



# Rapport de Projet de Fin d'Études

Digitalisation des processus SCRUM et pré-intégration BPMN au sein d'IDVEY

Logo\_Entreprise.png

**Réalisé par :**  
Houssem Chedly

**Encadrant Entreprise :**  
Nom Prénom – IDVEY

**Encadrant Académique :**  
Nom Prénom – Polytech Lyon

## Résumé

La gestion manuelle des projets Agile SCRUM peut rapidement devenir complexe et source d'erreurs, notamment lorsque les équipes utilisent des outils non centralisés. Ce projet de fin d'études vise à digitaliser l'ensemble du processus SCRUM au sein d'IDVEY à travers le développement d'une plateforme web complète. Celle-ci permet la gestion des projets, du Product Backlog, des Sprints, des tâches techniques et des blocages, tout en préparant l'intégration du moteur BPMN Camunda pour l'automatisation future des workflows.

L'architecture adoptée repose sur Angular pour l'interface utilisateur, Spring Boot pour la logique métier et PostgreSQL pour la persistance des données, le tout déployé dans un écosystème Docker. Les résultats démontrent la faisabilité d'une solution moderne, modulaire, scalable et capable d'améliorer la visibilité, la collaboration et la traçabilité des projets SCRUM.

**Mots-clés :** SCRUM, BPMN 2.0, Camunda, Angular, Spring Boot, PostgreSQL, Docker.

## Abstract

Manual management of Agile SCRUM projects often leads to inefficiencies, inconsistencies and a lack of visibility. This final-year project aims to digitalize the entire SCRUM workflow within IDVEY through the development of a unified web platform. The system manages projects, product backlog items, sprints, technical tasks and impediments, while preparing future integration with the BPMN Camunda engine for workflow automation.

The platform is built using Angular for the frontend, Spring Boot for backend logic, and PostgreSQL for data persistence, all deployed in a Docker-based environment. The results confirm the feasibility of a modular, scalable and efficient solution that enhances collaboration, transparency and workflow automation within SCRUM teams.

**Keywords :** SCRUM, BPMN 2.0, Camunda, Angular, Spring Boot, PostgreSQL, Docker.



# Table des matières

<b>Remerciements</b>	<b>7</b>
<b>A Présentation de l’Entreprise IDVEY</b>	<b>8</b>
A.1 Introduction . . . . .	8
A.2 Historique et évolution de l’entreprise . . . . .	8
A.3 Fiche signalétique . . . . .	9
A.4 Domaine d’activité et expertise . . . . .	9
A.4.1 Expertise humaine locale . . . . .	9
A.4.2 Maîtrise technologique . . . . .	9
A.5 Structure organisationnelle . . . . .	9
A.6 Mission, vision et valeurs de l’entreprise . . . . .	10
A.6.1 Mission . . . . .	10
A.6.2 Vision . . . . .	10
A.6.3 Valeurs fondamentales . . . . .	10
A.7 Conclusion du chapitre . . . . .	11
<b>B Présentation du projet</b>	<b>12</b>
B.1 Contexte général . . . . .	12
B.2 Problématique . . . . .	12
B.3 Objectifs du projet . . . . .	12
B.4 Cahier des charges . . . . .	13
B.4.1 Exigences fonctionnelles principales . . . . .	13
B.4.2 Exigences non fonctionnelles . . . . .	13
B.5 Vue d’ensemble du système à développer . . . . .	13
B.5.1 Interface utilisateur (Angular) . . . . .	13
B.5.2 Backend Spring Boot . . . . .	14
B.5.3 Moteur BPMN Camunda . . . . .	14
B.6 Workflow SCRUM retenu dans le projet . . . . .	14
B.7 Rôles utilisateurs prévus dans l’application . . . . .	15
B.7.1 Administrateur . . . . .	15
B.7.2 Product Owner (PO) . . . . .	15
B.7.3 Scrum Master (SM) . . . . .	15
B.7.4 Développeur . . . . .	15
B.8 Conclusion du chapitre . . . . .	15
<b>C État de l’art</b>	<b>16</b>
C.1 Introduction . . . . .	16
C.2 La méthodologie Agile SCRUM . . . . .	16
C.2.1 Piliers SCRUM . . . . .	16
C.2.2 Valeurs SCRUM . . . . .	17



C.2.3	Rôles SCRUM . . . . .	17
C.2.4	Événements SCRUM . . . . .	18
C.2.5	Artefacts SCRUM . . . . .	19
C.3	Outils SCRUM existants . . . . .	20
C.3.1	Jira Software . . . . .	20
C.3.2	Trello, Asana et Monday.com . . . . .	21
C.3.3	Azure DevOps . . . . .	21
C.3.4	Synthèse comparative des outils . . . . .	22
C.3.5	Limites de la méthodologie SCRUM . . . . .	22
C.4	BPMN 2.0 : un standard international . . . . .	23
C.4.1	Définition . . . . .	23
C.4.2	Objectifs et avantages . . . . .	24
C.4.3	Éléments graphiques de base . . . . .	24
C.4.4	Exemples de processus BPMN . . . . .	26
C.5	Camunda BPM . . . . .	26
C.5.1	Présentation générale . . . . .	26
C.5.2	Rôle de Camunda dans l'automatisation BPMN . . . . .	27
C.5.3	Architecture générale de Camunda 7 . . . . .	27
C.5.4	Camunda 7 vs Camunda 8 . . . . .	28
C.5.5	Pourquoi Camunda 7 pour ce projet ? . . . . .	28
C.6	Camunda Modeler . . . . .	29
C.6.1	Son rôle . . . . .	29
C.6.2	Avantages pour les développeurs et analystes métier . . . . .	29
C.6.3	Intégration avec Camunda 7 . . . . .	29
C.7	Conclusion du chapitre . . . . .	29
<b>D</b>	<b>Analyse et Conception du Système</b>	<b>30</b>
D.1	Introduction . . . . .	30
D.2	Analyse fonctionnelle . . . . .	30
D.2.1	Description générale du système . . . . .	30
D.2.2	Acteurs du système . . . . .	30
D.2.3	Besoins fonctionnels . . . . .	31
D.2.4	Besoins non fonctionnels . . . . .	31
D.3	Modélisation des exigences . . . . .	31
D.3.1	Diagramme des cas d'utilisation . . . . .	31
D.3.2	Description détaillée de quelques cas d'utilisation . . . . .	33
D.4	Modélisation UML . . . . .	33
D.4.1	Diagramme de classes . . . . .	33
D.4.2	Diagrammes de séquence . . . . .	35
D.4.3	Diagramme d'activité global . . . . .	38
D.5	Conception de l'architecture logicielle . . . . .	39
D.5.1	Vue d'ensemble . . . . .	39
D.5.2	Structure du frontend (Angular) . . . . .	42
D.5.3	Structure du backend (Spring Boot) . . . . .	42
D.5.4	Persistance des données (PostgreSQL) . . . . .	42
D.6	Intégration BPMN prévue avec Camunda . . . . .	43
D.6.1	Processus de création d'un projet . . . . .	44
D.6.2	Processus SCRUM d'un Sprint . . . . .	44
D.7	Architecture technique et déploiement . . . . .	45
D.7.1	Vue d'ensemble . . . . .	45
D.7.2	Schéma technique de l'infrastructure . . . . .	45
D.7.3	Rôle des composants . . . . .	47
D.7.4	Réseau et ports utilisés . . . . .	47
D.7.5	Déploiement via Docker Compose . . . . .	48



D.8 Conclusion . . . . .	48
<b>E Implémentation du Backend Spring Boot</b>	<b>49</b>
E.1 Introduction . . . . .	49
E.2 Architecture du backend . . . . .	49
E.2.1 Organisation en couches . . . . .	49
E.2.2 Organisation des packages . . . . .	50
E.2.3 Cycle de traitement d'une requête . . . . .	50
E.3 Modélisation des entités (JPA) . . . . .	51
E.3.1 Utilisateur et rôles . . . . .	51
E.3.2 Projet . . . . .	51
E.3.3 Stories (Product Backlog) . . . . .	52
E.3.4 Sprint . . . . .	52
E.3.5 Tâches techniques (Sprint Backlog) . . . . .	52
E.3.6 Blocages (Impediments) . . . . .	53
E.4 Repositories et accès aux données . . . . .	53
E.4.1 UserRepository . . . . .	54
E.4.2 ProjectRepository . . . . .	54
E.4.3 StoryRepository . . . . .	54
E.4.4 SprintRepository . . . . .	54
E.4.5 TaskRepository . . . . .	55
E.4.6 BlockerRepository . . . . .	55
E.5 Services métier . . . . .	55
E.5.1 UserService . . . . .	56
E.5.2 ProjectService . . . . .	56
E.5.3 StoryService (Product Backlog) . . . . .	56
E.5.4 SprintService . . . . .	57
E.5.5 TaskService . . . . .	57
E.5.6 BlockerService . . . . .	57
E.5.7 MeetingService . . . . .	57
E.5.8 NotificationService . . . . .	58
E.6 API REST . . . . .	58
E.6.1 Authentification (JWT) . . . . .	58
E.6.2 Gestion des utilisateurs (Admin) . . . . .	59
E.6.3 Gestion des projets . . . . .	59
E.6.4 Product Backlog (Stories) . . . . .	59
E.6.5 Sprint Backlog et gestion des sprints . . . . .	59
E.6.6 Tâches techniques (Task Board) . . . . .	60
E.6.7 Blocages (Impediments) . . . . .	60
E.6.8 Métriques et tableaux de bord . . . . .	60
E.6.9 Authentification (JWT) . . . . .	60
E.6.10 Gestion des utilisateurs (Admin) . . . . .	61
E.6.11 Gestion des projets . . . . .	61
E.6.12 Product Backlog (Stories) . . . . .	61
E.6.13 Sprint Backlog et gestion des sprints . . . . .	61
E.6.14 Tâches techniques (Task Board) . . . . .	62
E.6.15 Blocages (Impediments) . . . . .	62
E.6.16 Métriques et tableaux de bord . . . . .	62
E.7 Sécurité et authentification JWT . . . . .	62
E.7.1 Principe général . . . . .	63
E.7.2 Structure du token JWT . . . . .	63
E.7.3 Filtres de sécurité . . . . .	63
E.7.4 Gestion des rôles et permissions (RBAC) . . . . .	64
E.7.5 Gestion des refresh tokens . . . . .	64



E.7.6 Sécurité des mots de passe . . . . .	64
E.7.7 Avantages de JWT dans une architecture Angular / Spring Boot . . . . .	65
<b>E.8 Pré-intégration du moteur BPMN Camunda . . . . .</b>	<b>65</b>
E.8.1 Motivation : pourquoi intégrer Camunda ? . . . . .	65
E.8.2 Structure prévue pour l'intégration . . . . .	65
E.8.3 Processus BPMN prévus . . . . .	66
E.8.4 Déploiement automatique des fichiers BPMN . . . . .	66
E.8.5 Intégration future dans Angular . . . . .	66
E.8.6 Points techniques déjà implémentés . . . . .	67
<b>E.9 Conclusion . . . . .</b>	<b>67</b>
<b>F Implémentation du Frontend Angular . . . . .</b>	<b>68</b>
F.1 Introduction . . . . .	68
F.2 Architecture générale du frontend . . . . .	68
F.3 Gestion de l'authentification JWT . . . . .	69
F.4 Modules fonctionnels du frontend . . . . .	70
F.4.1 Module Projets (Product Owner) . . . . .	70
F.4.2 Module Backlog (PO) . . . . .	70
F.4.3 Module Sprints (Scrum Master) . . . . .	70
F.4.4 Module Tasks (Développeurs) . . . . .	71
F.4.5 Module Meetings (SCRUM) . . . . .	71
F.4.6 Module Dashboard (PO & SM) . . . . .	71
F.4.7 Module Admin . . . . .	71
F.5 Services Angular . . . . .	72
F.6 Interfaces graphiques (UI/UX) . . . . .	72
F.6.1 Page d'authentification . . . . .	72
F.6.2 Gestion des projets (PO) . . . . .	73
F.6.3 Gestion du Product Backlog . . . . .	76
F.6.4 Module Sprints (Scrum Master) . . . . .	78
F.6.5 Tableau Kanban (Développeurs) . . . . .	80
F.6.6 Blocages (Impediments) . . . . .	81
F.6.7 Dashboard (PO & SM) . . . . .	83
F.6.8 Module Administration . . . . .	84
F.7 Conclusion . . . . .	84
<b>G Déploiement et Intégration . . . . .</b>	<b>85</b>
G.1 Introduction . . . . .	85
G.2 Architecture technique déployée . . . . .	85
G.3 Configuration Docker . . . . .	87
G.4 Intégration du backend Spring Boot . . . . .	87
G.5 Déploiement du frontend Angular . . . . .	88
G.6 Intégration de Camunda 7 (pré-intégration) . . . . .	88
G.7 Tests de déploiement . . . . .	89
G.8 Conclusion . . . . .	90
<b>H Résultats et Tests . . . . .</b>	<b>91</b>
H.1 Introduction . . . . .	91
H.2 Tests fonctionnels . . . . .	91
H.2.1 Test 1 : Création d'un projet . . . . .	91
H.2.2 Test 2 : Gestion du Product Backlog . . . . .	92
H.2.3 Test 3 : Création et gestion d'un Sprint . . . . .	93
H.2.4 Test 4 : Gestion des blocages . . . . .	94
H.3 Tests API REST (Postman) . . . . .	95
H.3.1 Exemple : Test de l'authentification . . . . .	95

H.3.2 Exemple : Test de création d'un projet . . . . .	95
H.4 Tests de persistance PostgreSQL . . . . .	95
H.5 Tests de l'interface Angular . . . . .	96
H.6 Tests de sécurité . . . . .	97
H.7 Synthèse des résultats . . . . .	97
H.8 Conclusion . . . . .	98
<b>I Perspectives d'amélioration et Conclusion générale</b>	<b>99</b>
I.1 Introduction . . . . .	99
I.2 Perspectives d'amélioration . . . . .	99
I.2.1 Automatisation avancée via Camunda BPM . . . . .	99
I.2.2 Améliorations backend . . . . .	99
I.2.3 Améliorations frontend (Angular) . . . . .	100
I.2.4 Évolutions de l'architecture technique . . . . .	100
I.2.5 Améliorations fonctionnelles . . . . .	100
I.3 Conclusion générale . . . . .	100
<b>Liste des figures</b>	<b>102</b>
<b>Liste des tableaux</b>	<b>105</b>
<b>Liste des équations</b>	<b>106</b>
<b>Liste des codes</b>	<b>107</b>
<b>Index</b>	<b>108</b>
<b>Bibliographie</b>	<b>108</b>
<b>Conclusion</b>	<b>109</b>
<b>Table des annexes</b>	<b>110</b>
<b>Annexe A Annexe A — Interfaces Frontend Angular</b>	<b>114</b>
A.1 Authentification . . . . .	114
A.2 Gestion des projets . . . . .	115
A.3 Product Backlog . . . . .	118
A.4 Gestion des sprints . . . . .	120
A.5 Tableau Kanban . . . . .	122
A.6 Gestion des blocages (Impediments) . . . . .	123
A.7 Dashboard SCRUM . . . . .	125
A.8 Module Administration . . . . .	126
<b>Annexe B Annexe B — Backend (API REST, Logs, JSON)</b>	<b>127</b>
B.1 Documentation API REST (Swagger) . . . . .	128
B.2 Exemple de réponse JSON — Authentification JWT . . . . .	129
B.3 Exemple d'appel API — Création d'un projet . . . . .	129
B.4 Extrait de logs Spring Boot . . . . .	129
<b>Annexe C Annexe C — BPMN (Camunda)</b>	<b>130</b>
C.1 Processus BPMN — Création de projet . . . . .	131
C.2 Processus BPMN — Sprint Workflow . . . . .	132
<b>Annexe D Annexe D — UML (Modélisation du Système)</b>	<b>133</b>
D.1 Diagramme de classes . . . . .	133
D.2 Diagramme de cas d'utilisation . . . . .	135

---

D.3 Diagrammes de séquence . . . . .	136
<b>Annexe E Annexe E — Déploiement Docker et Infrastructure</b>	<b>137</b>
E.1 Fichier docker-compose.yml . . . . .	137
E.2 Architecture technique déployée . . . . .	138



# Remerciements

Au terme de ce projet de fin d'études, je souhaite exprimer ma profonde gratitude à toutes les personnes qui ont contribué, de près ou de loin, à la réussite de ce travail.

Je tiens à remercier du fond du cœur ma mère, pour son soutien inconditionnel, son courage et l'amour qu'elle m'a toujours donné. Mes pensées vont également à mon père et à mon grand frère, disparus trop tôt, mais dont la mémoire, les valeurs et les encouragements silencieux continuent de guider mes choix et mon parcours.

J'exprime ma profonde reconnaissance à ma grande sœur pour ses conseils avisés et son accompagnement constant, ainsi qu'à son époux, dont l'aide m'a été d'une grande valeur tout au long de ce projet. Je souhaite également adresser une pensée affectueuse à leurs enfants, Yassine et Sarah, dont la joie et l'innocence ont souvent été une source de motivation et de lumière.

Je remercie chaleureusement mes encadrants professionnels, **Mahmoud Somai** et **Mehdi Hemriti**, pour leur confiance, leur disponibilité et la qualité de leur accompagnement tout au long de ce stage. Leur expertise et leurs conseils ont grandement contribué à l'aboutissement de ce travail.

Mes remerciements s'adressent également à mon encadrante académique, **Madame Sonia Batis**, pour son suivi, ses orientations et son soutien durant toute la période du projet.

J'adresse aussi un remerciement tout particulier à mes amis, qui sont bien plus qu'un simple entourage : ils sont une véritable famille. Leur présence, leur soutien, leurs encouragements et leur énergie ont été essentiels pour garder le rythme et la motivation dans les moments difficiles.

Enfin, je remercie toutes les personnes au sein d'IDVEY et de Polytech Lyon qui ont, de près ou de loin, contribué à rendre cette expérience riche, formatrice et mémorable.

À toutes et à tous, merci.

## CHAPITRE A

# Présentation de l'Entreprise IDVEY

### A.1 Introduction

Ce premier chapitre présente l'entreprise IDVEY, structure d'accueil de mon stage de fin d'études. L'objectif est de fournir une vision claire de son positionnement, de son activité, de son organisation interne ainsi que des ressources mises à disposition des projets informatiques, dont celui présenté dans ce rapport.



FIGURE A.1.1 – Logo de l'entreprise IDVEY

### A.2 Historique et évolution de l'entreprise

IDVEY est une entreprise tunisienne fondée en 2018, initialement spécialisée dans la transformation digitale multisectorielle. Durant ses premières années, l'entreprise a proposé des services variés tels que le développement web et mobile, l'intégration de solutions collaboratives et la digitalisation de processus métier.

À partir de 2022, IDVEY opère un virage stratégique majeur. Face à la saturation du marché du développement généraliste et à l'essor de la digitalisation réglementaire en Afrique francophone, l'entreprise décide de se spécialiser dans un domaine à forte valeur ajoutée : la conformité réglementaire, la sérialisation pharmaceutique et les solutions numériques associées.

Cette spécialisation a permis à IDVEY de se positionner comme un acteur expert, capable de répondre à des besoins précis, complexes et critiques dans l'industrie pharmaceutique.

## A.3 Fiche signalétique

Pour synthétiser son identité, la fiche signalétique d'IDVEY se présente comme suit au niveau du tableau A.3.1.

Élément	Description
Nom	IDVEY
Statut juridique	Société privée (SARL)
Année de création	2018
Secteur d'activité	Transformation digitale réglementaire
Zone d'intervention	Afrique du Nord et Afrique francophone
Clients principaux	Laboratoires pharmaceutiques et acteurs réglementés
Site web	<a href="http://www.idvey.com">www.idvey.com</a>

TABLE A.3.1 – Fiche signalétique de l'entreprise IDVEY

## A.4 Domaine d'activité et expertise

IDVEY intervient principalement dans la digitalisation des processus réglementaires au sein du secteur pharmaceutique. Son expertise repose sur deux piliers essentiels.

### A.4.1 Expertise humaine locale

L'entreprise s'appuie sur une équipe maîtrisant en profondeur les normes pharmaceutiques en Afrique francophone, notamment en Tunisie et en Algérie. Cette connaissance locale est stratégique dans un domaine fortement réglementé.

### A.4.2 Maîtrise technologique

IDVEY propose et intègre plusieurs solutions hautement spécialisées, notamment :

- des solutions eCTD (Electronic Common Technical Document) pour la gestion des dossiers réglementaires ;
- des systèmes de sérialisation et de traçabilité des médicaments ;
- des plateformes SaaS adaptées au secteur réglementé ;
- des projets de transformation digitale sur mesure.

Ces solutions s'appuient sur des technologies modernes, fiables et conformes aux standards internationaux.

## A.5 Structure organisationnelle

L'organisation interne d'IDVEY repose sur une structure flexible, orientée vers l'expertise et l'agilité. Elle se compose de trois pôles principaux :

- **Pôle opérationnel** : composé d'ingénieurs informatiques, d'une coordinatrice administrative et de stagiaires. Il est responsable de la mise en œuvre technique, du développement logiciel et du suivi des projets.

- **Réseau de consultants et experts métiers** : spécialistes en réglementation pharmaceutique, conformité et qualité, intervenant selon les besoins des projets.
- **Pôle créatif** : rassemblant les profils UI/UX, chargé de l'expérience utilisateur et de l'identité visuelle des solutions digitales proposées aux clients.

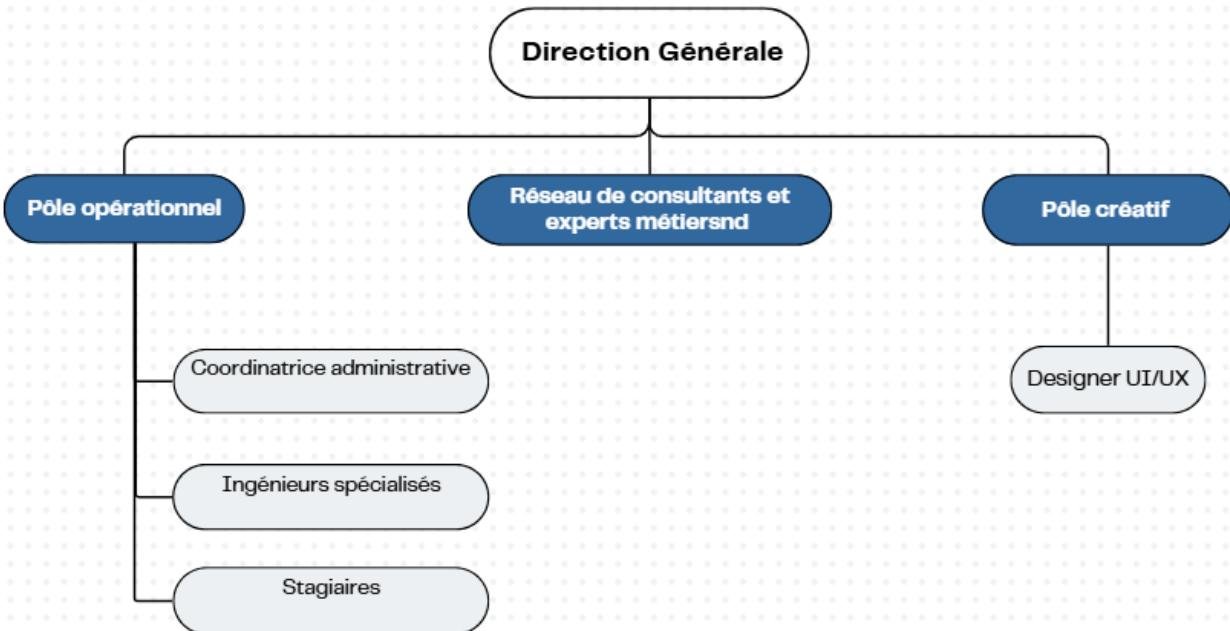


FIGURE A.5.1 – Organigramme de l'entreprise IDVEY

## A.6 Mission, vision et valeurs de l'entreprise

### A.6.1 Mission

La mission d'IDVEY est d'accompagner les acteurs pharmaceutiques dans leur transformation digitale en leur offrant des solutions fiables, conformes aux réglementations internationales et adaptées aux réalités locales des marchés africains.

