être "Done" avant que la Story parent puisse passer à "In Review"
- Les Sub-tasks sont généralement traitées dans l'ordre de priorité (Highest en premier)

---

### 6. Gestion de la qualité

**Definition of Ready (DoR) - Checklist** Avant de prendre une Story en sprint, vérifier :

- [ ] Description complète au bon format
- [ ] Critères d'acceptation définis et testables
- [ ] Story estimée en points
- [ ] Priorité MoSCoW assignée
- [ ] Epic parent assignée
- [ ] Dépendances identifiées et documentées
- [ ] Pas de bloqueurs techniques / stories connus
- [ ] Case "DoR validée" cochée dans Jira

**Definition of Done (DoD) - Checklist** Avant de marquer une Story comme "Done", vérifier :

- [ ] Tous les critères d'acceptation sont validés
- [ ] Toutes les Sub-tasks sont terminées
- [ ] Code développé
- [ ] Tests unitaires écrits et passants (couverture > 80%)

- [ ] Tests d'intégration passants
- [ ] Auto review effectuée
- [ ] Code mergé sur la branche develop
- [ ] Déployé en environnement de test
- [ ] Tests fonctionnels manuels effectués
- [ ] Documentation technique mise à jour
- [ ] Aucun bug bloquant ou critique ouvert
- [ ] Case "DoD validée" cochée dans Jira

**Métriques à suivre**

- Vélocité par sprint (évolution)
- Taux de complétion du sprint (% de Stories Done)
- Lead time moyen (temps Backlog → Done)
- Nombre de bugs post-release
- Respect de la DoD

---

**Conclusion**

Cet espace de travail Jira est conçu pour offrir un cadre structuré et professionnel de gestion de projet en suivant la méthodologie Scrum. Même dans un contexte de travail en solo, l'adoption de ces pratiques apporte une aide considérable lors du développement.

Les choix effectués (Scrum plutôt que Kanban, hiérarchie à trois niveaux, différenciation Stories/Tasks, champs spécifiques) répondent à des objectifs précis de traçabilité et d'amélioration continue. Cette documentation servira de référence tout au long du projet et démontre la capacité à structurer et documenter un processus de développement complet.

---

**Document créé le** : 7 novembre 2025
**Version** : 1.2
**Auteur** : Jocelyn SAINSON
**Projet** : Dev - JO2024