### A.6.2 Vision

L'entreprise ambitionne de devenir le partenaire de référence en digitalisation réglementaire en Afrique francophone grâce à :

- une forte expertise technologique ;
- un accompagnement personnalisé des clients ;
- une proximité géographique et culturelle avec les laboratoires et institutions.

### A.6.3 Valeurs fondamentales

Les valeurs clés qui guident IDVEY sont :

- **Engagement** : la satisfaction des clients est placée au cœur des priorités ;

- **Efficacité** : les solutions proposées visent la performance et l'amélioration continue ;
- **Confiance** : les relations sont basées sur la transparence et la fiabilité ;
- **Innovation** : la recherche de solutions sur mesure et adaptées à chaque contexte.

## A.7 Conclusion du chapitre

IDVEY est une entreprise dynamique, spécialisée et structurée pour répondre aux exigences de la digitalisation réglementaire. Son expertise, son organisation et sa vision stratégique créent un environnement favorable à l'accueil de projets innovants tels que celui développé dans le cadre de ce projet de fin d'études.

## CHAPITRE B

# Présentation du projet

## B.1 Contexte général

La gestion des projets en entreprise repose souvent sur des outils manuels tels que les fichiers Excel, les échanges par e-mail ou des réunions répétitives. Ces méthodes entraînent fréquemment :

- un manque de visibilité sur l'avancement réel des tâches,
- des retards non anticipés,
- une détection tardive des blocages,
- une difficulté à coordonner les équipes.

Afin d'améliorer la fiabilité du suivi et de réduire la charge administrative, l'entreprise IDVEY a souhaité automatiser plusieurs processus liés à la gestion de projets, et plus particulièrement aux méthodes Agiles de type **SCRUM**.

## B.2 Problématique

Les outils existants sont souvent trop complexes, non adaptés ou insuffisamment automatisés. Les besoins identifiés sont les suivants :

- centraliser les projets, user stories, tâches et sprints ;
- faciliter les réunions SCRUM (Daily, Sprint Planning, Review, Rétrospective) ;
- détecter rapidement les blocages et notifier automatiquement les acteurs concernés ;
- offrir une interface simple pour les rôles SCRUM : Product Owner, Scrum Master, Équipe de développement ;
- fournir un moteur BPMN (Camunda) pour orchestrer et automatiser les workflows.

Ces besoins ont conduit à la mise en place d'une plateforme interne de gestion de projets SCRUM entièrement digitalisée.

## B.3 Objectifs du projet

Les objectifs principaux du stage sont les suivants :

- **Modéliser** les processus métier SCRUM en BPMN 2.0 (Camunda) ;
- **Développer un backend** en Spring Boot pour gérer projets, tâches, sprints et utilisateurs ;
- **Développer un frontend** Angular pour permettre aux utilisateurs d'interagir avec le système ;

- Automatiser les notifications, validations et workflows via le moteur Camunda ;
- Construire un tableau de bord interactif affichant :
  - vitesse,
  - burn-down chart,
  - suivi des stories,
  - taux de blocage.

Le projet s'inscrit dans une démarche d'amélioration continue de l'organisation interne d'ID-VEY, et vise à doter l'entreprise d'un outil similaire à Jira Software, mais adapté aux contraintes locales.

## B.4 Cahier des charges

Le cahier des charges fonctionnel impose les exigences suivantes :

### B.4.1 Exigences fonctionnelles principales

- Gestion des utilisateurs (Admin, Product Owner, Scrum Master, Développeur) ;
- Création, modification, suppression de projets ;
- Gestion du **Product Backlog** ;
- Gestion des **sprints** et du Sprint Backlog ;
- Gestion des **user stories** et des tâches ;
- Gestion des **blocages** et système d'escalade ;
- Automation via BPMN (notifications, escalade, validation QA) ;
- Tableaux de bord départementaux (PO, SM, Admin).

### B.4.2 Exigences non fonctionnelles

- Authentification sécurisée via **JWT** ;
- Architecture REST ;
- Base de données PostgreSQL ;
- Réactivité de l'interface (Angular 19) ;
- Conteneurisation **Docker** ;
- Compatibilité Camunda Modeler pour importer/exporter les processus BPMN.

## B.5 Vue d'ensemble du système à développer

Le système développé se compose de trois grandes couches :

### B.5.1 Interface utilisateur (Angular)

Permettre aux utilisateurs SCRUM de :

- consulter leurs projets,
- gérer le backlog et les sprints,

- participer aux réunions,
- déclarer ou résoudre les blocages,
- consulter les tableaux de bord.

### B.5.2 Backend Spring Boot

Fourni sous forme d'API REST :

- authentification et autorisation JWT,
- gestion des projets, stories, tâches, sprints,
- détecteur automatique de blocages,
- communication avec le moteur BPMN Camunda.

### B.5.3 Moteur BPMN Camunda

Responsable de :

- automatiser les workflows SCRUM,
- envoyer des notifications,
- escalader les cas prioritaires,
- synchroniser les états (To Do → In Progress → Review → Done).

## B.6 Workflow SCRUM retenu dans le projet

Le workflow général adopté est représenté sur la Figure B.6.1. (Le diagramme BPMN sera détaillé au Chapitre 4.)

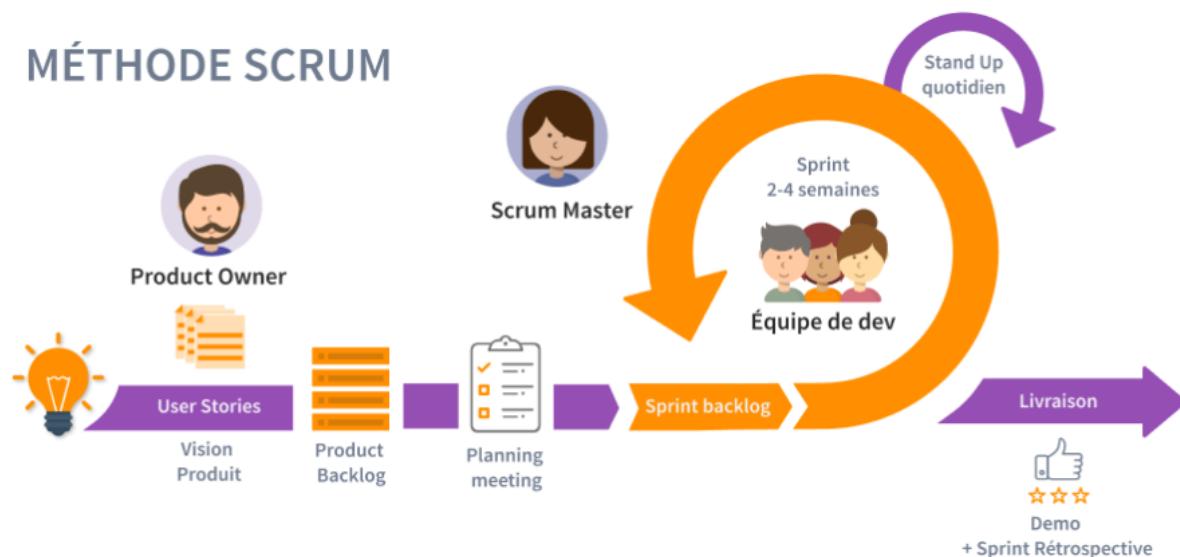


FIGURE B.6.1 – Workflow SCRUM général du projet (schéma conceptuel)

## B.7 Rôles utilisateurs prévus dans l'application

Les rôles SCRUM implémentés dans l'application sont :

### B.7.1 Administrateur

- gère les comptes utilisateurs,
- assigne les rôles,
- supervise l'ensemble des projets.

### B.7.2 Product Owner (PO)

- gère le backlog produit,
- définit les priorités,
- valide les stories.

### B.7.3 Scrum Master (SM)

- gère les sprints,
- anime les réunions,
- détecte et traite les blocages.

### B.7.4 Développeur

- exécute les tâches du sprint,
- signale les obstacles,
- met à jour l'état des stories.

## B.8 Conclusion du chapitre

Ce chapitre a présenté le contexte, les objectifs et les exigences du projet. Le système à développer vise à moderniser le suivi SCRUM en intégrant un moteur BPMN et une interface ergonomique. Le chapitre suivant étudiera l'état de l'art : SCRUM, BPMN, Jira et les outils existants.



## CHAPITRE C

# État de l'art

## C.1 Introduction

Ce chapitre présente un état de l'art complet autour des technologies et méthodes utilisées dans ce projet : la méthodologie Agile SCRUM, les outils existants du marché, le standard BPMN 2.0, ainsi que la plateforme Camunda BPM adoptée pour l'automatisation des workflows SCRUM. L'objectif est d'établir un socle théorique solide avant d'aborder l'analyse et la conception du système.

## C.2 La méthodologie Agile SCRUM

Scrum est une structure Agile qui facilite la collaboration au sein des équipes et les aide à réaliser des tâches à haute valeur ajoutée. Il repose sur une approche itérative et incrémentale, favorisant la livraison rapide et continue de valeur au client. La méthode Scrum fonctionne sur le principe des sprints : des cycles de travail de deux à quatres semaines à l'issue desquels un livrable est attendu. Outre le sprint, il existe deux autres événements Scrum : les réunions debout quotidiennes et les rétrospectives de sprint. Les réunions debout quotidiennes ont lieu tous les jours, comme leur nom le suggère. En 15 minutes, elles permettent à l'équipe Scrum d'interagir et de coordonner ses tâches pour la journée. Quant aux rétrospectives de sprint, elles sont organisées par le Scrum master à chaque fin de sprint. C'est l'occasion pour l'équipe de faire le point sur son travail et de mettre en place des changements pour les prochains sprints.

### C.2.1 Piliers SCRUM

SCRUM s'appuie sur trois piliers fondamentaux :

- **Transparence** : Tous les aspects importants du processus doivent être visibles et compréhensibles par tous. Cela inclut les objectifs, les priorités, les obstacles, et les progrès réalisés
- **Inspection** : À intervalles réguliers (Daily Scrum, Review, Rérospective), les équipes inspectent leur travail, les processus, et les résultats pour identifier les écarts ou les problèmes potentiels.
- **Adaptation** : Si, à la suite d'une inspection, des écarts sont identifiés, l'équipe ajuste son plan, ses processus ou ses comportements pour rester sur la bonne voie.

Ces piliers assurent que l'équipe SCRUM puisse continuellement apprendre et s'adapter pour livrer un produit de meilleure qualité.

## C.2.2 Valeurs SCRUM

SCRUM repose également sur cinq valeurs essentielles :

- l'engagement : l'équipe Scrum doit être unie, et ses membres doivent faire preuve de confiance mutuelle. Ils s'engagent pour la durée du sprint et consacrent leurs efforts à l'amélioration constante pour obtenir la meilleure solution possible.
- le courage : lors d'un sprint Scrum, votre équipe rencontrera sans doute des problèmes qui ne pourront pas être complètement résolus. Les équipes Scrum ont le courage de poser ouvertement des questions difficiles et d'y répondre honnêtement afin de parvenir à la solution la plus adaptée.
- la concentration : l'équipe Scrum travaille sur des tâches extraites du backlog produit à chaque sprint. Elle se concentre sur le travail qu'elle a sélectionné dans le backlog pour réaliser ses livrables d'ici la fin du sprint.
- le respect : la collaboration est cruciale en Scrum. Pour l'encourager, les membres de l'équipe doivent faire preuve de respect, aussi bien entre eux qu'auprès du Scrum Master et du processus Scrum.
- l'ouverture : le déroulement du processus Scrum est semé d'embûches. Les membres de l'équipe Scrum doivent être ouverts aux nouvelles idées et occasions qui leur permettront de tirer de nouveaux enseignements et les aideront à améliorer leur produit ou processus.

## C.2.3 Rôles SCRUM

SCRUM définit trois rôles essentiels, chacun ayant des responsabilités précises et complémentaires : le **Product Owner**, le **Scrum Master** et l'**équipe de développement**. Ces rôles assurent le bon déroulement du cadre Agile et garantissent la livraison d'un produit à forte valeur ajoutée.

**a) Le Product Owner (PO)** Le Product Owner est responsable de maximiser la valeur du produit issue du travail de l'équipe de développement. Il représente la voix du client et porte la vision du produit.

**Principales responsabilités :**

- définir et prioriser le *Product Backlog* selon la valeur métier ;
- garantir que le backlog est clair, visible et compris par tous ;
- collaborer avec les parties prenantes (utilisateurs, clients, responsables métiers) ;
- décider des fonctionnalités à développer, ajuster ou supprimer.

Le PO doit être disponible, réactif, et disposer d'une bonne connaissance métier pour effectuer les bons choix stratégiques.

**b) Le Scrum Master** Le Scrum Master est le gardien du cadre SCRUM. Il agit comme facilitateur, coach et parfois médiateur au sein de l'équipe.

**Principales responsabilités :**

- s'assurer que SCRUM est compris et correctement appliqué ;
- faciliter les événements SCRUM (Daily, Sprint Planning, Review, Rétrospective) ;
- aider l'équipe à identifier et lever les obstacles (impediments) ;
- accompagner le Product Owner et l'équipe dans l'adoption des pratiques Agiles.

Le Scrum Master n'est pas un chef de projet : il ne donne pas d'ordres. Il aide l'équipe à s'auto-organiser et à améliorer continuellement ses méthodes de travail.

**c) L'équipe de développement** L'équipe de développement est un groupe auto-organisé et pluridisciplinaire dont la mission est de construire un incrément de produit potentiellement livrable à chaque Sprint.

#### Caractéristiques principales :

- composée généralement de 3 à 9 membres ;
- responsabilité collective de livrer un incrément conforme à la définition de « Done » ;
- les membres décident eux-mêmes de l'organisation du travail nécessaire pour atteindre les objectifs ;
- l'équipe est multidisciplinaire : développeurs, testeurs, designers, intégrateurs, etc.

L'équipe est autonome et n'a pas de hiérarchie interne : elle s'organise librement pour atteindre les objectifs fixés.

### C.2.4 Événements SCRUM

SCRUM est structuré autour de cinq événements clés qui rythment le déroulement du Sprint. Ils garantissent la transparence, l'inspection régulière et l'adaptation continue du travail réalisé par l'équipe.

**a) Le Sprint** Le Sprint constitue le cœur du cadre SCRUM. Il s'agit d'un cycle de développement court, d'une durée fixe (généralement entre une et quatre semaines), durant lequel une version potentiellement livrable du produit est développée.

#### Chaque Sprint :

- commence immédiatement à la fin du Sprint précédent ;
- doit avoir un objectif clair appelé *Sprint Goal* ;
- produit un incrément conforme à la définition de « Done » ;
- ne doit pas être perturbé par des changements majeurs.

Cette stabilité permet à l'équipe de travailler de manière concentrée et prévisible.

**b) Le Sprint Planning** Le Sprint Planning ouvre chaque Sprint. Il permet de définir le travail à accomplir durant l'itération à venir.

#### Durant cet événement :

- le Product Owner présente les éléments prioritaires du Product Backlog ;
- l'équipe de développement évalue la faisabilité ;
- l'équipe sélectionne les éléments à intégrer dans le Sprint (Sprint Backlog) ;
- un objectif de Sprint clair est défini collectivement.

Le Scrum Master veille au bon déroulement de cette réunion et au respect du cadre SCRUM.

**c) Le Daily Scrum** Le Daily Scrum est une réunion quotidienne, courte (15 minutes maximum), qui permet à l'équipe de synchroniser ses travaux.

#### Chaque membre partage succinctement :

- ce qu'il a fait la veille ;
- ce qu'il compte faire aujourd'hui ;
- les éventuels obstacles rencontrés.

Cette réunion favorise une communication fluide, une détection rapide des impediments, et un ajustement quotidien du plan de travail.

**d) La Sprint Review** La Sprint Review se tient à la fin du Sprint, en présence des parties prenantes. Elle a pour objectif :

- d'inspecter l'incrément produit ;
- de démontrer ce qui a été développé ;
- de recueillir les retours des utilisateurs, clients ou métiers ;
- de mettre à jour le Product Backlog en conséquence.

Il s'agit d'un événement collaboratif, orienté vers l'amélioration du produit et l'adaptation rapide aux besoins du marché.

**e) La Sprint Retrospective** La Sprint Retrospective clôture chaque Sprint. Elle est exclusivement réservée à l'équipe SCRUM.

Objectifs :

- analyser le fonctionnement de l'équipe ;
- identifier ce qui a bien fonctionné ;
- repérer les points d'amélioration ;
- définir des actions concrètes à mettre en œuvre dès le prochain Sprint.

Animée par le Scrum Master, cette réunion renforce l'amélioration continue et l'esprit d'équipe.

### C.2.5 Artefacts SCRUM

Les artefacts SCRUM assurent une transparence totale du travail réalisé. Ils fournissent une vision claire de ce qui est en cours, de ce qui a été accompli et de ce qui reste à faire. SCRUM en définit trois.

**a) Le Product Backlog** Le Product Backlog est une liste priorisée de tout le travail à réaliser pour améliorer le produit.

Caractéristiques :

- il est créé et maintenu par le Product Owner ;
- les éléments ne sont pas tous destinés à être développés : ce sont des opportunités ;
- il évolue continuellement selon les retours clients, les besoins métiers et l'apprentissage de l'équipe ;
- les éléments sont estimés, décrits et priorisés.

Le Product Owner doit régulièrement réorganiser et affiner ce backlog afin de maximiser la valeur produite.

**b) Le Sprint Backlog** Le Sprint Backlog est un sous-ensemble du Product Backlog.

Il contient :

- les éléments sélectionnés pour le Sprint ;
- un plan détaillé pour les réaliser ;
- une vision claire de l'avancement jour après jour.

Le Sprint Backlog est dynamique : l'équipe peut le réajuster durant le Sprint tant que cela ne remet pas en cause l'objectif fixé.

c) **L'Incrément** L'incrément est le résultat concret produit à la fin d'un Sprint.

Il peut être :

- une fonctionnalité développée,
- une amélioration,
- une correction,
- ou toute autre partie du produit.

Un incrément doit être potentiellement livrable et respecter la définition de « Done », partagée par toute l'équipe SCRUM.

Il représente la somme de tous les incréments produits jusqu'ici et doit être intéressant, cohérent et utilisable.

### C.3 Outils SCRUM existants

La gestion Agile et SCRUM est aujourd'hui largement supportée par plusieurs outils numériques. Ces plateformes offrent différents niveaux de fonctionnalités, allant de la simple gestion de tâches à des systèmes complets de pilotage Agile avec reporting avancé.

Les outils analysés dans cette section sont :

- Jira Software,
- Trello, Asana, Monday,
- Azure DevOps.

#### C.3.1 Jira Software

Jira Software est l'outil SCRUM le plus utilisé dans l'industrie logicielle. Développé par Atlassian, il est réputé pour sa richesse fonctionnelle et ses possibilités de personnalisation.

##### Avantages :

- gestion complète du backlog (création, estimation, priorisation) ;
- tableaux Scrum et Kanban configurables ;
- métriques avancées : burn-down chart, vitesse, cycle time, lead time ;
- automatisations simples via rules (notifications, transitions automatiques, conditions) ;
- intégration native avec Bitbucket, GitHub, GitLab et pipelines CI/CD ;
- marketplace riche en plugins (Structure, Automation, BigPicture...).

##### Limites :

- interface complexe pour les nouveaux utilisateurs ;
- configuration avancée souvent difficile ;
- personnalisation limitée des workflows sans plugins tiers ;
- absence totale de support BPMN 2.0 natif ;
- coût élevé pour les entreprises à grande échelle.

### C.3.2 Trello, Asana et Monday.com

Ces outils sont orientés vers la gestion visuelle des tâches grâce à des tableaux Kanban simples.

#### Avantages :

- interface intuitive et facile à prendre en main ;
- gestion simple des cartes, listes et deadlines ;
- collaboration rapide (commentaires, pièces jointes, assignations) ;
- automatisations légères via Power-Ups (Trello) ou règles intégrées ;
- outils adaptés aux petites équipes ou projets non techniques.

#### Limites :

- absence de gestion structurée du backlog ;
- pas de fonctionnalités SCRUM avancées (Sprint, vitesse, burn-down) ;
- reporting très limité ;
- pas d'intégration BPMN ou de workflows complexes ;
- peu adaptés aux projets techniques nécessitant coordination et scalabilité.

### C.3.3 Azure DevOps

Azure DevOps est une solution DevOps complète proposée par Microsoft. Elle combine la gestion SCRUM, les pipelines CI/CD et la gestion de code source.

#### Avantages :

- gestion puissante du backlog avec hiérarchie (epics, features, user stories, tasks) ;
- tableaux Scrum et Kanban performants ;
- intégration native avec Azure Pipelines (CI/CD) ;
- bonne traçabilité entre code, builds, tests et user stories ;
- reporting personnalisable via Analytics View ;
- adapté aux grandes organisations déjà sous Azure.

#### Limites :

- interface complexe et plus technique que Jira ;
- manque de flexibilité dans la personnalisation des workflows ;
- aucune prise en charge BPMN 2.0 ;
- nécessite souvent une expertise DevOps pour une utilisation complète.

### C.3.4 Synthèse comparative des outils

Outil	SCRUM complet	Automatisation BPMN	Complexité
Jira Software	Oui	Non	Élevée
Trello / Asana / Monday	Partiel	Non	Faible
Azure DevOps	Oui	Non	Moyenne/Élevée
Solution interne + Camunda	Oui	Oui	Variable

TABLE C.3.1 – Comparaison synthétique des principaux outils SCRUM

### C.3.5 Limites de la méthodologie SCRUM

Bien que la méthodologie SCRUM soit largement adoptée dans l'industrie et reconnue pour sa flexibilité, elle présente plusieurs limites dans un contexte réel d'entreprise. Ces limites apparaissent souvent lorsque SCRUM est appliqué sans outillage adapté ou dans un environnement complexe.

**a) Manque d'automatisation et forte dépendance au manuel** SCRUM repose sur un suivi opérationnel majoritairement manuel :

- gestion manuelle des blocages (*impediments*) ;
- suivi manuel de l'avancement des tâches (Daily, Board) ;
- absence d'alertes automatiques en cas de retard ou dépassement ;
- priorisation du backlog souvent gérée à la main.

Sans outils puissants, ces éléments peuvent entraîner lenteur, erreurs humaines ou manque de réactivité.

**b) Difficulté à standardiser les workflows** SCRUM décrit un cadre, mais laisse une grande liberté dans :

- les processus internes de validation ;
- les règles d'escalade ;
- la gestion des dépendances ;
- les étapes de QA, de revue et d'intégration.

Chaque équipe peut adopter sa propre version de SCRUM, rendant complexe :

- l'uniformisation entre plusieurs équipes ;
- la traçabilité des processus ;
- la gestion multi-projets.

**c) Difficulté à gérer les blocages (*impediments*)** Les impediments sont au cœur du rôle du Scrum Master mais SCRUM n'impose aucun mécanisme formel pour :

- détecter automatiquement un blocage ;
- notifier les personnes concernées ;
- escalader un problème si aucun progrès n'est réalisé ;
- monitorer les temps de résolution.

Dans les organisations, ce point est souvent source :

- de retards importants ;
- de perte de visibilité ;
- d'accumulation de dette technique.

**d) Manque de traçabilité si SCRUM n'est pas outillé** Sans un système robuste :

- les décisions ne sont pas historisées ;
- les changements de backlog ne sont pas tracés ;
- les actions de Daily ne sont pas suivies ;
- les problèmes récurrents ne sont pas archivés.

Cela peut nuire à la qualité du suivi projet et à l'amélioration continue.

**e) Gestion limitée du pilotage et des métriques** Les métriques SCRUM de base (burn-down, vitesse) sont utiles mais limitées.

Elles ne suffisent pas pour :

- analyser les goulots d'étranglement dans le workflow ;
- mesurer le temps moyen de résolution des blocages ;
- détecter les anomalies de flux (cycle time anormal, latence entre tâches) ;
- identifier les dépendances critiques non résolues.

SCRUM, dans sa forme brute, manque donc d'un véritable système d'analyse opérationnelle.

**f) Dépendance aux compétences humaines** SCRUM repose sur :

- un PO compétent pour prioriser correctement ;
- un Scrum Master expérimenté pour lever les obstacles ;
- une équipe disciplinée pour s'auto-organiser.

Lorsque ces conditions ne sont pas pleinement réunies, l'efficacité du cadre SCRUM peut chuter.

**g) Nécessité d'un outil complémentaire** Toutes ces limites montrent que SCRUM :

- nécessite un support logiciel robuste ;
- a besoin d'automatisations pour gagner en efficacité ;
- doit être complété par une gestion de processus formelle (BPMN).

**Liens avec notre projet** Ces limites justifient la création d'une plateforme :

- intégrant un moteur BPMN (Camunda) pour automatiser les workflows ;
- offrant un suivi centralisé et en temps réel des blocages ;
- fournissant des métriques avancées ;
- standardisant les processus SCRUM dans toute l'organisation.

Ainsi, notre solution vient répondre directement aux faiblesses actuelles des équipes utilisant SCRUM sans automatisation.

## C.4 BPMN 2.0 : un standard international

### C.4.1 Définition

La norme **BPMN 2.0, Business Process Model and Notation** est un standard international . Elle a pour objectif de modéliser de façon intelligible les processus métiers au sein de l'organisation de l'entreprise, de sorte que tous les employés puissent comprendre facilement chacun de ces processus. La norme est un langage visuel pour schématiser les processus les plus complexes.

### C.4.2 Objectifs et avantages

BPMN vise à :

- standardiser la représentation des processus ;
- faciliter la communication entre métiers et techniques ;
- préparer l'automatisation des workflows ;
- rendre les processus traçables et optimisables.

### C.4.3 Éléments graphiques de base

**Événements** représentés par un cercle et marquent un début, une fin ou une étape intermédiaire.

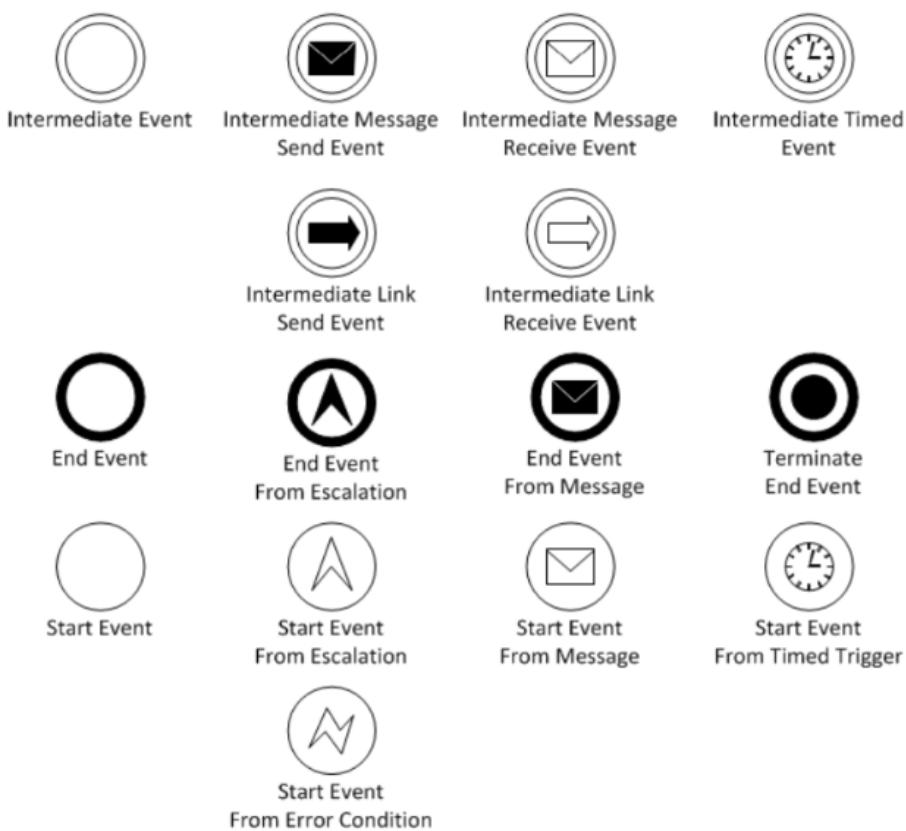


FIGURE C.4.1 – Symboles des événements BPMN

**Activités** qui permettent d'exécuter des tâches, des sous-processus et des activités, sont des actions à réaliser pour atteindre l'objectif du processus. Elles peuvent impliquer une activité humaine comme les tâches utilisateurs et manuelles, mais peuvent également être automatisées dans le cadre de tâches, services et script. Elles sont représentées par un rectangle aux coins arrondis.

- **Task** : tâche simple.
- **Sub-process** : processus imbriqué.
- **Call Activity** : appel d'un autre processus.

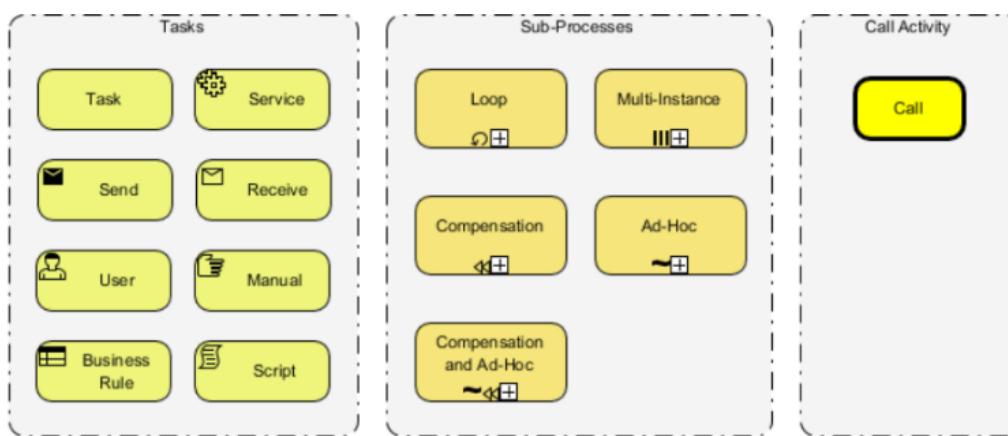


FIGURE C.4.2 – Symboles des activités BPMN

**Passerelles** sont des étapes dans le processus de divergences ou convergences du flux. Elles sont en capacité de décrire le chemin de flux de séquence d'un processus. Elles sont représentées par des losanges, au centre duquel on peut retrouver un symbole qui indique les sorties de flux possibles.



FIGURE C.4.3 – Symboles des passerelles BPMN

**Connecteurs** qui permettent de relier les différentes étapes entre elles.



FIGURE C.4.4 – Symboles des connecteurs BPMN

#### C.4.4 Exemples de processus BPMN

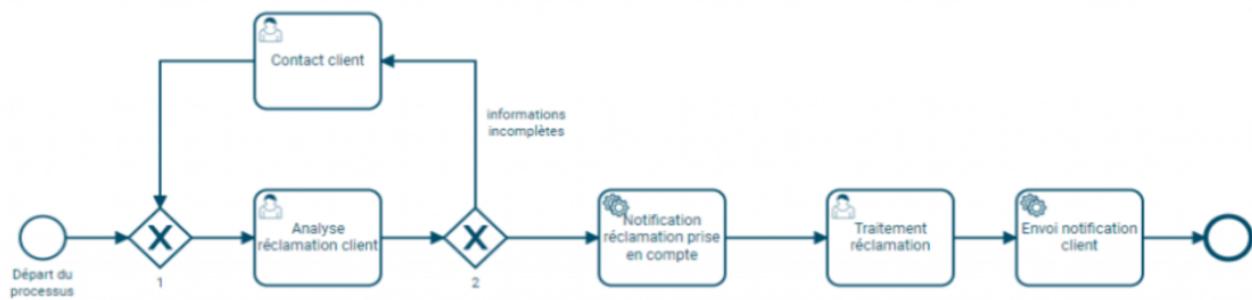


FIGURE C.4.5 – Exemple de processus BPMN : Analyse d'une réclamation

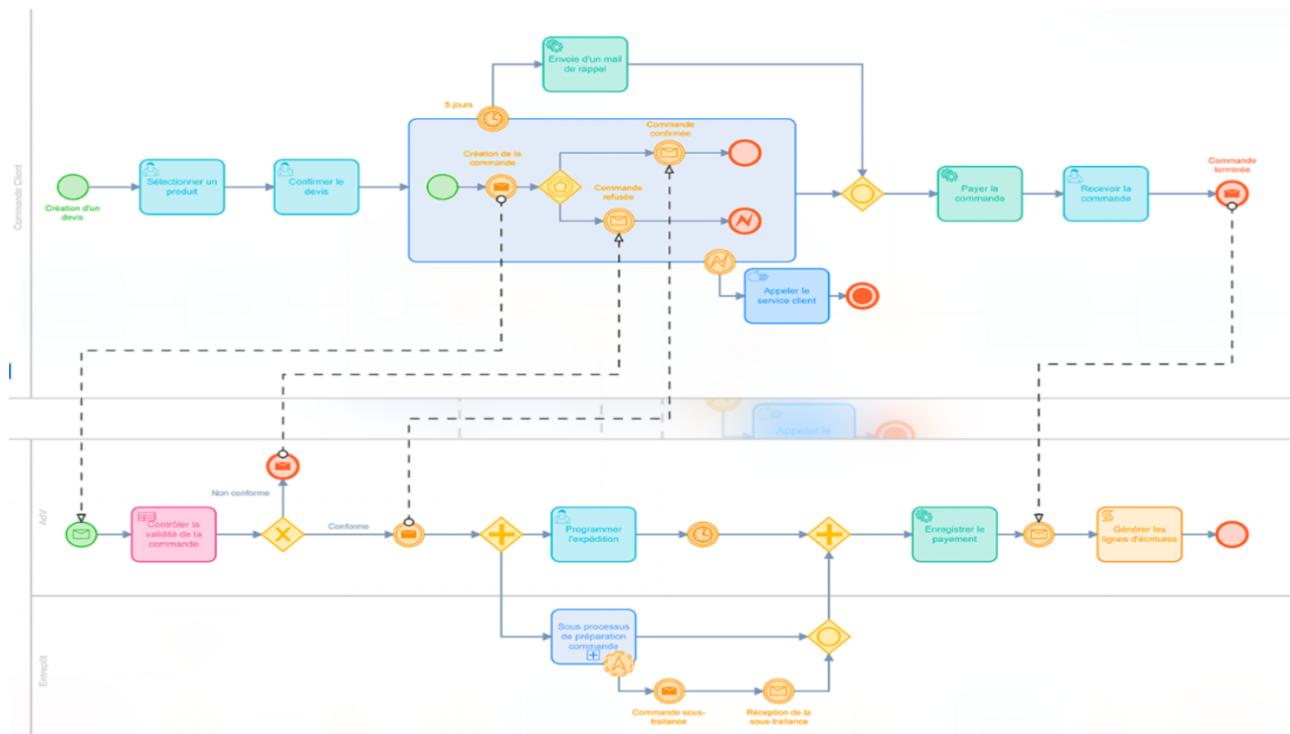


FIGURE C.4.6 – Exemple de processus BPMN : Création d'un devis

### C.5 Camunda BPM

#### C.5.1 Présentation générale

Camunda est une plateforme open source dédiée à la gestion et à l'automatisation des processus métier. Il s'agit d'un framework Java capable d'exécuter :

- des modèles BPMN 2.0 pour l'automatisation des processus ;
- des modèles DMN pour la gestion des règles de décision ;
- des modèles CMMN pour les cas complexes.

Grâce à sa compatibilité native avec Java et Spring Boot, Camunda est largement utilisé dans les applications web d'entreprise nécessitant un moteur workflow robuste, flexible et facile à intégrer.

### C.5.2 Rôle de Camunda dans l'automatisation BPMN

Camunda permet d'exécuter directement des diagrammes BPMN 2.0 en transformant chaque élément du modèle en une étape exécutable. Concrètement, chaque tâche, passerelle ou événement défini dans le diagramme devient une action réalisable par le moteur.

**Une étape BPMN peut être :**

- exécutée automatiquement (appel d'un service REST, script, automatisation interne) ;
- assignée à un utilisateur via les tâches humaines (*User Tasks*) ;
- monitorée en temps réel via Camunda Cockpit.

Camunda apporte ainsi :

- automatisation ;
- traçabilité complète ;
- supervision centralisée ;
- optimisation continue des workflows.

### C.5.3 Architecture générale de Camunda 7

L'architecture de Camunda 7 est conçue pour être simple, légère et adaptée à un déploiement local ou embarqué dans une application Spring Boot.

Elle repose sur les composants suivants :

- **Process Engine (moteur BPMN Java)** : exécute les modèles BPMN, DMN et CMMN ;
- **Base de données SQL** : stocke modèles, instances, historique, utilisateurs et autorisations ;
- **API REST** : permet d'interagir avec le moteur depuis une application externe (Angular, Postman, etc.) ;
- **Interfaces Web** :
  - **Camunda Cockpit** : supervision en temps réel des processus ;
  - **Camunda Tasklist** : gestion des tâches manuelles et formulaires ;
  - **Camunda Admin** : gestion des utilisateurs, rôles et autorisations.

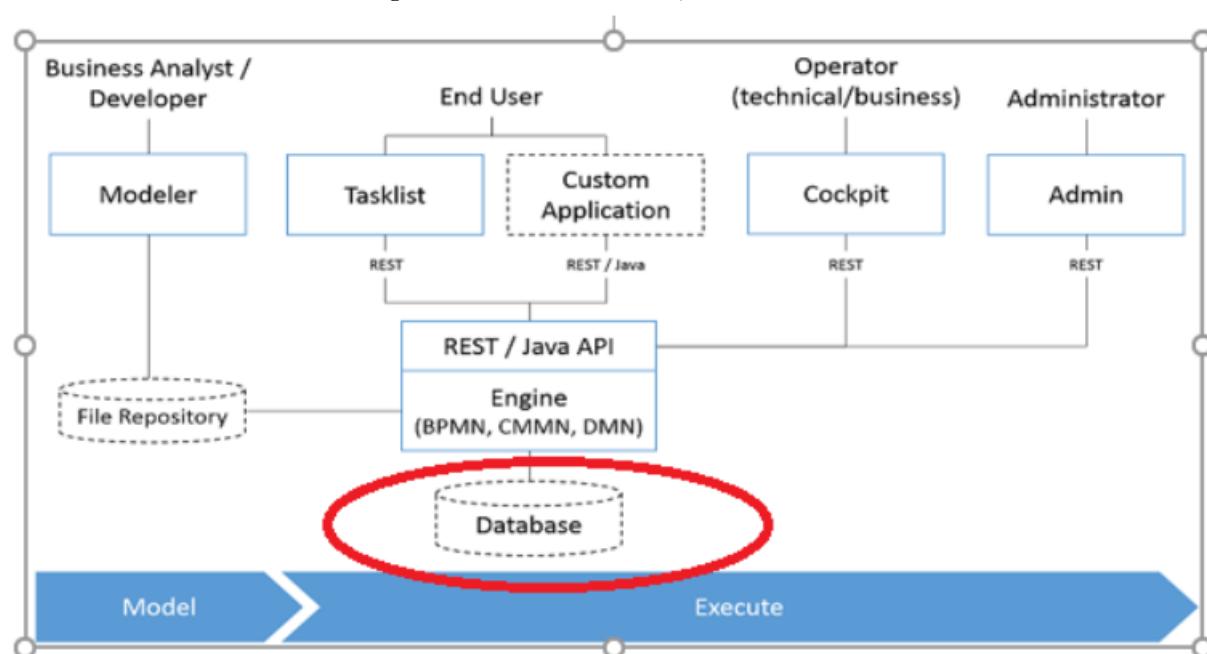


FIGURE C.5.1 – Architecture générale de Camunda 7

#### C.5.4 Camunda 7 vs Camunda 8

Camunda propose aujourd’hui deux versions majeures, répondant à des besoins différents.

##### a) Tableau comparatif

Critère	Camunda 7	Camunda 8
<b>Moteur de workflow</b>	Java embeddable (Camunda Engine)	Zeebe (moteur distribué orienté microservices)
<b>Architecture</b>	Monolithique ou embarquée	Cloud native, microservices, scalable
<b>Déploiement</b>	Local, Docker, Spring Boot	Kubernetes, Docker, SaaS (Camunda Cloud)
<b>Communication</b>	REST synchrones	gRPC asynchrone
<b>Stockage</b>	Base de données SQL	Event sourcing + Elasticsearch
<b>Monitoring</b>	Cockpit (intégré)	Operate, Optimize (services séparés)
<b>Modélisation</b>	BPMN/DMN avec Camunda Modeler	BPMN/DMN avec Camunda Modeler

TABLE C.5.1 – Comparaison détaillée entre Camunda 7 et Camunda 8

##### b) Différences principales

- **Moteur** : Camunda 7 utilise un moteur Java embarqué (*camunda-engine*), tandis que Camunda 8 repose sur Zeebe, un moteur distribué orienté microservices.
- **Technologies** : Camunda 8 privilégie Kubernetes, Elasticsearch et gRPC, tandis que Camunda 7 repose sur Java et SQL.
- **Approche développement** : Camunda 7 utilise les *Java Delegates*. Camunda 8 utilise des *job workers* en gRPC (multi-langage).
- **Monitoring** : Camunda 7 intègre directement Cockpit. Camunda 8 nécessite plusieurs outils externes (Operate, Optimize...).
- **Déploiement** : Camunda 7 est simple à déployer (local, Docker, Spring Boot). Camunda 8 demande une infrastructure cloud ou Kubernetes.

##### c) Cas d’usage recommandés

- **Camunda 7** Adapté aux projets internes, aux applications embarquées, aux systèmes Spring Boot. Simple à installer, monitorer et exécuter en local.
- **Camunda 8** Recommandé pour des systèmes hautement distribués, des architectures cloud natives, et des besoins de scalabilité horizontale.

#### C.5.5 Pourquoi Camunda 7 pour ce projet ?

Pour les besoins de notre projet, Camunda 7 représente le choix le plus pertinent car :

- il s’intègre parfaitement dans un environnement Java/Spring Boot ;
- il ne nécessite pas de maîtrise de Kubernetes ou d’infrastructures cloud avancées ;

- il dispose d'une interface de monitoring intégrée (Cockpit) très utile pour le suivi ;
- son moteur BPMN est robuste et largement suffisant pour nos workflows SCRUM ;
- il permet un déploiement rapide en local via Docker.

Camunda 7 offre donc un excellent compromis entre simplicité, efficacité et compatibilité technique.

## C.6 Camunda Modeler

### C.6.1 Son rôle

Camunda Modeler est l'outil officiel de modélisation graphique fourni par Camunda. Il permet de créer :

- des modèles BPMN 2.0 (processus métier),
- des modèles DMN (règles de décision),
- des modèles CMMN (cas complexes).

Ces modèles sont ensuite directement déployables dans le moteur Camunda.

### C.6.2 Avantages pour les développeurs et analystes métier

- **Visualisation claire** : représentation graphique des processus sans coder ;
- **Collaboration facilitée** entre profils techniques et fonctionnels ;
- **Productivité accrue** : configuration rapide des tâches automatiques, scripts, formulaires ;
- **Modèles exécutables directement** sans conversion complexe ;
- **Supports natifs pour Java Delegates** et listeners Camunda 7.

### C.6.3 Intégration avec Camunda 7

- Les modèles BPMN sont enregistrés sous forme de fichiers .bpnn ;
- Ils peuvent être déployés via l'interface web, l'API REST, ou intégrés au code Java ;
- Le Modeler permet de configurer directement :
  - les *Java Delegates*,
  - les formulaires utilisateurs,
  - les conditions et expressions,
  - les écouteurs d'événements (Event Listeners).
- Compatible avec l'architecture embarquée de Camunda 7, garantissant un cycle de développement rapide.

## C.7 Conclusion du chapitre

L'association BPMN 2.0, Camunda 7 et Camunda Modeler forme un écosystème performant pour concevoir, exécuter et superviser des processus métier automatisés. Dans le contexte de ce projet, Camunda 7 a été retenu pour sa simplicité de déploiement, sa compatibilité avec Spring Boot, et son adéquation naturelle avec une application web interne. Ce socle technologique constitue une base solide pour la mise en œuvre de workflows SCRUM automatisés, qui seront présentés dans le chapitre suivant.

# CHAPITRE D

# Analyse et Conception du Système

## D.1 Introduction

Ce chapitre présente l'analyse fonctionnelle et technique du système développé, ainsi que la conception logicielle et architecturale. Il s'appuie sur :

- les besoins exprimés par l'entreprise,
- la documentation fournie,
- le code réel du frontend Angular et backend Spring Boot,
- l'architecture décrite dans les documents techniques,
- l'intégration planifiée du moteur BPMN Camunda.

L'objectif est de présenter une vision claire, cohérente et professionnalisée du fonctionnement interne de la plateforme.

## D.2 Analyse fonctionnelle

### D.2.1 Description générale du système

Le système développé est une application web destinée à digitaliser et automatiser la gestion des projets Agile SCRUM. Il permet :

- la gestion des utilisateurs et des rôles,
- la gestion des projets,
- la constitution des équipes (PO, SM, Dev),
- la gestion du Product Backlog,
- la gestion des sprints et des tâches,
- la détection et la gestion des blocages (*impediments*),
- la tenue des réunions SCRUM,
- l'intégration future de workflows BPMN via Camunda.

### D.2.2 Acteurs du système

- **Administrateur** : gère les utilisateurs, les rôles et les accès.
- **Product Owner (PO)** : crée et priorise le backlog, pilote la vision du produit.

- **Scrum Master (SM)** : supervise les sprints, résout les blocages, facilite SCRUM.
- **Développeur** : réalise les tâches techniques définies dans le Sprint Backlog.
- **Système Camunda** (acteur logiciel, à venir) : exécute les processus BPMN.

### D.2.3 Besoins fonctionnels

- Authentification sécurisée (JWT).
- Gestion des rôles et permissions (RBAC).
- CRUD complet sur les projets, sprints, stories, tasks.
- Système de notifications (emails, alertes UI, futur BPMN).
- Dashboard PO et SM (vitesse, burndown, bloqueurs, avancement).

### D.2.4 Besoins non fonctionnels

- **Performance** : temps de réponse rapide grâce au cache Angular + API REST optimisée.
- **Sécurité** : JWT, refresh tokens, validation backend.
- **Scalabilité** : micro-services Docker, backend stateless.
- **Maintainability** : architecture modulaire en couches (frontend + backend + BPMN).

## D.3 Modélisation des exigences

### D.3.1 Diagramme des cas d'utilisation

Le diagramme de cas d'utilisation ci-dessous présente l'ensemble des interactions possibles entre les acteurs du système (Administrateur, Product Owner, Scrum Master, Développeurs) et l'application.

Il illustre les fonctionnalités majeures : gestion des utilisateurs, gestion des projets, gestion du Product Backlog, préparation des sprints, mise à jour des tâches, gestion des blocages et participation aux événements SCRUM. Ce diagramme constitue une vue globale du périmètre fonctionnel de la plateforme.

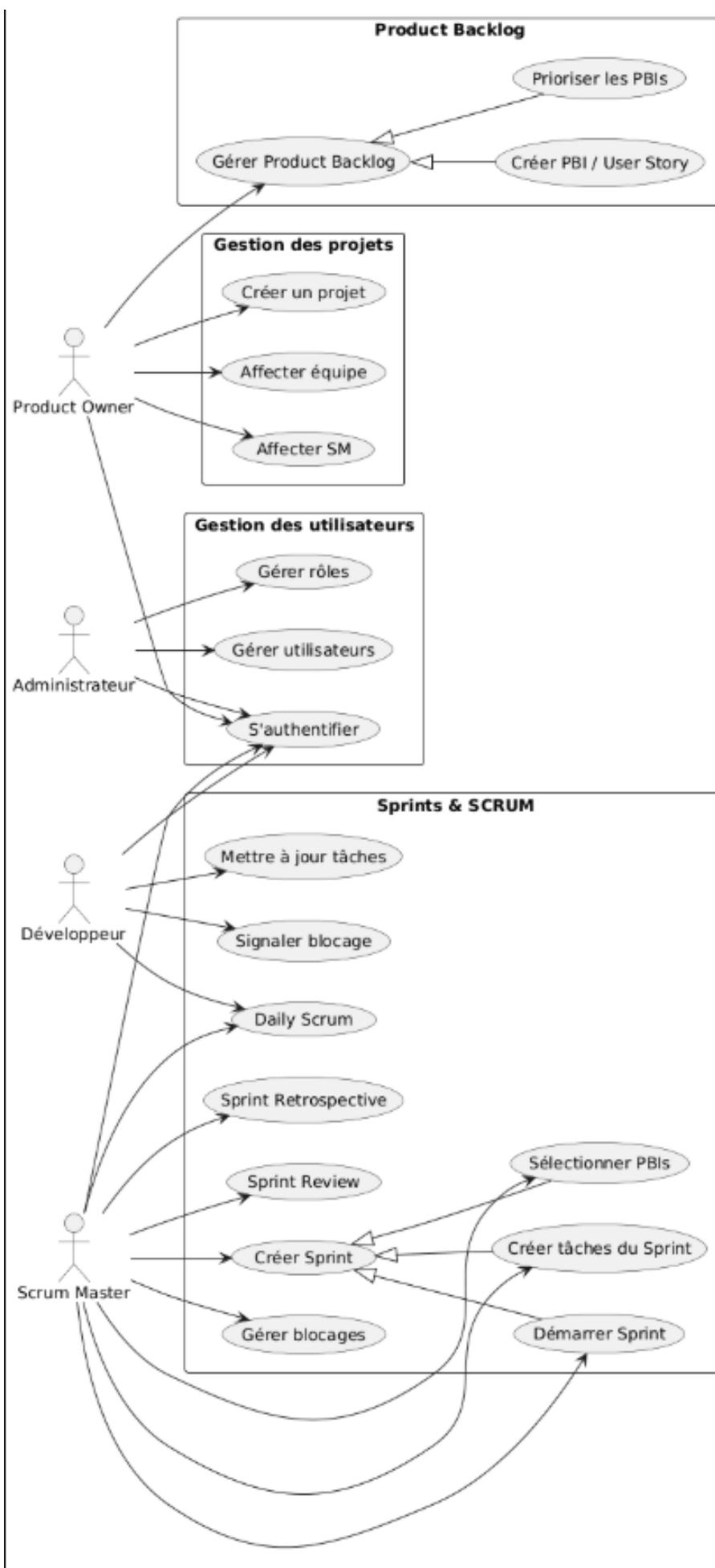


FIGURE D.3.1 – Diagramme des cas d'utilisation du système  
32

### D.3.2 Description détaillée de quelques cas d'utilisation

#### UC01 : Créer un projet

- Acteur principal : Product Owner
- Préconditions : être authentifié et avoir le rôle PO
- Scénario principal :
  - A. L'utilisateur accède à la page « Projets »
  - B. Il clique sur « Nouveau projet »
  - C. Il saisit les informations du projet
  - D. Le système enregistre le projet

#### UC02 : Création du Product Backlog

- **Acteur principal :** Product Owner
- **Description :** Le Product Owner crée les éléments du Product Backlog (PBIs / User Stories), en précisant le titre, la description, les critères d'acceptation et la priorité.

#### UC03 : Création du Sprint Backlog

- **Acteur principal :** Scrum Master
- **Description :** Le Scrum Master crée un sprint, sélectionne les PBIs provenant du Product Backlog, définit la capacité et crée les tâches techniques associées avant de démarrer le sprint.

## D.4 Modélisation UML

### D.4.1 Diagramme de classes

Le diagramme de classes ci-dessous présente la structure logique du système du point de vue objet. Il met en évidence les principales entités manipulées par l'application (User, Project, Story, Sprint, Task, Blocker) ainsi que leurs relations. Ce modèle orienté objet reflète fidèlement les entités du backend développé en Spring Boot, et constitue la base de la persistance des données dans PostgreSQL.

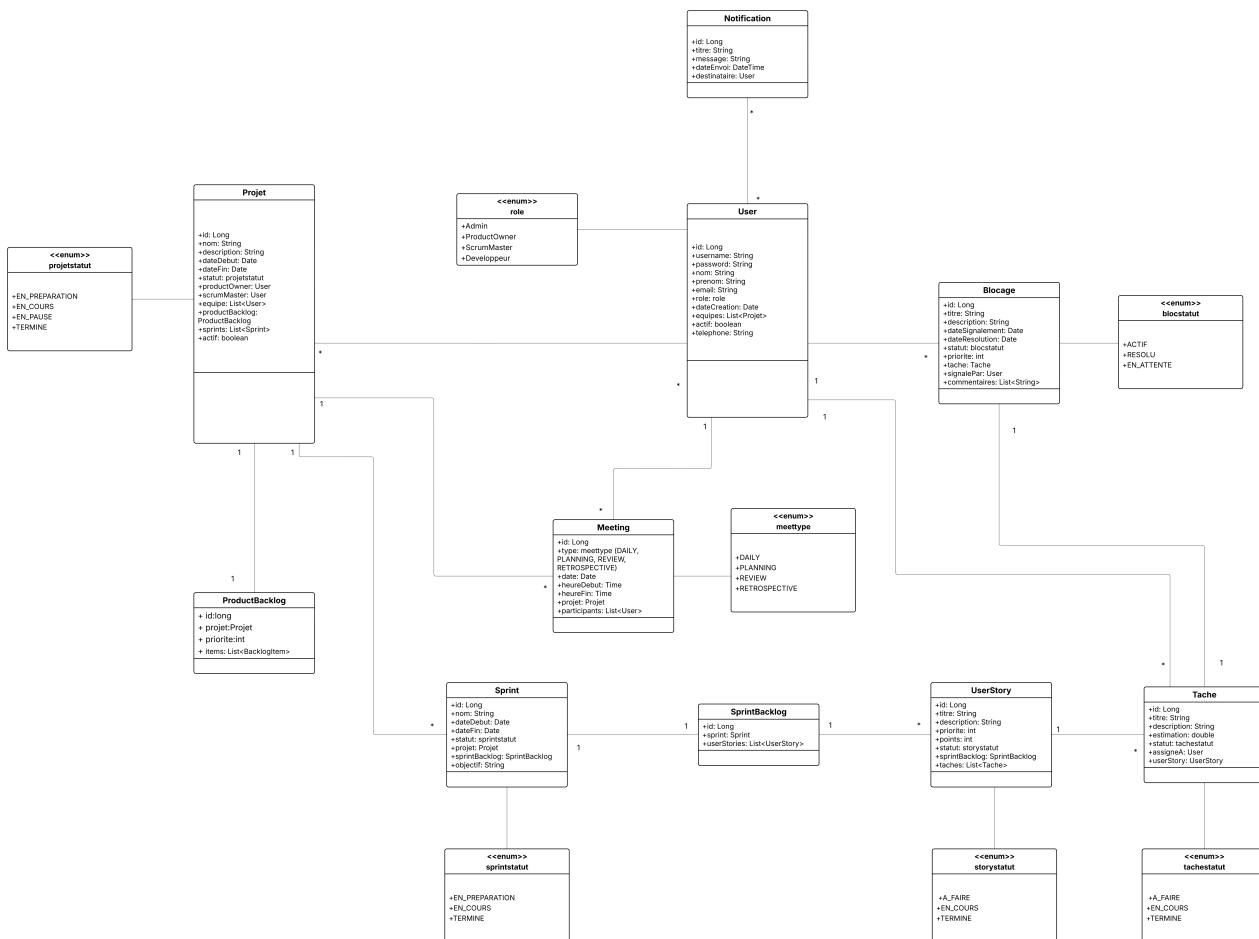


FIGURE D.4.1 – Diagramme de classes du système

Ce diagramme sera basé sur :

- **User, Role** -> extraction de ton backend
- **Project, Sprint, Story, Task**
- **Blocker, Meeting, Team**

## D.4.2 Diagrammes de séquence

### Authentification JWT

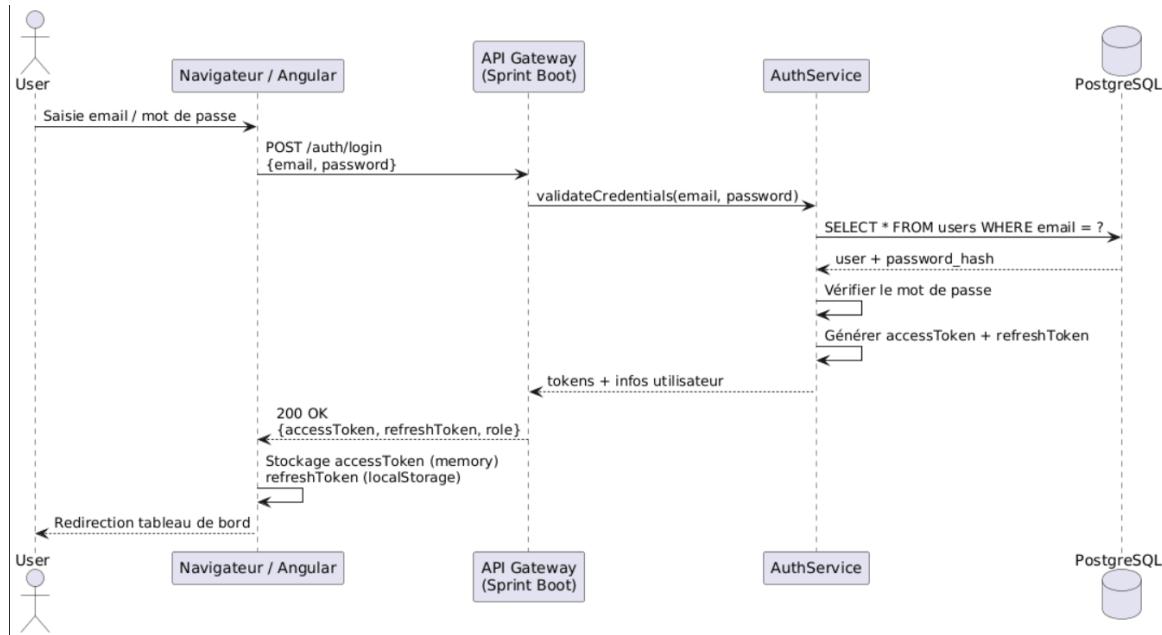


FIGURE D.4.2 – Diagramme de séquence — Authentification JWT

## Création d'un projet

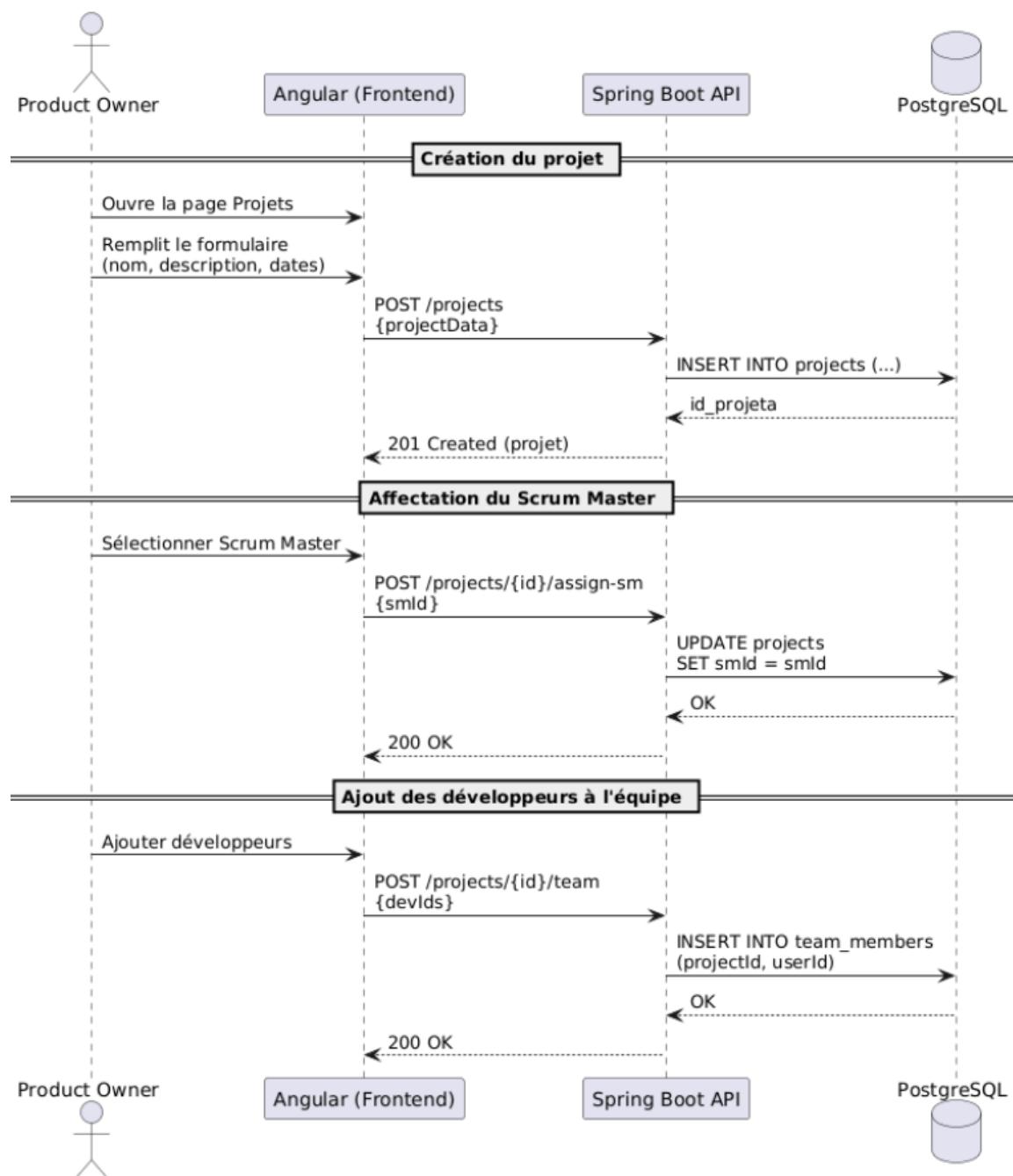


FIGURE D.4.3 – Diagramme de séquence — Crédit d'un projet

## Création d'un product backlog

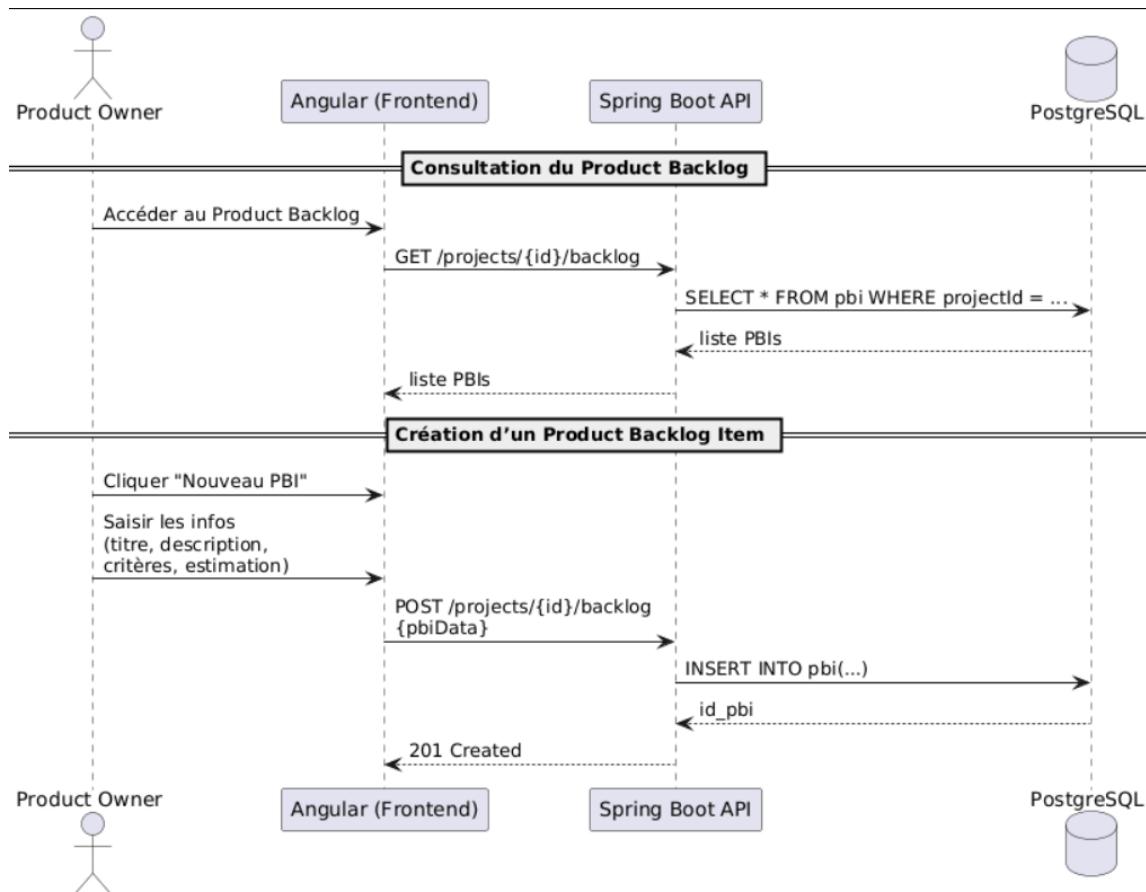


FIGURE D.4.4 – Diagramme de séquence — Crédit d'un product backlog

## Création d'un sprint backlog

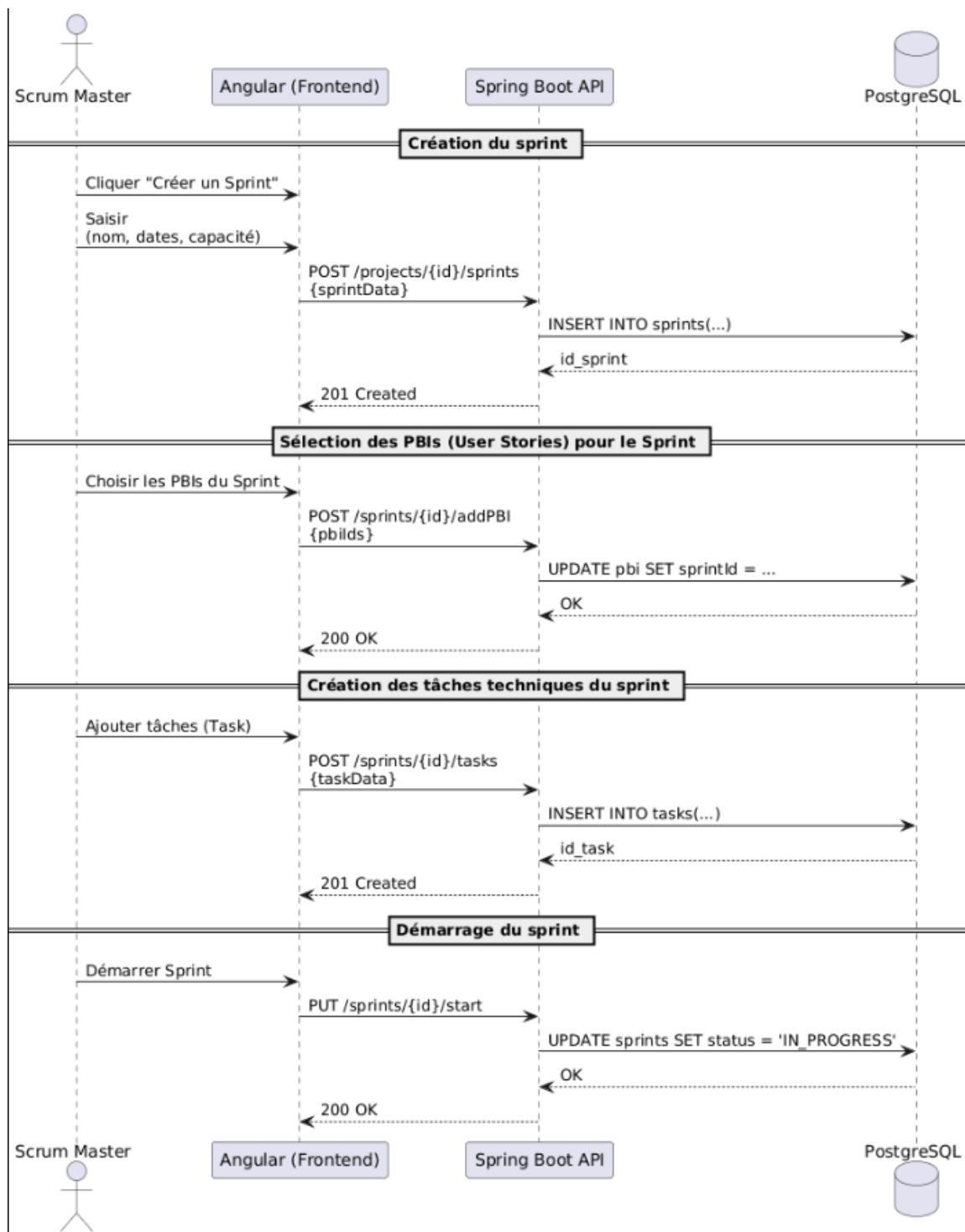


FIGURE D.4.5 – Diagramme de séquence — Crédit d'un sprint backlog

### D.4.3 Diagramme d'activité global

Le diagramme d'activité ci-dessous illustre le déroulement global d'un projet SCRUM dans la plateforme, en mettant en évidence le rôle de chaque acteur (Administrateur, Product Owner, Scrum Master et Développeurs) tout au long du cycle de vie d'un sprint.

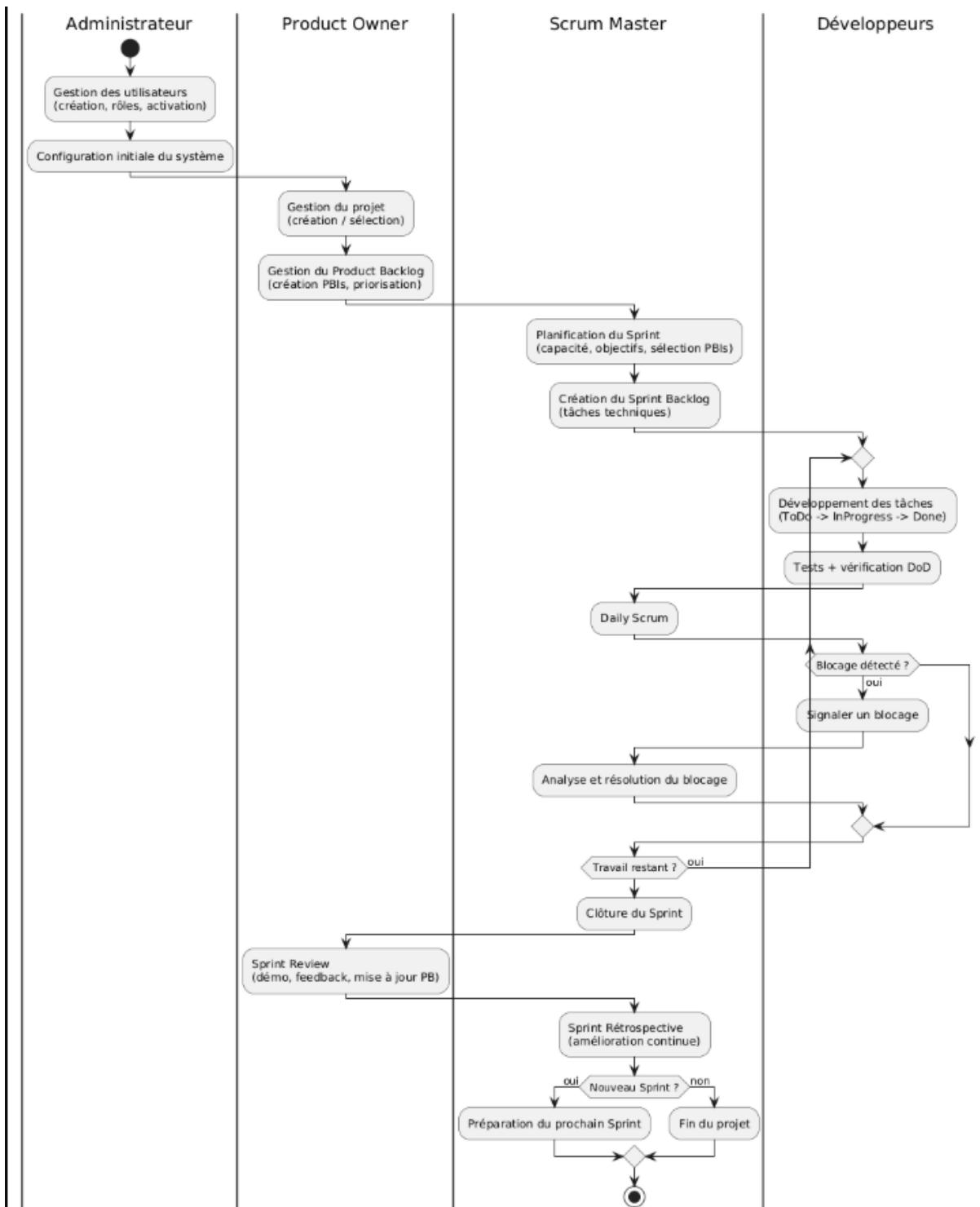


FIGURE D.4.6 – Diagramme d'activité global du système avec swimlanes

## D.5 Conception de l'architecture logicielle

### D.5.1 Vue d'ensemble

L'architecture du système repose sur une structure en trois couches cohérentes :

- une **interface web** développée en Angular (SPA) exécutée dans le navigateur ;

- un **backend Spring Boot** exposant une API REST sécurisée (JWT) et intégrant les règles métier du processus SCRUM ainsi que le moteur BPMN Camunda ;
- une **base de données PostgreSQL** pour la persistance des projets, utilisateurs, backlogs, sprints, tâches et blocages.

La Figure D.5.1 illustre les différents composants logiciels et leurs interactions. Ce diagramme est basé sur le modèle défini spécifiquement pour notre plateforme (Angular + Spring Boot + PostgreSQL + Camunda).

## Architecture logicielle du système (Hexagonale)

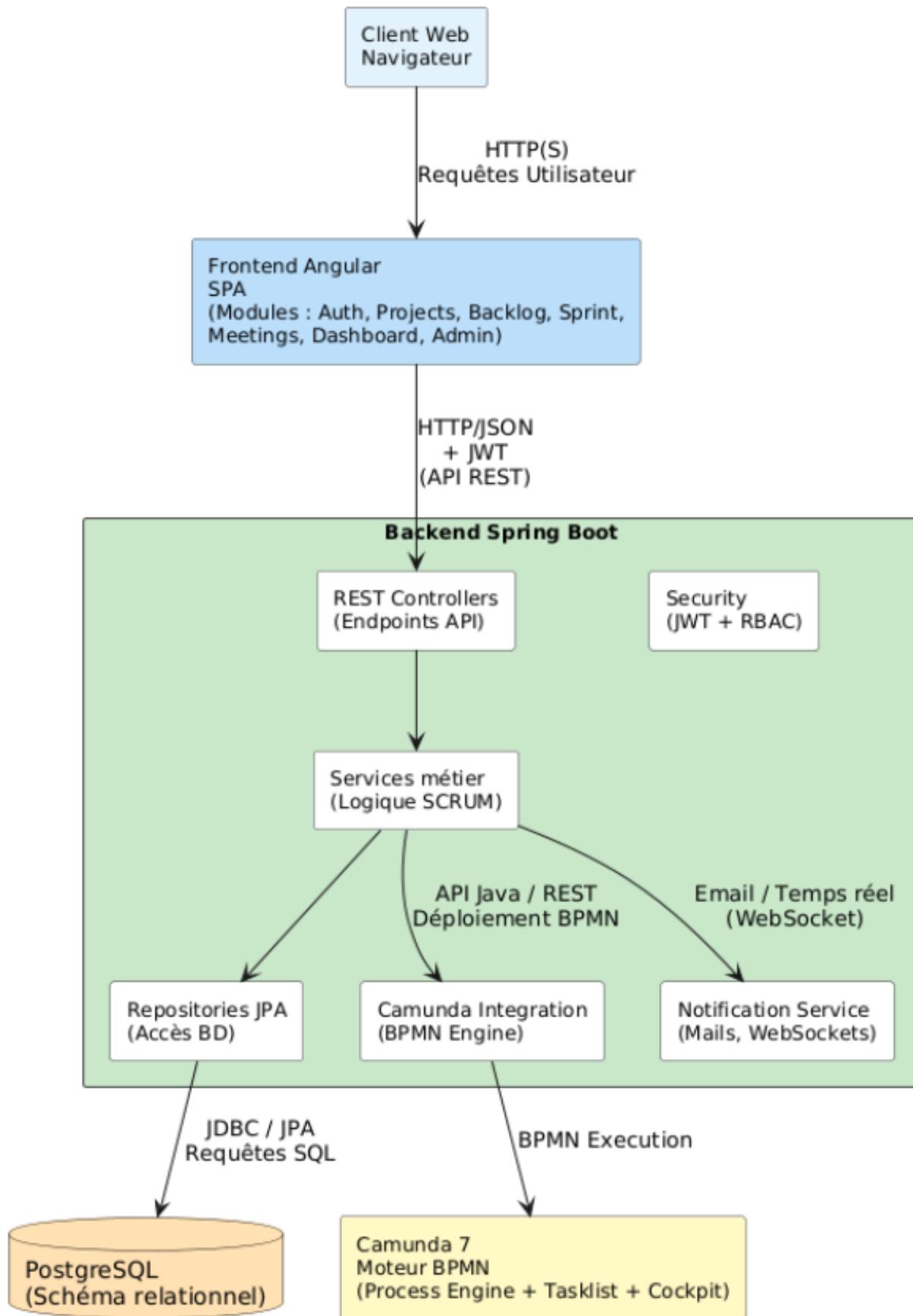


FIGURE D.5.1 – Architecture logicielle de la plateforme (Angular / Spring Boot / PostgreSQL / Camunda)

### D.5.2 Structure du frontend (Angular)

Le frontend est une application *Single Page Application* structurée en modules fonctionnels. Chaque module encapsule un ensemble de composants, services et routes :

- **Module Auth** : connexion sécurisée, gestion du token JWT, redirection selon rôle ;
- **Module Projets** : création, édition, gestion des équipes (PO, SM, Dev) ;
- **Module Backlog** : gestion du Product Backlog, création/édition des items, priorisation par glisser-déposer ;
- **Module Sprints** : création du Sprint Backlog, tableau Kanban (To Do / In Progress / Done), gestion des tâches de sprint ;
- **Module Meetings** : Daily Scrum, Sprint Review, Sprint Retrospective ;
- **Module Dashboard** : indicateurs PO/SM (vitesse, burndown, blocages, avancement) ;
- **Module Admin** : gestion des utilisateurs et des rôles.

Le frontend communique exclusivement avec le backend via HTTP/JSON en envoyant un jeton JWT dans l'en-tête **Authorization**. Il ne contient aucune logique métier : il gère uniquement l'affichage, les formulaires et la navigation.

### D.5.3 Structure du backend (Spring Boot)

Le backend est conçu selon une architecture en couches :

- **Controllers REST** : exposent les routes de l'API (`/auth`, `/projects`, `/backlog`, `/sprints`, `/blockers`, etc.) ;
- **Services métier** : implémentent les règles SCRUM (gestion des projets, backlog, sprint backlog, transitions Kanban, gestion des blocages) ;
- **Repositories JPA** : accès à la base de données via Spring Data JPA ;
- **Security (JWT + RBAC)** : filtre d'authentification, génération/validation des tokens, gestion des rôles (Admin, PO, SM, Dev) ;
- **Camunda Integration** : futur point d'intégration du moteur BPMN pour l'exécution de workflows (gestion des blocages, validation des user stories, automatisation de réunions SCRUM) ;
- **Notification Service** : envoi de notifications (email ou WebSocket) lors d'événements clés (création de projet, blocage détecté, début de sprint).

Ce découpage clarifie la responsabilité de chaque composant et facilite l'extension future du système, notamment pour la partie BPMN ou l'ajout de nouveaux modules.

### D.5.4 Persistance des données (PostgreSQL)

La base PostgreSQL contient l'ensemble des entités décrivant un environnement SCRUM : `users`, `roles`, `projects`, `stories`, `sprints`, `tasks`, `blockers`, `meetings`, `metrics`, etc. Le mapping entre classes Java et tables est assuré via les annotations JPA.

**Différence entre un diagramme de classes UML et un schéma de base de données :**

- Le **diagramme de classes UML** représente la structure logique du système : objets, attributs, relations, cardinalités. C'est une vue conceptuelle orientée développement.

- Le schéma de base de données représente la structure physique du stockage : tables, colonnes, clés primaires, clés étrangères, types SQL. C'est une vue technique orientée persistance.

Le diagramme UML exprime la modélisation métier, tandis que le schéma PostgreSQL exprime la manière dont ces données sont réellement enregistrées et liées dans la base.

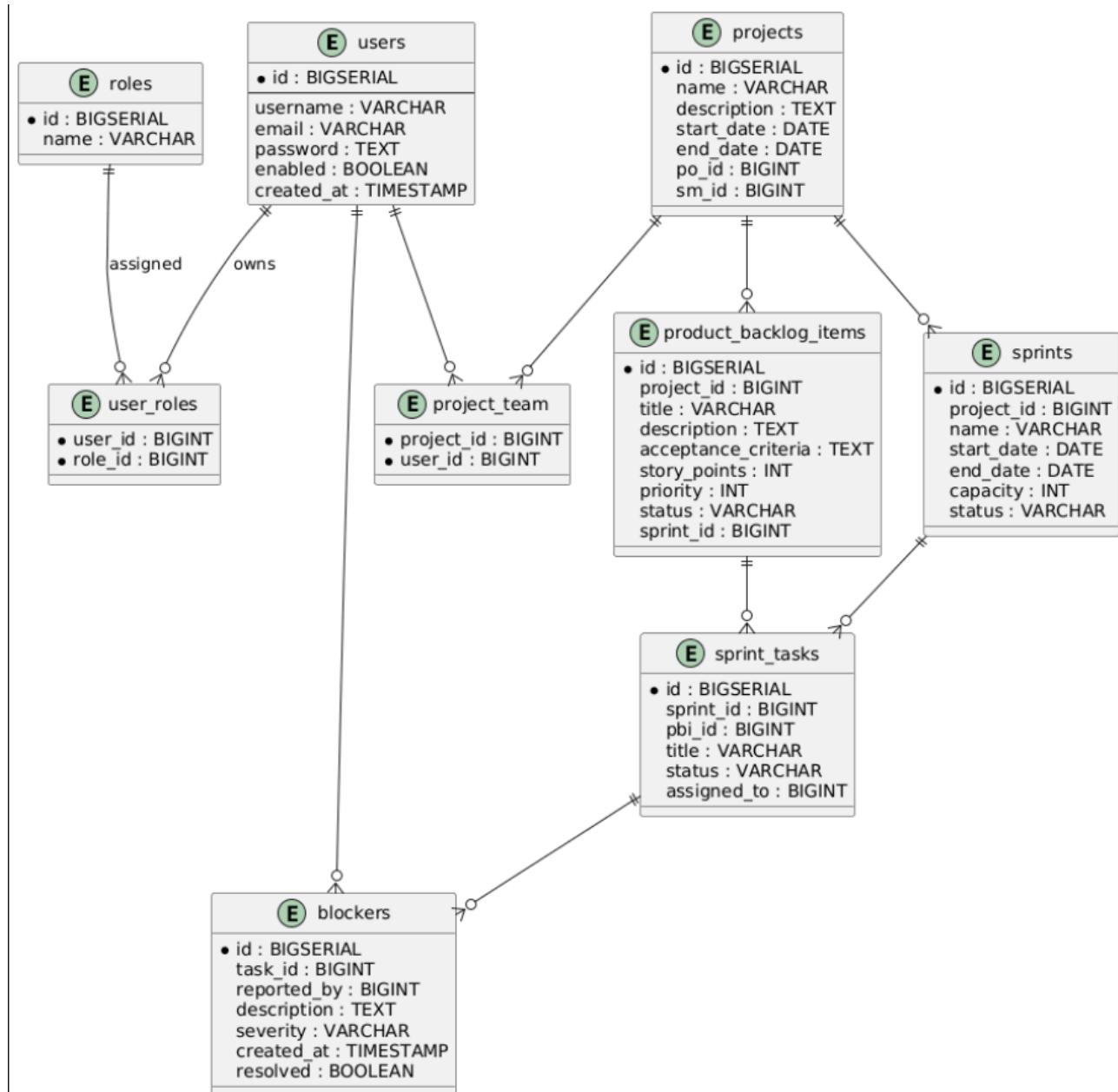


FIGURE D.5.2 – Schéma de la base de données PostgreSQL

## D.6 Intégration BPMN prévue avec Camunda

L'intégration de la notation BPMN 2.0 dans l'application a pour objectif de permettre au Scrum Master de modéliser, automatiser et superviser les processus clés du cadre SCRUM. Même si l'intégration technique de Camunda n'est pas encore finalisée, le backend Spring Boot est déjà structuré pour accueillir :

- le moteur Camunda 7 embarqué ;

- le déploiement automatique de fichiers BPMN ;
- l'exécution de processus métier liés à SCRUM ;
- une interface future permettant au Scrum Master de créer et modifier des processus BPMN.

Les processus BPMN suivants ont été modélisés dans le cadre du projet.

#### D.6.1 Processus de création d'un projet

Le premier diagramme représente le workflow de création et d'initialisation d'un projet SCRUM. Il modélise les interactions entre :

- le **Product Owner** : création du projet, définition du backlog, affectation des rôles ;
- le **Scrum Master et l'équipe de développement** : confirmation de disponibilité, participation aux réunions ;
- le système : gestion du lifecycle du meeting, des feedbacks et de la mise à jour du Product Backlog.

Ce processus BPMN permet de structurer les étapes initiales du projet, notamment la communication entre les acteurs SCRUM.

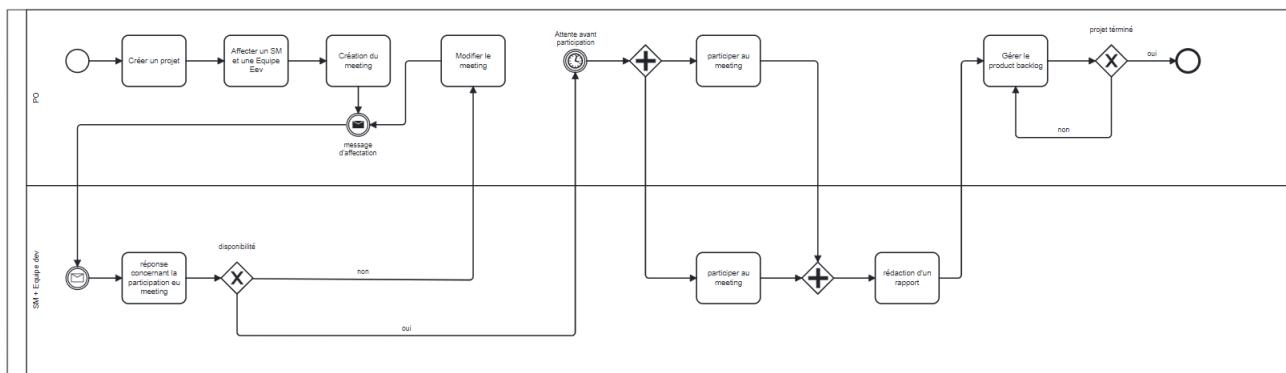


FIGURE D.6.1 – Processus BPMN — Création d'un projet

#### D.6.2 Processus SCRUM d'un Sprint

Le second diagramme modélise l'ensemble du déroulement d'un Sprint SCRUM, depuis le Sprint Planning jusqu'à la livraison de l'incrément. Il couvre :

- la planification du Sprint (PO + SM + Dev) ;
- la création du Sprint Backlog ;
- le développement des items et la validation DoD ;
- le Daily Scrum et la gestion des blocages ;
- la Sprint Review et la mise à jour du Product Backlog ;
- la Sprint Retrospective et la préparation du prochain Sprint.

Ce workflow BPMN formalise clairement le cycle de vie d'un Sprint et prépare son automatisation future via Camunda.

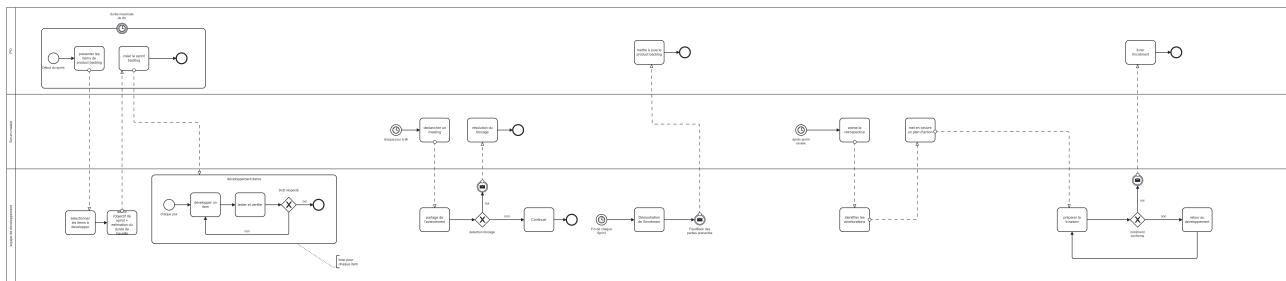


FIGURE D.6.2 – Processus BPMN — Sprint SCRUM

Avec ces deux modèles BPMN, la plateforme est prête à intégrer une orchestration automatisée des flux SCRUM grâce au moteur Camunda 7. Une fois l'intégration finalisée, chaque processus pourra être déclenché, suivi et supervisé directement depuis l'application via des API REST ou des événements métier.

## D.7 Architecture technique et déploiement

### D.7.1 Vue d'ensemble

L'architecture technique repose sur un environnement entièrement conteneurisé grâce à Docker. Chaque composant de la plateforme est exécuté dans un conteneur isolé, ce qui garantit la portabilité, la reproductibilité et la facilité de déploiement de l'application.

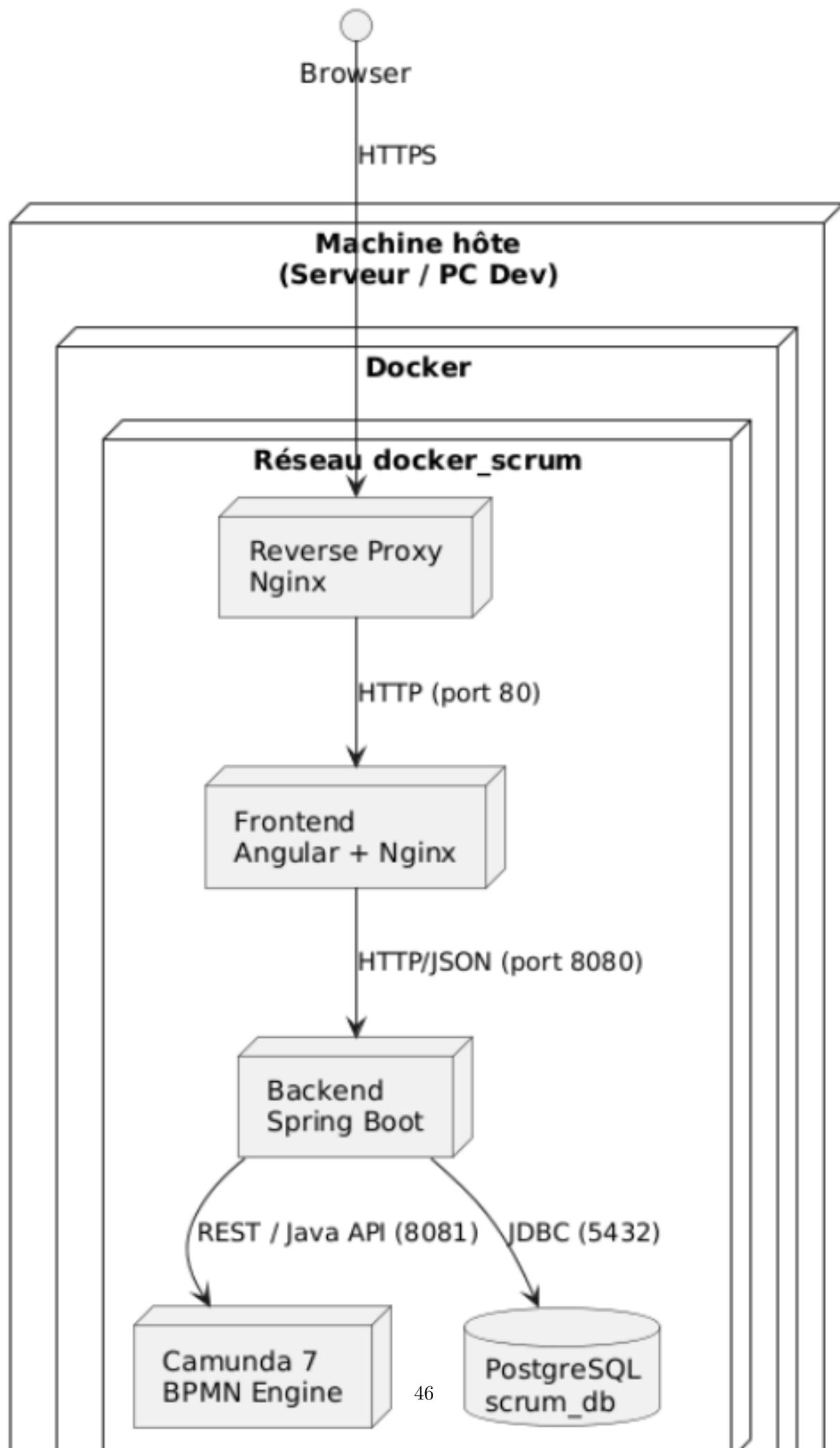
L'infrastructure regroupe les services suivants :

- **Reverse Proxy Nginx** : point d'entrée unique, gestion du routage et du HTTPS ;
- **Frontend Angular** : service statique Nginx hébergeant la SPA ;
- **Backend Spring Boot** : API REST exposant la logique métier et la sécurité JWT ;
- **Camunda 7** : moteur BPMN pour les workflows ;
- **PostgreSQL** : base de données relationnelle ;

L'ensemble fonctionne dans un réseau Docker interne qui assure la communication entre les différents conteneurs.

### D.7.2 Schéma technique de l'infrastructure

La figure D.7.1 montre la structure technique de l'environnement de déploiement, incluant les services Docker, leurs interactions et les flux réseau.



### D.7.3 Rôle des composants

#### Reverse Proxy Nginx

- assure le routage des requêtes entrantes ;
- gère le HTTPS en production ;
- redirige :
  - les requêtes / vers le frontend Angular ;
  - les requêtes /api/ vers le backend Spring Boot.

#### Frontend Angular

- application statique servie par Nginx ;
- communication avec le backend uniquement via HTTP/JSON ;
- authentification via tokens JWT envoyés dans les en-têtes.

#### Backend Spring Boot

- gère la logique métier SCRUM ;
- implémente les services REST ;
- gère la sécurité avec Spring Security + JWT ;
- communique :
  - avec PostgreSQL via JPA ;
  - avec Camunda via son moteur embarqué ou l'API REST.

#### Camunda 7

- moteur BPMN embarqué ;
- prévu pour :
  - automatiser la gestion des blocages ;
  - automatiser les workflows SCRUM (futur) ;
- expose Cockpit, Tasklist et Admin sur le port 8081.

#### PostgreSQL

- stocke les utilisateurs, projets, sprints, backlog, tâches et blocages ;
- garantit l'intégrité référentielle ;
- reçoit toutes les transactions via Spring Data JPA.

### D.7.4 Réseau et ports utilisés

Les ports des services sont généralement les suivants :

- **80 / 443** : reverse proxy Nginx (HTTP / HTTPS) ;
- **4200** : Angular (en développement uniquement) ;
- **8080** : API Spring Boot ;
- **8081** : interface Camunda (Cockpit, Tasklist, Admin) ;
- **5432** : PostgreSQL ;

En production Docker, seuls les ports 80/443 et 8080/8081 sont exposés. Les autres restent internes au réseau Docker.

### D.7.5 Déploiement via Docker Compose

Le déploiement du système s'effectue via un fichier `docker-compose.yml`, qui orchestre :

- le lancement des conteneurs ;
- la création des réseaux ;
- les dépendances entre services ;
- les volumes persistants pour PostgreSQL.

Une fois configuré, l'ensemble de la plateforme peut être lancé avec :

---

```
1 docker-compose up -d
```

---

Ce mode de déploiement permet :

- une mise en place rapide sur n'importe quel environnement ;
- une isolation parfaite entre services ;
- une reproductibilité idéale pour les tests et démonstrations.

## D.8 Conclusion

Ce chapitre a détaillé l'ensemble de l'analyse fonctionnelle, UML, conception logicielle et architecture technique du système. La base est suffisamment robuste pour accueillir l'automatisation BPMN prévue avec Camunda 7, et le design en couches (Angular / Spring Boot / PostgreSQL) garantit une bonne maintenabilité et une évolutivité future.

## CHAPITRE E

# Implémentation du Backend Spring Boot

### E.1 Introduction

Le backend constitue le cœur de l'application. Il centralise l'ensemble de la logique métier, expose une API REST sécurisée, gère la persistance des données dans PostgreSQL, et prépare l'intégration du moteur BPMN Camunda pour l'automatisation des processus SCRUM.

Développé avec Spring Boot, le backend repose sur une architecture modulaire en couches (Controller, Service, Repository, Entity, Security), ce qui garantit la maintenabilité, la testabilité et l'évolutivité du système. Ce chapitre détaille la conception technique, les entités principales, la logique SCRUM implémentée, les API REST, ainsi que la sécurité par jetons JWT.

### E.2 Architecture du backend

Le backend a été conçu selon une architecture en couches, inspirée des principes de séparation des responsabilités et de l'architecture hexagonale. L'objectif est de dissocier clairement :

- l'exposition des fonctionnalités (*Controllers REST*) ;
- la logique métier (*Services*) ;
- l'accès aux données (*Repositories JPA*) ;
- les aspects techniques transverses (*sécurité JWT, gestion des erreurs, intégration Camunda*).

Cette structure facilite la maintenabilité du code, la réutilisation des services métier et l'ajout de nouvelles fonctionnalités sans remettre en cause l'architecture globale.

#### E.2.1 Organisation en couches

Le backend est organisé en plusieurs couches logicielles :

- **Couche présentation (API)** : matérialisée par les *Controllers*. Ils reçoivent les requêtes HTTP provenant du frontend Angular, valident les données d'entrée, appellent les services métier appropriés, et renvoient des réponses au format JSON.

- **Couche métier** : composée des *Services* qui implémentent les règles SCRUM propres à l'application : création de projet, gestion du Product Backlog, préparation de sprint, mise à jour des tâches, signalement des blocages, etc.
- **Couche d'accès aux données** : constituée des *Repositories* Spring Data JPA, en charge de la persistance des entités dans la base PostgreSQL. Cette couche isole le reste du code des détails SQL.
- **Couche technique transversale** : elle regroupe notamment :
  - la configuration **Spring Security** (filtres JWT, gestion des rôles) ;
  - la gestion des exceptions globales ;
  - la **pré-intégration Camunda** (configuration du moteur BPMN, points d'extension pour les workflows).

### E.2.2 Organisation des packages

Pour refléter cette architecture, le code est structuré en packages (espaces de noms) cohérents. À titre d'exemple, on peut distinguer :

- **config** : configuration générale Spring Boot, CORS, gestion des beans ;
- **security** : classes liées à l'authentification et à l'autorisation (filtres JWT, services de détails utilisateur, configuration des rôles) ;
- **controller** : REST controllers exposant les endpoints (**AuthController**, **ProjectController**, **BacklogController**, **SprintController**, etc.) ;
- **service** : services métier comme **ProjectService**, **StoryService**, **SprintService**, **TaskService**, **BlockerService** ;
- **repository** : interfaces JPA (**UserRepository**, **ProjectRepository**, **StoryRepository**, ...) ;
- **model ou entity** : entités JPA qui représentent les objets persistés (User, Project, Story, Sprint, Task, Blocker, etc.) ;
- **camunda** (prévu) : classes liées à la configuration et à l'intégration du moteur BPMN ;
- **exception** : exceptions personnalisées et gestionnaire global des erreurs.

Cette structuration claire simplifie la navigation dans le projet, réduit le couplage entre les composants et permet à plusieurs développeurs de travailler simultanément sur des parties différentes du backend.

### E.2.3 Cycle de traitement d'une requête

Lorsqu'un utilisateur interagit avec l'interface Angular, le traitement côté backend se déroule généralement selon les étapes suivantes :

- A. Le frontend envoie une requête HTTP vers un endpoint REST (par exemple : POST /api/projects) en incluant un jeton JWT valable dans l'en-tête **Authorization**.
- B. Le filtre de sécurité vérifie la validité du token (signature, expiration, rôles) et authentifie l'utilisateur.
- C. Le *Controller* concerné reçoit la requête, effectue les premières validations et appelle le service métier approprié.
- D. Le *Service* applique les règles métier : création ou mise à jour d'un projet, d'une story, d'un sprint, etc.
- E. Le *Repository* persiste les modifications dans PostgreSQL via JPA.

F. Le service renvoie un objet de réponse que le controller transforme en JSON.

G. Le frontend reçoit la réponse et met à jour l'interface utilisateur.

Ce cycle illustre la responsabilité de chaque couche et montre que la logique métier reste centralisée dans les services, ce qui facilite l'évolution future de l'application (notamment l'intégration de Camunda pour certains scénarios).

## E.3 Modélisation des entités (JPA)

La couche de persistance de l'application repose sur Spring Data JPA, qui permet de manipuler les données via des entités Java annotées. Les entités représentent les objets métier du cadre SCRUM : utilisateurs, projets, éléments du Product Backlog, sprints, tâches techniques et blocages.

Chaque entité est reliée aux autres par des associations (OneToMany, ManyToOne, ManyToMany) qui modélisent fidèlement les relations SCRUM : un projet contient un backlog, un sprint contient des stories, une story est composée de tâches, etc.

Les sections suivantes décrivent les entités principales du backend.

### E.3.1 Utilisateur et rôles

L'entité **User** représente les différents acteurs de l'application : Administrateur, Product Owner, Scrum Master et Développeur. Dans l'implémentation, tous les utilisateurs possèdent le rôle de base **USER**, auquel s'ajoutent éventuellement : **ADMIN**, **PO** ou **SM**.

Cette approche permet une gestion flexible des permissions (RBAC). Un utilisateur contient notamment :

- un identifiant unique ;
- un nom d'utilisateur ;
- un mot de passe chiffré ;
- une collection de rôles ;
- les projets auxquels il est associé.

Les relations JPA les plus importantes sont :

- **ManyToMany** : un utilisateur peut appartenir à plusieurs équipes de projets ;
- **OneToMany** : un PO ou un SM peut superviser plusieurs projets.

### E.3.2 Projet

L'entité **Project** constitue la racine fonctionnelle d'un environnement SCRUM. Elle contient :

- un nom et une description ;
- un Product Owner (**@ManyToOne**) ;
- un Scrum Master (**@ManyToOne**) ;
- une équipe de développeurs (**@ManyToMany**) ;



- une liste d'éléments du Product Backlog (**Story**) ;
- une liste de sprints associés.

Cette structure permet d'isoler proprement les données d'un projet et de respecter la logique SCRUM : chaque projet dispose de son backlog, son sprint backlog, son équipe et ses métriques.

### E.3.3 Stories (Product Backlog)

L'entité **Story** représente les éléments du Product Backlog. Dans SCRUM, une story décrit un besoin utilisateur, accompagné de critères d'acceptation et d'une estimation en points d'effort.

Une story contient :

- un titre et une description ;
- des critères d'acceptation ;
- une priorité (définie par le PO) ;
- une estimation en points ;
- une éventuelle inclusion dans un sprint.

Relations principales :

- **ManyToOne Project** : une story appartient à un projet ;
- **OneToMany Task** : chaque story est décomposée en tâches techniques.

### E.3.4 Sprint

L'entité **Sprint** représente un cycle SCRUM complet : planification, développement, revue et rétrospective.

Elle contient notamment :

- un nom de sprint ;
- les dates de début et de fin ;
- la capacité en points (vitesse cible) ;
- la liste des stories sélectionnées (**ManyToMany**) ;
- son état : préparé, en cours, terminé.

Cette entité permet de structurer l'avancement du projet et de suivre la réalisation de chaque incrément.

### E.3.5 Tâches techniques (Sprint Backlog)

L'entité **Task** représente les tâches du Sprint Backlog. Elles sont créées par le SM et l'équipe pendant le Sprint Planning.

Une tâche contient :

- un intitulé technique ;

- un statut Kanban : To Do, In Progress, Review, Done ;
- un développeur assigné ;
- le temps passé ;
- un lien avec la story correspondante.

Les relations JPA :

- **ManyToOne Story** : une tâche appartient à une seule story ;
- **ManyToOne Sprint** : une tâche est incluse dans un sprint en cours.

### E.3.6 Blocages (Impediments)

L'entité **Blocker** permet de suivre les obstacles rencontrés par les développeurs : dépendance non résolue, anomalie bloquante, absence d'environnement, etc.

Elle contient :

- une description du blocage ;
- la date de détection ;
- l'utilisateur qui l'a signalé ;
- le Scrum Master responsable de la résolution ;
- un statut : ouvert, en cours, résolu.

Cette entité est essentielle pour le tableau de bord du Scrum Master et l'optimisation des processus SCRUM.

## E.4 Repositories et accès aux données

L'accès à la base de données est assuré par **Spring Data JPA**, qui permet de manipuler les entités de manière déclarative sans écrire de SQL. Chaque entité métier (**User**, **Project**, **Story**, **Sprint**, **Task**, **Blocker**) dispose d'un repository dédié, responsable des opérations CRUD et de la récupération de données spécifiques au fonctionnement SCRUM.

Les repositories héritent de **JpaRepository**, ce qui leur fournit automatiquement les opérations de base :

- **save()** — création ou mise à jour ;
- **findById()** — recherche par identifiant ;
- **findAll()** — récupération complète d'une table ;
- **deleteById()** — suppression d'un élément ;
- pagination et tri.

Spring permet également de créer des requêtes personnalisées via :

- les **méthodes dérivées** : exemple **findByIdProjectId()**,
- des requêtes JPQL avec **@Query**.

Ce mécanisme fournit une couche d'accès aux données simple, robuste et parfaitement intégrée au modèle objet.

#### E.4.1 UserRepository

Le **UserRepository** gère l'accès aux utilisateurs, nécessaires pour l'authentification JWT et la gestion des rôles.

Méthodes courantes :

- `findByUsername(String username)`
- `existsByUsername(String username)`
- `findByRolesName(String role)`

Utilisation :

- connexion et génération de tokens,
- affectation des rôles PO, SM, Admin, Dev,
- récupération des membres d'une équipe projet.

#### E.4.2 ProjectRepository

Le **ProjectRepository** offre la gestion des projets SCRUM. Il permet :

- de récupérer les projets créés par un Product Owner ;
- d'obtenir les projets associés à un Scrum Master ;
- de charger l'équipe assignée au projet ;
- de récupérer le backlog associé.

Méthodes dérivées possibles :

- `findByProductOwnerId(Long id)`
- `findByScrumMasterId(Long id)`

#### E.4.3 StoryRepository

Le **StoryRepository** assure la gestion du Product Backlog. Il permet :

- de récupérer toutes les stories d'un projet ;
- de trier les items selon leur priorité ;
- de connaître les éléments non planifiés (pour Sprint Planning).

Exemples de méthodes dérivées :

- `findByProjectId(Long id)`
- `findBySprintIsNullOrderByPriorityAsc()`

#### E.4.4 SprintRepository

Le **SprintRepository** gère les cycles SCRUM. Il permet par exemple :

- de lister les sprints d'un projet ;
- de récupérer le sprint actif ;
- d'obtenir ses stories planifiées.

Ce repository est central pour la mécanique du Sprint Planning.

#### E.4.5 TaskRepository

Le **TaskRepository** permet de manipuler les tâches du Sprint Backlog. On y retrouve notamment :

- les tâches associées à une story ;
- les tâches d'un sprint ;
- les tâches assignées à un développeur.

Méthodes typiques :

- `findBySprintId(Long id)`
- `findByDeveloperId(Long id)`

#### E.4.6 BlockerRepository

Les blocages (**Blocker**) sont un élément fondamental du suivi SCRUM. Le repository associé permet :

- de lister les blocages ouverts ;
- de filtrer ceux signalés par un développeur ;
- de récupérer les blocages résolus par le Scrum Master ;
- de suivre l'évolution de leur statut.

Ce repository alimente directement le tableau de bord du Scrum Master.

### Synthèse

L'ensemble des repositories forme la couche d'accès aux données de l'application. Grâce à Spring Data JPA :

- le code est plus concis ;
- les accès BD sont sécurisés et optimisés ;
- la logique métier des services reste isolée ;
- la maintenance et l'évolution du modèle deviennent simples.

Cette organisation respecte les bonnes pratiques de conception en couches (back-end REST), et facilite l'intégration future du moteur BPMN Camunda.

### E.5 Services métier

La couche **service** joue un rôle central dans l'architecture du backend. Elle contient toute la logique métier liée au fonctionnement de SCRUM dans l'application : gestion des projets, organisation du Product Backlog, préparation des sprints, mise à jour des tâches et suivi des blocages.

Un service Spring Boot est annoté avec `@Service` et intervient comme intermédiaire entre :

- les **controllers REST** (communication avec le frontend) ;

- les **repositories JPA** (accès aux données).

Cette structure permet d'isoler les règles métier et de maintenir un code clair, réutilisable et testable.

### E.5.1 UserService

Le **UserService** gère l'ensemble des opérations liées aux utilisateurs :

- création d'un utilisateur ;
- affectation et gestion des rôles ;
- récupération des membres d'une équipe ;
- chargement des informations utilisateur pour l'authentification JWT.

Le service repose fortement sur :

- **UserRepository** pour la persistance,
- **PasswordEncoder** pour le chiffrement des mots de passe.

Il constitue la base du module d'administration et de sécurité.

### E.5.2 ProjectService

**ProjectService** implémente la logique SCRUM liée aux projets :

- création d'un projet ;
- affectation du Product Owner, Scrum Master et des développeurs ;
- récupération du backlog du projet ;
- mise à jour des informations générales du projet.

Ce service orchestre également l'association entre projets, équipes et éléments du backlog.

### E.5.3 StoryService (Product Backlog)

Le **StoryService** gère les éléments du Product Backlog.

Il permet :

- la création de nouvelles stories ;
- la modification et la priorisation par le PO ;
- l'affectation d'une story à un sprint ;
- la récupération des stories non planifiées pour le Sprint Planning.

Ce service garantit la cohérence du backlog et son organisation selon les principes SCRUM.

### E.5.4 SprintService

Le **SprintService** contient la logique principale du Sprint Backlog :

- création d'un sprint (dates, capacité, objectif) ;
- sélection des stories à partir du Product Backlog ;
- démarrage du sprint (changement d'état + verrouillage du backlog) ;
- clôture du sprint avec archivage de l'historique.

Il assure le respect des règles SCRUM concernant l'engagement, la préparation et la fin d'un sprint.

### E.5.5 TaskService

Ce service manipule les tâches techniques du Sprint Backlog.

Fonctionnalités principales :

- création et mise à jour des tâches ;
- changement d'état Kanban : `To Do` → `In Progress` → `Review` → `Done` ;
- assignation à un développeur ;
- calcul du pourcentage d'avancement du sprint.

Il communique étroitement avec le **SprintService** pour maintenir la cohérence du sprint.

### E.5.6 BlockerService

Le **BlockerService** gère les **impédiments**, un élément essentiel du rôle du Scrum Master.

Il permet :

- la déclaration d'un blocage ;
- l'affectation au Scrum Master ;
- le suivi de la résolution ;
- la mise à jour du statut (ouvert, en cours, résolu).

Ces informations alimentent le **Dashboard Scrum Master**.

### E.5.7 MeetingService

Ce service encadre les événements SCRUM :

- Daily Scrum : enregistrement des points bloquants ;
- Sprint Review : résumé de l'incrément et feedback des parties prenantes ;
- Sprint Retrospective : actions d'amélioration continue.

Il structure les données nécessaires à la traçabilité des réunions.

### E.5.8 NotificationService

Le **NotificationService** gère les notifications importantes du système :

- notification de blocage ;
- début ou fin d'un sprint ;
- assignation d'une tâche ;
- mise à jour du backlog.

Les notifications peuvent être envoyées :

- par email,
- via WebSocket,
- ou à terme, par un processus BPMN automatisé.

## Synthèse

La couche service assure l'ensemble des règles métier du système. Elle se place au cœur de l'architecture Spring Boot et garantit :

- une application cohérente avec les principes SCRUM ;
- un code modulaire et évolutif ;
- une facilité de test et de maintenance ;
- une compatibilité naturelle avec l'intégration BPMN de Camunda.

## E.6 API REST

L'application expose une API REST structurée autour des modules métiers : authentification, gestion des utilisateurs, gestion des projets, backlog, sprints, tâches et blocages. Le frontend Angular communique exclusivement avec ces endpoints, en envoyant un jeton JWT dans l'en-tête **Authorization: Bearer <token>**.

Chaque ressource respecte les principes REST :

- **GET** : lecture,
- **POST** : création,
- **PUT** : modification,
- **DELETE** : suppression.

Les sections suivantes présentent les endpoints principaux de l'API et leur rôle fonctionnel.

### E.6.1 Authentification (JWT)

- **POST /auth/login** Authentifie un utilisateur et renvoie un *access token* et un *refresh token*.
- **POST /auth/refresh** Permet d'obtenir un nouveau token à partir du refresh token.
- **GET /auth/me** Retourne les informations de l'utilisateur connecté.

Ces endpoints sont consommés par le module Angular **auth** et par les gardes de routes (route guards).



## E.6.2 Gestion des utilisateurs (Admin)

- POST /users : création d'un utilisateur ;
- GET /users : liste des utilisateurs ;
- GET /users/{id} : récupération d'un utilisateur ;
- PUT /users/{id} : modification ;
- DELETE /users/{id} : suppression ;
- PUT /users/{id}/roles : affectation de rôles (ADMIN, PO, SM, DEV).

Ces opérations sont disponibles uniquement pour les administrateurs.

## E.6.3 Gestion des projets

- POST /projects : création d'un projet ;
- GET /projects : liste des projets de l'utilisateur ;
- GET /projects/{id} : détails d'un projet ;
- PUT /projects/{id} : mise à jour ;
- PUT /projects/{id}/team : ajout / suppression de membres ;
- DELETE /projects/{id} : suppression.

Un Product Owner peut créer et gérer ses projets ; un Scrum Master peut en consulter la structure.

## E.6.4 Product Backlog (Stories)

- POST /projects/{id}/stories : ajout d'un élément au Product Backlog ;
- GET /projects/{id}/stories : liste des stories ;
- PUT /stories/{id} : modification d'une story ;
- DELETE /stories/{id} : suppression ;
- PUT /stories/{id}/priority : mise à jour de la priorité ;
- PUT /stories/{id}/assignToSprint/{sprintId} : affectation d'une story à un sprint.

Ces endpoints implémentent l'organisation complète du Product Backlog selon SCRUM.

## E.6.5 Sprint Backlog et gestion des sprints

- POST /projects/{id}/sprints : création d'un sprint ;
- GET /projects/{id}/sprints : liste des sprints du projet ;
- GET /sprints/{id} : détails d'un sprint ;
- PUT /sprints/{id}/start : démarrage du sprint ;
- PUT /sprints/{id}/close : clôture ;

Ces endpoints sont principalement utilisés par le Scrum Master.

## E.6.6 Tâches techniques (Task Board)

- POST /sprints/{id}/tasks : création d'une nouvelle tâche ;
- GET /sprints/{id}/tasks : liste du Sprint Backlog ;
- PUT /tasks/{id} : mise à jour d'une tâche ;
- PUT /tasks/{id}/status : changement de colonne Kanban ;
- PUT /tasks/{id}/assign/{userId} : assignation à un développeur.

Cela permet de gérer le tableau Kanban du sprint (To Do / In Progress / Review / Done).

## E.6.7 Blocages (Impediments)

- POST /tasks/{taskId}/blockers : signalement d'un blocage ;
- GET /blockers : liste des blocages ouverts ;
- PUT /blockers/{id} : mise à jour du statut ;
- DELETE /blockers/{id} : suppression après résolution.

Les endpoints de blocage sont essentiels pour la visibilité du Scrum Master et pour l'automatisation future via BPMN.

## E.6.8 Métriques et tableaux de bord

- GET /analytics/project/{id} : statistiques du projet ;
- GET /analytics/sprint/{id} : métriques du sprint ;
- GET /analytics/blockers : indicateurs sur les blocages.

Ces endpoints alimentent le **Dashboard Angular** du PO et du SM.

## Synthèse

L'API REST constitue l'interface centrale entre le frontend Angular et la logique métier Spring Boot. Elle suit une organisation claire, conforme aux standards REST, et sécurisée par des tokens JWT. Cette API permet :

- une modularité forte,
- une maintenance facilitée,
- une future intégration BPMN (Camunda) via des endpoints dédiés,
- l'évolution vers une architecture microservices si nécessaire.

## E.6.9 Authentification (JWT)

- POST /auth/login Authentifie un utilisateur et renvoie un *access token* et un *refresh token*.
- POST /auth/refresh Permet d'obtenir un nouveau token à partir du refresh token.
- GET /auth/me Retourne les informations de l'utilisateur connecté.

Ces endpoints sont consommés par le module Angular **auth** et par les gardes de routes (route guards).

### E.6.10 Gestion des utilisateurs (Admin)

- POST /users : création d'un utilisateur ;
- GET /users : liste des utilisateurs ;
- GET /users/{id} : récupération d'un utilisateur ;
- PUT /users/{id} : modification ;
- DELETE /users/{id} : suppression ;
- PUT /users/{id}/roles : affectation de rôles (ADMIN, PO, SM, DEV).

Ces opérations sont disponibles uniquement pour les administrateurs.

### E.6.11 Gestion des projets

- POST /projects : création d'un projet ;
- GET /projects : liste des projets de l'utilisateur ;
- GET /projects/{id} : détails d'un projet ;
- PUT /projects/{id} : mise à jour ;
- PUT /projects/{id}/team : ajout / suppression de membres ;
- DELETE /projects/{id} : suppression.

Un Product Owner peut créer et gérer ses projets ; un Scrum Master peut en consulter la structure.

### E.6.12 Product Backlog (Stories)

- POST /projects/{id}/stories : ajout d'un élément au Product Backlog ;
- GET /projects/{id}/stories : liste des stories ;
- PUT /stories/{id} : modification d'une story ;
- DELETE /stories/{id} : suppression ;
- PUT /stories/{id}/priority : mise à jour de la priorité ;
- PUT /stories/{id}/assignToSprint/{sprintId} : affectation d'une story à un sprint.

Ces endpoints implémentent l'organisation complète du Product Backlog selon SCRUM.

### E.6.13 Sprint Backlog et gestion des sprints

- POST /projects/{id}/sprints : création d'un sprint ;
- GET /projects/{id}/sprints : liste des sprints du projet ;
- GET /sprints/{id} : détails d'un sprint ;
- PUT /sprints/{id}/start : démarrage du sprint ;
- PUT /sprints/{id}/close : clôture ;

Ces endpoints sont principalement utilisés par le Scrum Master.

### E.6.14 Tâches techniques (Task Board)

- POST /sprints/{id}/tasks : création d'une nouvelle tâche ;
- GET /sprints/{id}/tasks : liste du Sprint Backlog ;
- PUT /tasks/{id} : mise à jour d'une tâche ;
- PUT /tasks/{id}/status : changement de colonne Kanban ;
- PUT /tasks/{id}/assign/{userId} : assignation à un développeur.

Cela permet de gérer le tableau Kanban du sprint (To Do / In Progress / Review / Done).

### E.6.15 Blocages (Impediments)

- POST /tasks/{taskId}/blockers : signalement d'un blocage ;
- GET /blockers : liste des blocages ouverts ;
- PUT /blockers/{id} : mise à jour du statut ;
- DELETE /blockers/{id} : suppression après résolution.

Les endpoints de blocage sont essentiels pour la visibilité du Scrum Master et pour l'automatisation future via BPMN.

### E.6.16 Métriques et tableaux de bord

- GET /analytics/project/{id} : statistiques du projet ;
- GET /analytics/sprint/{id} : métriques du sprint ;
- GET /analytics/blockers : indicateurs sur les blocages.

Ces endpoints alimentent le **Dashboard Angular** du PO et du SM.

## Synthèse

L'API REST constitue l'interface centrale entre le frontend Angular et la logique métier Spring Boot. Elle suit une organisation claire, conforme aux standards REST, et sécurisée par des tokens JWT. Cette API permet :

- une modularité forte,
- une maintenance facilitée,
- une future intégration BPMN (Camunda) via des endpoints dédiés,
- l'évolution vers une architecture microservices si nécessaire.

## E.7 Sécurité et authentification JWT

La sécurité de l'application repose sur un mécanisme d'authentification moderne basé sur les **JSON Web Tokens (JWT)**. Ce système protège les endpoints REST du backend et garantit que seuls les utilisateurs authentifiés (Admin, PO, SM, Développeurs) peuvent accéder aux ressources.

Le choix de JWT permet :

- une authentification stateless (aucune session stockée côté serveur) ;
- une compatibilité parfaite avec Angular ;
- une gestion fine des rôles (RBAC) ;
- une intégration future simplifiée avec Camunda et Docker.

### E.7.1 Principe général

Le processus d'authentification se déroule en trois étapes :

- A. **Login** : l'utilisateur envoie son *username* et *password*.
- B. **Émission du token JWT** : le backend vérifie les informations, puis génère :
  - un **access token** (durée courte) ;
  - un **refresh token** (durée longue).
- C. **Appels sécurisés** : le frontend inclut l'access token dans chaque requête : `Authorization: Bearer <token>`

Si le token expire, le frontend utilise le `/auth/refresh` pour en obtenir un nouveau sans redemander les identifiants.

### E.7.2 Structure du token JWT

Le token se compose de trois parties :

- **Header** : type du token + algorithme (HS256)
- **Payload** : données utilisateur (id, username, rôles)
- **Signature** : garantit l'intégrité du token

Exemple d'informations stockées :

- **sub** : nom d'utilisateur ;
- **roles** : USER, ADMIN, PO, SM ;
- **exp** : date d'expiration.

### E.7.3 Filtres de sécurité

Spring Boot utilise deux filtres principaux :

- **AuthenticationFilter** Intercepte les requêtes de login, vérifie les identifiants et génère les tokens.
- **AuthorizationFilter** Intercepte toutes les requêtes, extrait le token et valide :
  - la signature du token ;
  - la date d'expiration ;
  - les rôles de l'utilisateur.

Si le token est invalide, expiré ou absent, la requête est automatiquement refusée.

#### E.7.4 Gestion des rôles et permissions (RBAC)

L'application repose sur un modèle RBAC (Role-Based Access Control).

Tous les utilisateurs possèdent le rôle de base :

→ **USER**

Puis viennent les rôles spécifiques :

- **ADMIN** : gestion complète du système ;
- **PO** : gestion du Product Backlog, projets ;
- **SM** : gestion des sprints, tâches, blocages ;
- **DEV** : réalisation des tâches techniques.

L'accès aux endpoints REST se définit dans **SecurityConfig**, par exemple :

- `/admin/**` → réservé à ADMIN ;
- `/projects/**` → réservé aux utilisateurs authentifiés ;
- `/sprints/**` → PO + SM ;
- `/blockers/**` → SM + DEV.

Cela garantit une séparation nette des responsabilités SCRUM.

#### E.7.5 Gestion des refresh tokens

Pour éviter les déconnexions intempestives, un refresh token permet d'obtenir un nouveau token lorsque l'ancien expire :

→ POST `/auth/refresh`

Le backend vérifie :

- la validité du refresh token ;
- l'utilisateur associé ;
- la non-expiration.

Un nouvel access token est alors renvoyé au frontend.

#### E.7.6 Sécurité des mots de passe

Les mots de passe sont :

- hashés avec **BCrypt** ;
- jamais stockés en clair ;
- validés via un **PasswordEncoder Spring**.

Ceci garantit une protection solide contre les attaques de type brute force.

## E.7.7 Avantages de JWT dans une architecture Angular / Spring Boot

Ce choix technologique offre plusieurs atouts :

- pas de sessions côté serveur (scalable) ;
- communication simple via headers HTTP ;
- compatible mobile / SPA / microservices ;
- isolation parfaite entre frontend et backend ;
- compatible avec Docker et déploiement distribué.

### Synthèse

La sécurité JWT assure une authentification robuste, flexible et adaptée aux architectures modernes. Dans cette application, elle garantit :

- la protection des endpoints REST,
- une gestion propre des rôles SCRUM,
- une expérience utilisateur fluide (grâce au refresh token),
- une base solide pour l'intégration BPMN et les futures évolutions.

## E.8 Pré-intégration du moteur BPMN Camunda

Bien que l'intégration complète de Camunda 7 ne soit pas finalisée au moment de la rédaction de ce rapport, l'architecture du backend Spring Boot a été conçue de manière à accueillir le moteur BPMN sans modification majeure.

Cette section décrit les éléments déjà en place, ainsi que les mécanismes préparatoires pour l'automatisation future des processus SCRUM.

### E.8.1 Motivation : pourquoi intégrer Camunda ?

Les processus SCRUM comportent de nombreuses étapes répétitives ou dépendantes d'événements. L'utilisation de **BPMN 2.0** et du moteur **Camunda 7** permet :

- d'automatiser certaines actions (notifications, transitions d'état, validations) ;
- de modéliser les workflows SCRUM de manière graphique et standardisée ;
- d'améliorer la traçabilité et l'exécution des processus ;
- de réduire les erreurs humaines et les oubliés (blocages non signalés, tâches non clôturées, etc.).

L'objectif final est de permettre au Scrum Master d'accéder à un module dédié dans l'interface Angular pour créer, déployer et exécuter des processus BPMN.

### E.8.2 Structure prévue pour l'intégration

Le backend comprend déjà un module `camunda` destiné à accueillir :

- le moteur BPMN embarqué dans Spring Boot ;
- le déploiement automatique des fichiers .bpnn lors du démarrage ;
- la configuration du *ProcessEngine* ;
- les classes Java déléguées (*Delegates*) à exécuter depuis un workflow ;
- un point d'entrée REST pour lancer ou interagir avec un processus.

Cette infrastructure garantit une intégration progressive, sans refonte du backend.

### E.8.3 Processus BPMN prévus

Plusieurs processus BPMN ont été identifiés et modélisés. Ils correspondent à des étapes clés du fonctionnement SCRUM :

- **Processus de création d'un projet** (validation par PO, enregistrement automatique, notifications).
- **Processus de planification d'un sprint** (sélection automatique des stories, validation des capacités, démarrage du sprint).
- **Processus de gestion des blocages** (signalement → analyse SM → escalade → résolution).
- **Processus d'ajout d'un élément au Product Backlog** (critères d'acceptation → estimation → insertion dans backlog).

Ces workflows BPMN ont été conçus dans **Camunda Modeler** et sont prêts à être intégrés dans le moteur embarqué.

### E.8.4 Déploiement automatique des fichiers BPMN

Le backend contient déjà une structure prévue pour :

- déposer des fichiers .bpnn dans un dossier dédié ;
- charger automatiquement ces processus à l'exécution du backend ;
- les rendre disponibles dans Camunda Cockpit ;
- permettre l'exécution via un appel REST.

Cela permettra au Scrum Master de travailler en autonomie sur ses modèles.

### E.8.5 Intégration future dans Angular

Une interface dédiée est prévue pour le rôle **Scrum Master**, permettant :

- de consulter les processus BPMN disponibles ;
- d'initialiser ou suivre l'exécution d'un processus ;
- d'importer un nouveau fichier BPMN (glisser-déposer) ;
- d'obtenir une visualisation graphique (Camunda Viewer).

Cette fonctionnalité transforme l'application en un véritable outil de gestion de processus métier SCRUM.

### E.8.6 Points techniques déjà implémentés

Le backend inclut déjà :

- une structure de package dédiée à Camunda ;
- la configuration Spring pour activer le moteur BPMN ;
- des endpoints REST prévus pour interagir avec Camunda ;
- un espace pour les classes *Java Delegates* ;
- des services structurés de manière compatible avec une orchestration BPMN ;
- la gestion des événements SCRUM qui serviront de points d'entrée (signalement d'un blocage, démarrage sprint...).

Ainsi, l'intégration de Camunda ne nécessite plus qu'une connexion technique et le déploiement des modèles BPMN.

### Synthèse

L'intégration complète du moteur Camunda est prévue dans la continuité du projet. Les fondations sont déjà en place grâce :

- à une architecture backend modulaire,
- à une séparation claire entre logique métier et orchestration,
- à l'utilisation de JPA et JWT,
- à une API REST déjà adaptée aux workflows.

Le système est donc prêt à évoluer vers une automatisation avancée des processus SCRUM.

### E.9 Conclusion

Le backend Spring Boot constitue une base solide pour la digitalisation complète des pratiques SCRUM. Grâce à son architecture modulaire, à la sécurité JWT, aux services métier clairement définis et à la préparation de l'intégration BPMN, le système est à la fois performant, maintenable et extensible.

Le chapitre suivant présente l'implémentation du frontend Angular.



## CHAPITRE F

# Implémentation du Frontend Angular

## F.1 Introduction

Le frontend de l'application a été développé avec le framework **Angular**. Le choix d'Angular s'explique par plusieurs avantages :

- une architecture modulaire adaptée aux applications complexes ;
- une gestion native du routing, des services et de l'injection de dépendances ;
- une forte intégration avec TypeScript, garantissant robustesse et maintenabilité ;
- une excellente compatibilité avec les API REST exposées par Spring Boot.

L'application est conçue comme une **Single Page Application (SPA)**, ce qui permet une navigation fluide, sans rechargement complet de page. Le frontend constitue l'interface principale des rôles SCRUM : Product Owner, Scrum Master, Développeur et Administrateur.

## F.2 Architecture générale du frontend

L'architecture Angular repose sur une organisation en modules fonctionnels, chacun chargé d'une partie du domaine SCRUM. Cette structuration améliore la lisibilité du code et facilite l'évolution future.

- **auth** : authentification JWT, gestion des rôles ;
- **projects** : création, édition et sélection de projets ;
- **backlog** : gestion du Product Backlog (stories) ;
- **sprints** : gestion du Sprint Backlog, tâches, tableau Kanban ;
- **tasks** : suivi et mise à jour des tâches par les développeurs ;
- **meetings** : module prévu pour les cérémonies SCRUM ;
- **dashboard** : métriques PO/SM (burndown, vitesse, blocages) ;
- **admin** : gestion des utilisateurs et des rôles.

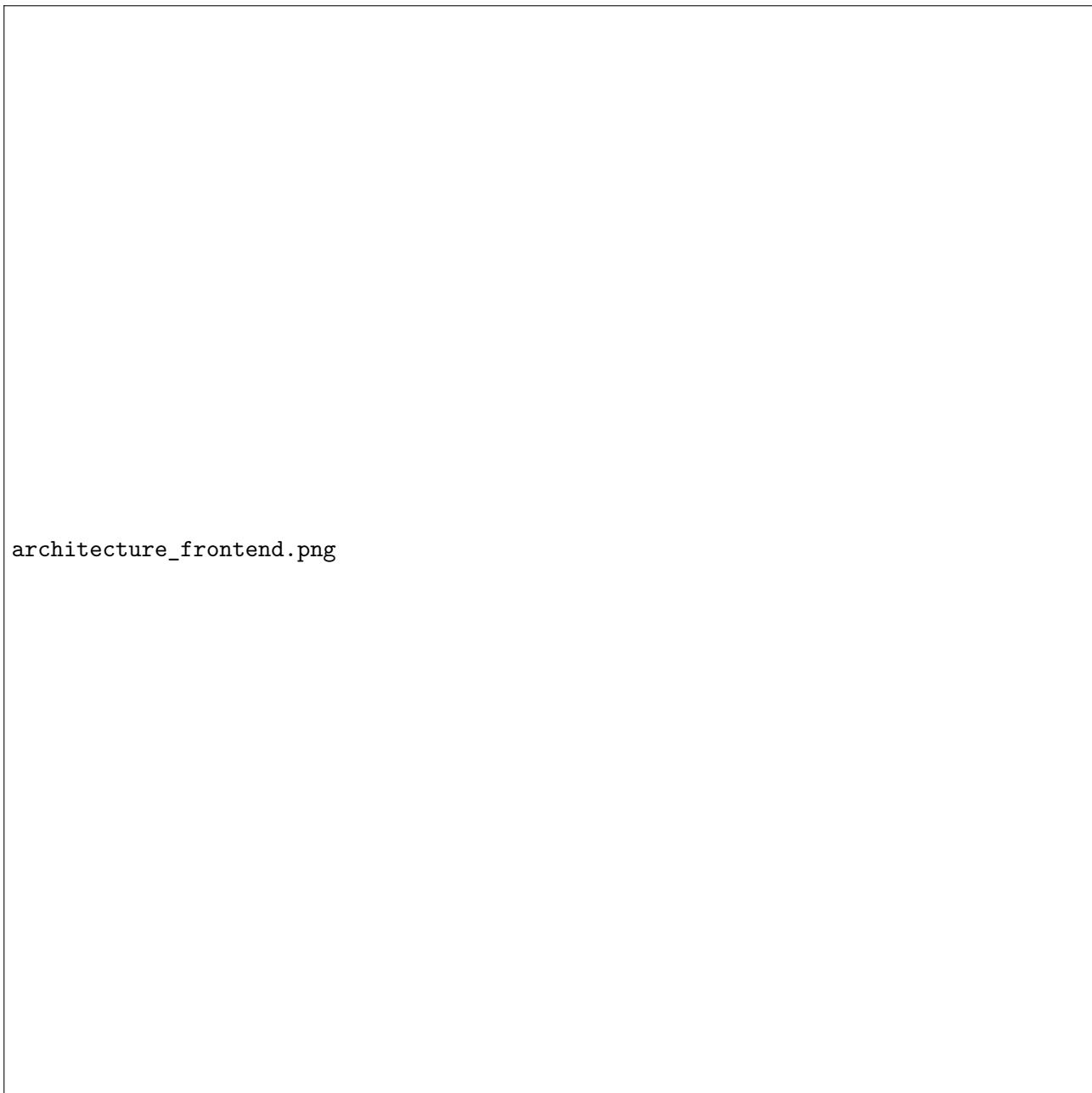


FIGURE F.2.1 – Structure générale du frontend Angular

### F.3 Gestion de l'authentification JWT

L'authentification repose sur un **AuthService** qui communique avec le backend via les endpoints `/auth/login` et `/auth/refresh`. Lorsqu'un utilisateur se connecte :

- A. il envoie son *username* et *password* ;
- B. le backend renvoie un *access token* et un *refresh token* ;
- C. l'*access token* est stocké dans `localStorage` ;
- D. toutes les requêtes HTTP suivantes incluent automatiquement le token.

Un **HTTP Interceptor** ajoute automatiquement le header :

```
Authorization: Bearer <token>
```

Deux gardes (guards) assurent la sécurité du routing :

- **AuthGuard** : accès réservé aux utilisateurs connectés ;
- **RoleGuard** : restriction selon rôle (Admin, PO, SM, Dev).

Cela garantit une isolation nette des fonctionnalités selon les responsabilités SCRUM.

## F.4 Modules fonctionnels du frontend

### F.4.1 Module Projets (Product Owner)

Le Product Owner peut :

- créer un projet ;
- définir le Scrum Master et les membres du développement ;
- consulter les informations générales du projet ;
- accéder au backlog, aux sprints et aux métriques associées.

Les pages principales incluent :

- création d'un projet ;
- liste des projets ;
- tableau d'administration de l'équipe projet.

### F.4.2 Module Backlog (PO)

Le PO gère entièrement le **Product Backlog** :

- création, modification et suppression de stories ;
- gestion des priorités via drag-and-drop ;
- définition des critères d'acceptation ;
- estimation des story points.

Une interface claire permet au PO de maintenir un backlog toujours à jour.

### F.4.3 Module Sprints (Scrum Master)

Le Scrum Master dispose d'un module complet pour :

- créer un sprint ;
- définir sa capacité ;
- sélectionner les stories du backlog ;
- gérer le Sprint Board (*To Do* → *In Progress* → *Review* → *Done*) ;
- clôturer le sprint.

Le tableau Kanban est mis à jour en temps réel par les développeurs.

#### F.4.4 Module Tasks (Développeurs)

Les développeurs peuvent :

- consulter leurs tâches assignées ;
- changer le statut (drag-and-drop) ;
- ajouter un temps passé ;
- signaler un blocage.

Chaque mise à jour est envoyée au backend via `TaskService`.

#### F.4.5 Module Meetings (SCRUM)

Le module Meetings était prévu pour gérer :

- Daily Scrum ;
- Sprint Review ;
- Sprint Retrospective.

Bien qu'il n'ait pas pu être implémenté à temps, l'architecture est prête :

- un dossier Angular dédié (`/meetings`) existe ;
- les entités `Meeting` sont présentes côté backend ;
- la future intégration Camunda permettra d'automatiser le suivi des réunions.

Ce module sera ajouté dans une version future de l'application.

#### F.4.6 Module Dashboard (PO & SM)

Ce module affiche des métriques générées par le backend :

- burndown chart ;
- vélocité ;
- taux de complétion ;
- nombre de blocages ;
- indicateurs de santé du backlog.

Des graphiques `Chart.js` permettent une représentation claire des données.

#### F.4.7 Module Admin

L'administrateur peut :

- créer des utilisateurs ;
- modifier leurs rôles ;
- activer ou désactiver des comptes.

Cette interface simplifie la gestion des rôles SCRUM.

## F.5 Services Angular

Chaque module Angular repose sur un service dédié communiquant avec le backend :

- **AuthService** : login, refresh token, gestion du profil ;
- **ProjectService** : création et affichage des projets ;
- **StoryService** : gestion du Product Backlog ;
- **SprintService** : gestion des sprints ;
- **TaskService** : mise à jour des tâches ;
- **BlockerService** : signalement et résolution des blocages ;
- **AnalyticsService** : récupération des métriques ;

Chaque service utilise `HttpClient` et gère ses propres endpoints REST.

## F.6 Interfaces graphiques (UI/UX)

Cette section présente les principales interfaces développées dans l'application Angular. Pour chaque module fonctionnel (Authentification, Projets, Backlog, Sprints, Tâches, Blocages, Dashboard, Administration), une capture d'écran représentative est fournie.

Toutes les figures sont placées dans le dossier `Images/` du rapport.

### F.6.1 Page d'authentification

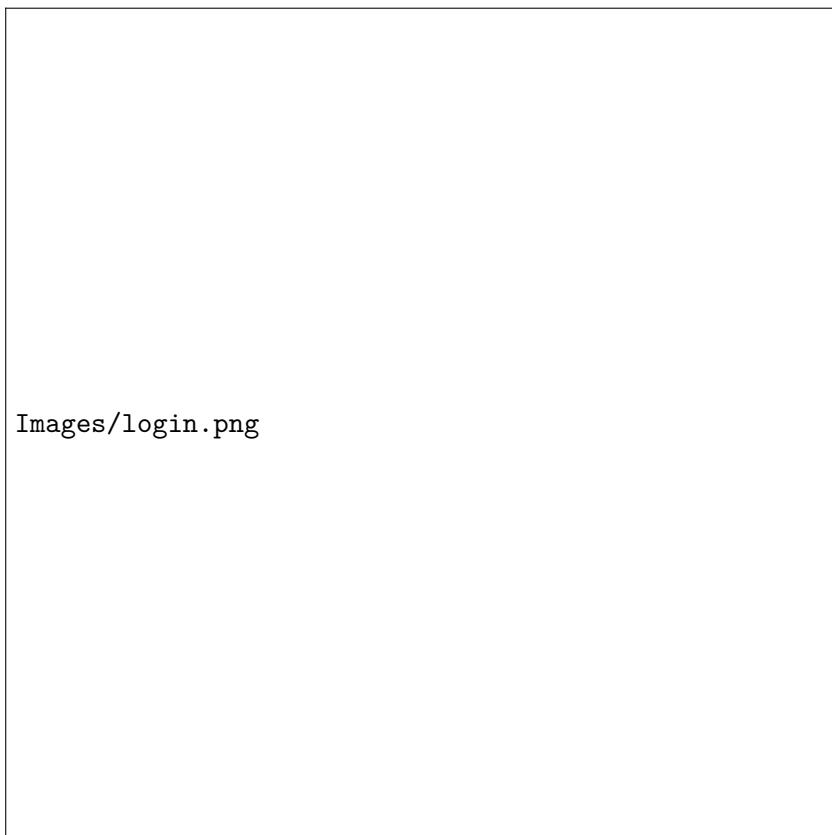


FIGURE F.6.1 – Interface de connexion de l'application

## F.6.2 Gestion des projets (PO)

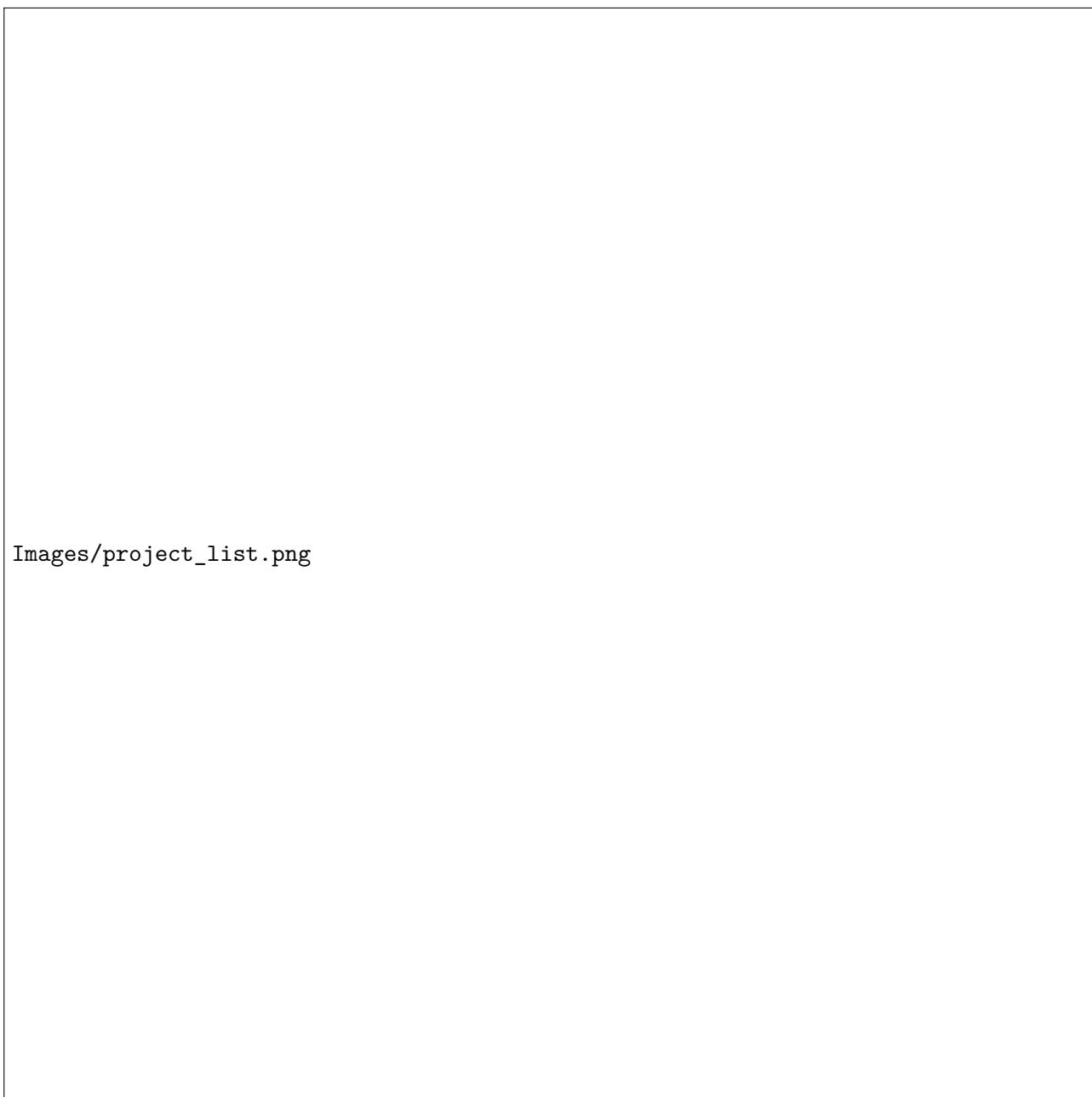
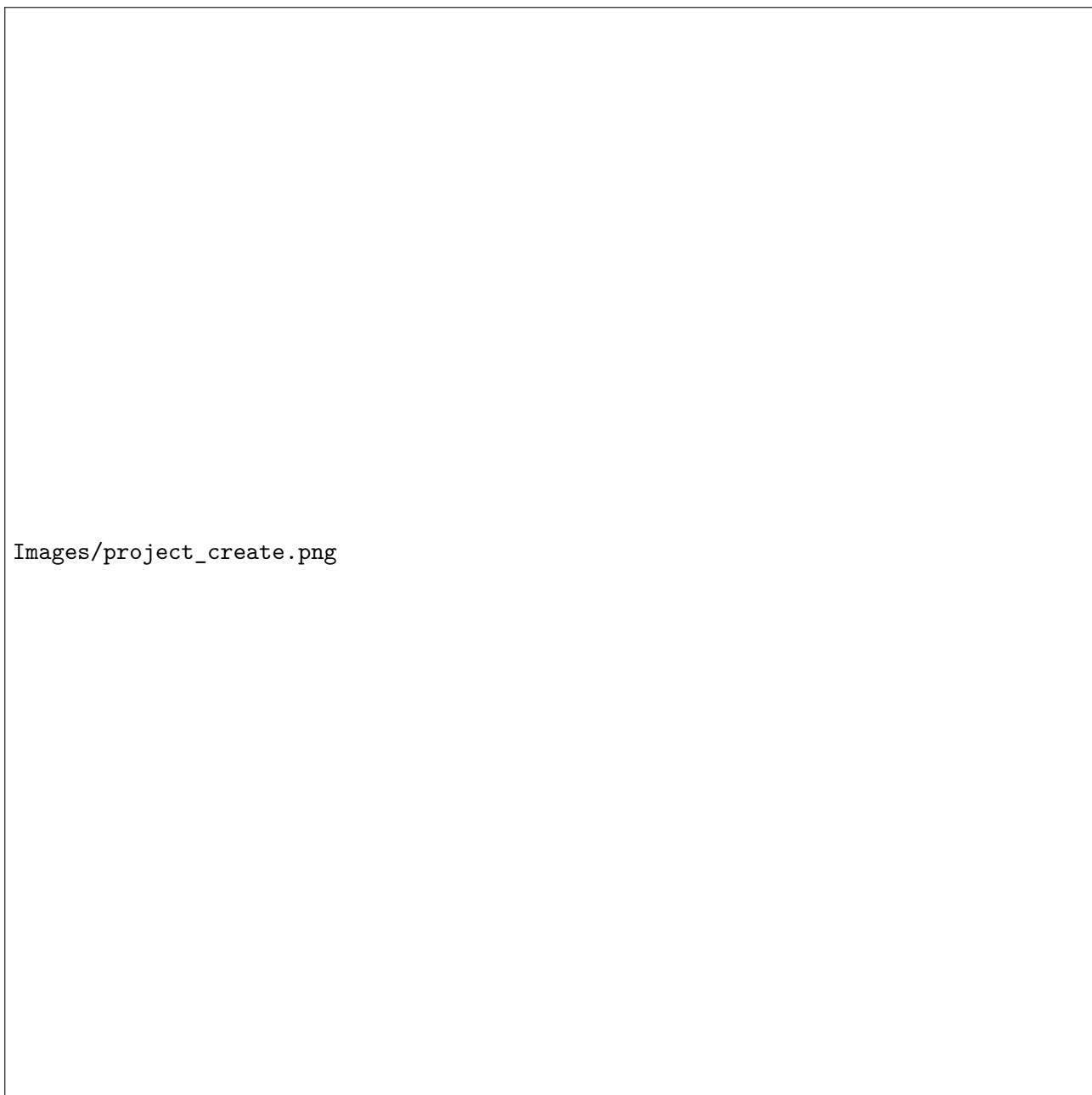
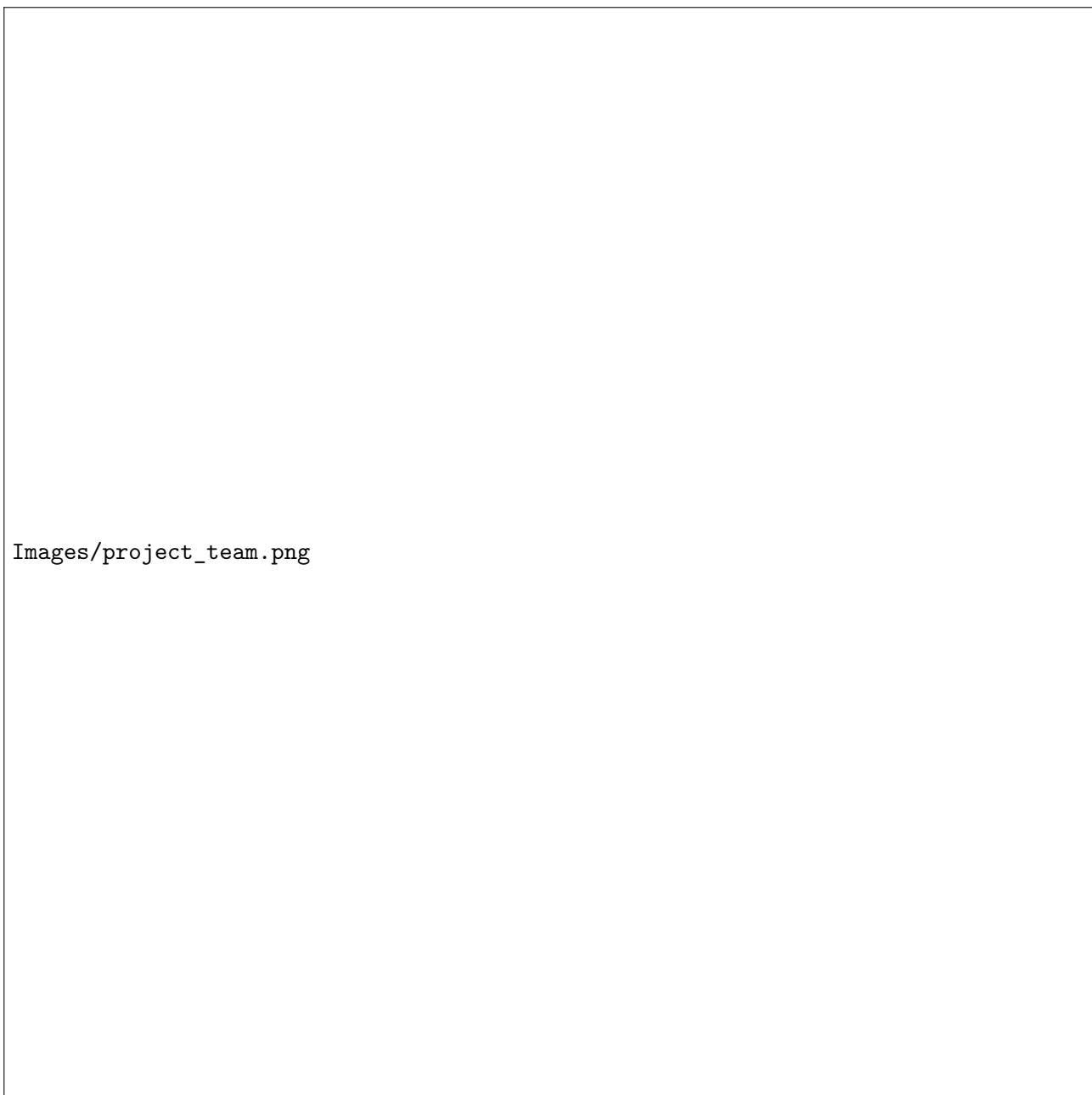


FIGURE F.6.2 – Liste des projets du Product Owner



Images/project\_create.png

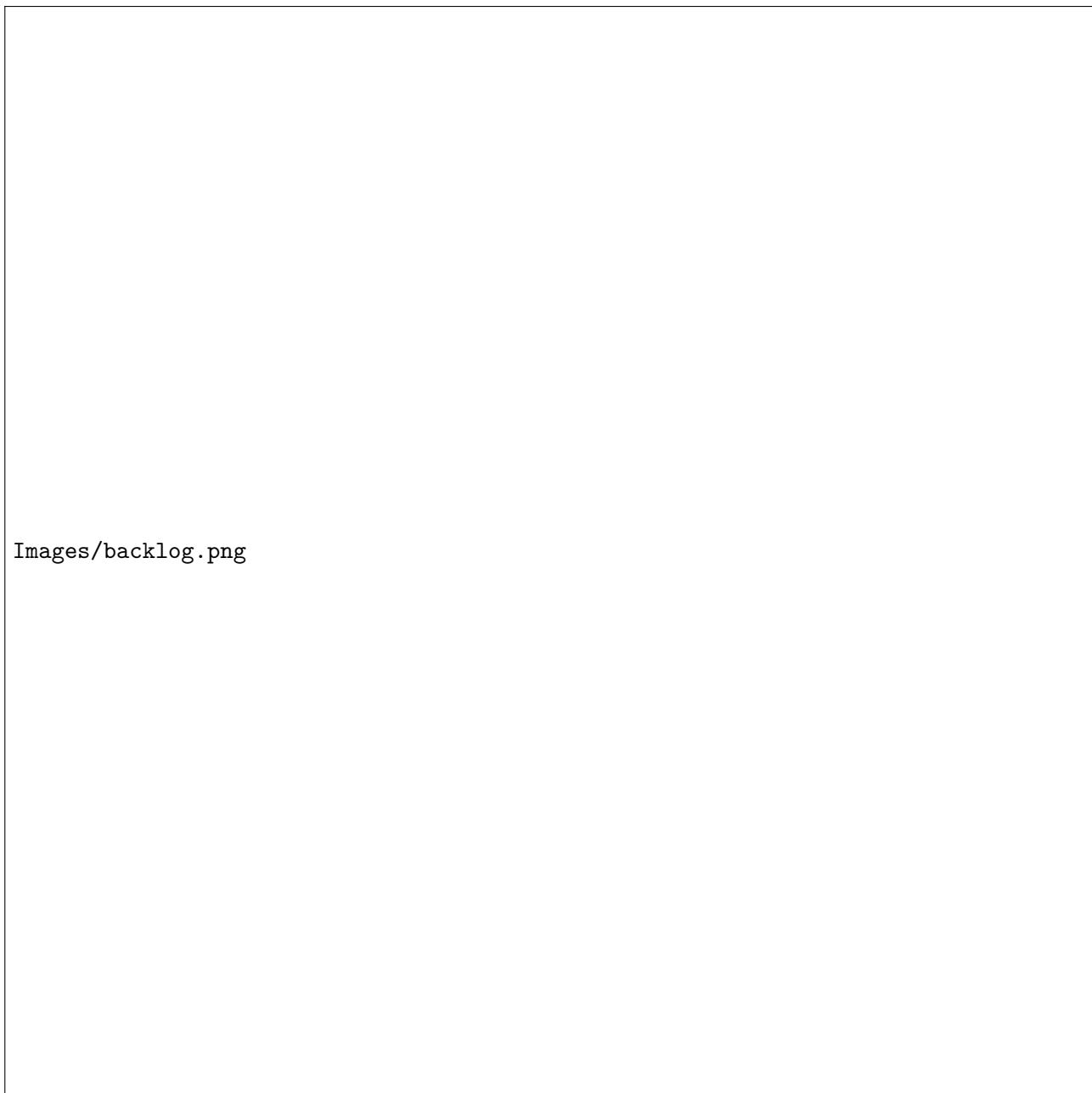
FIGURE F.6.3 – Formulaire de création d'un projet



Images/project\_team.png

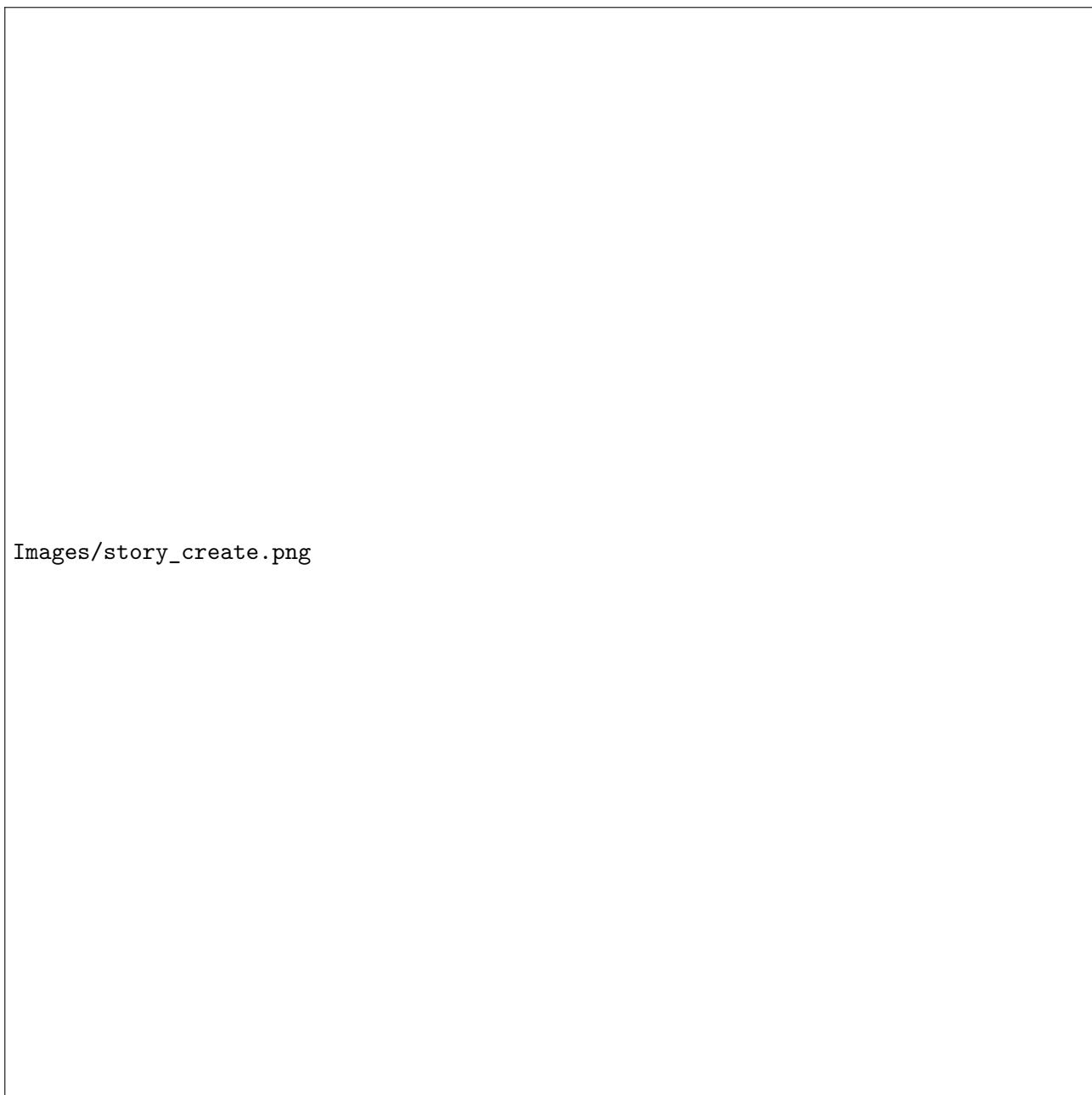
FIGURE F.6.4 – Affectation du Scrum Master et des développeurs

### F.6.3 Gestion du Product Backlog



Images/backlog.png

FIGURE F.6.5 – Vue du Product Backlog



Images/story\_create.png

FIGURE F.6.6 – Crédit d'une User Story

#### F.6.4 Module Sprints (Scrum Master)



Images/sprint\_list.png

FIGURE F.6.7 – Liste des sprints du projet

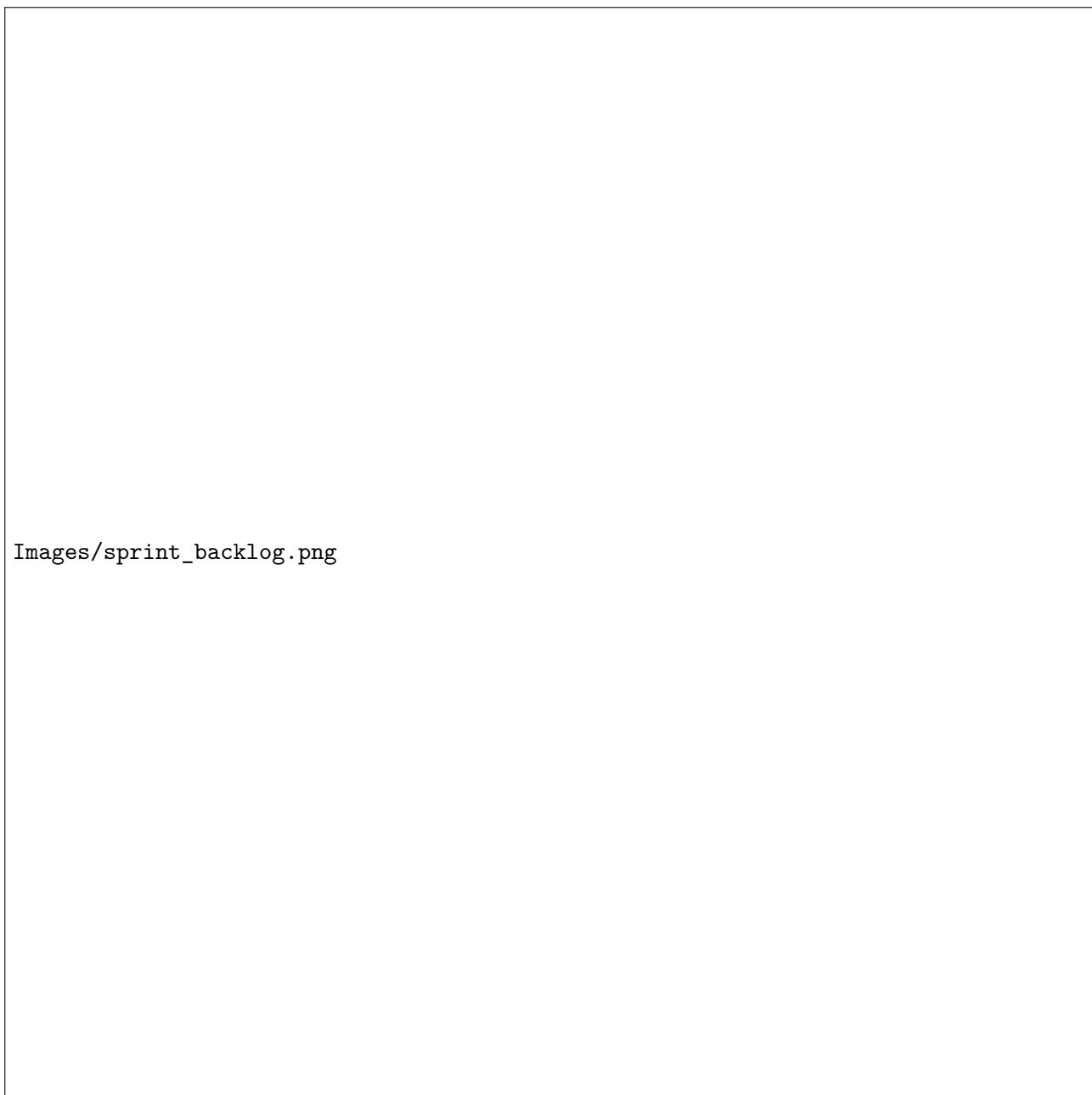
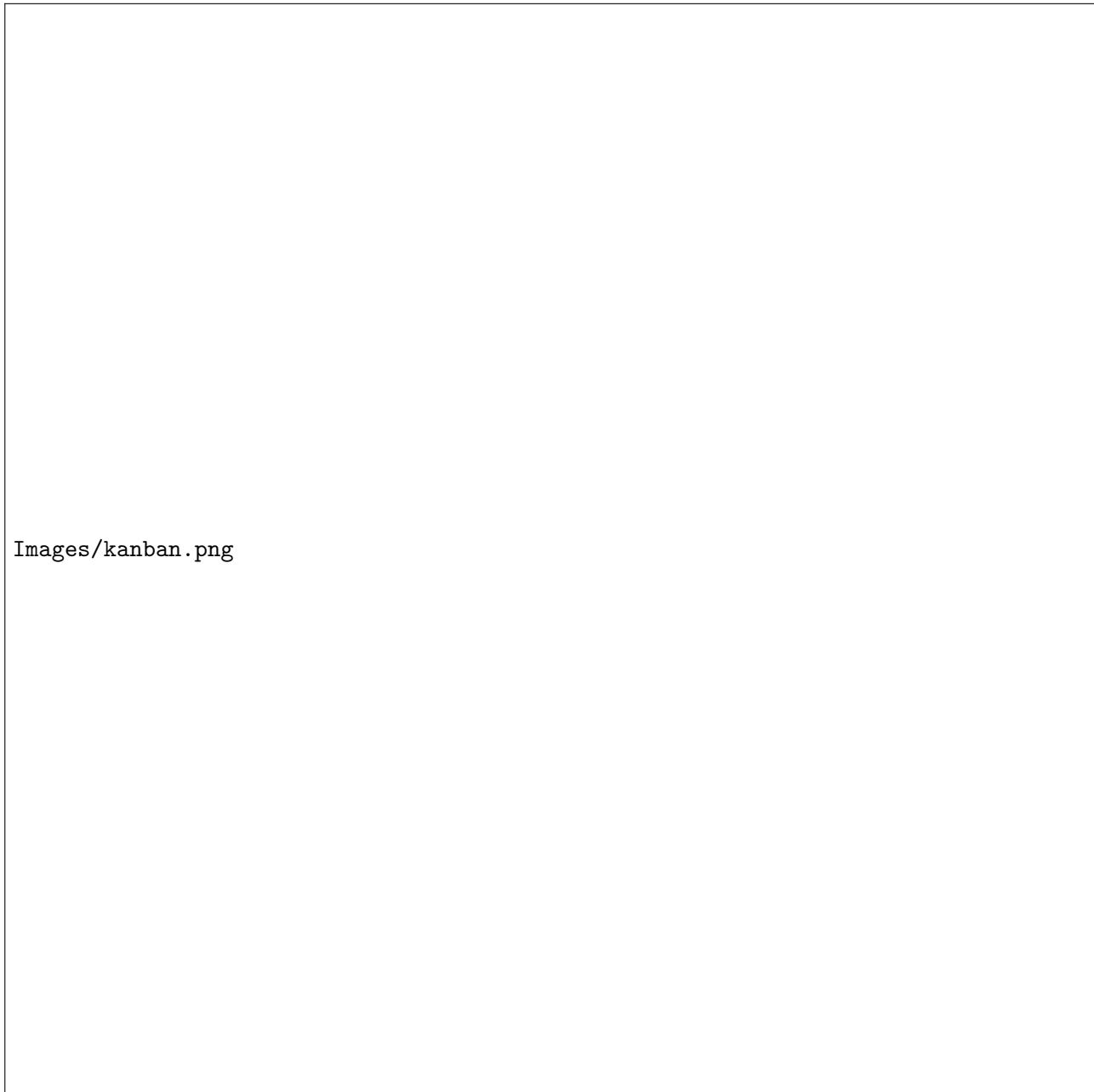


FIGURE F.6.8 – Sprint Backlog — sélection des stories et tâches

### F.6.5 Tableau Kanban (Développeurs)



Images/kanban.png

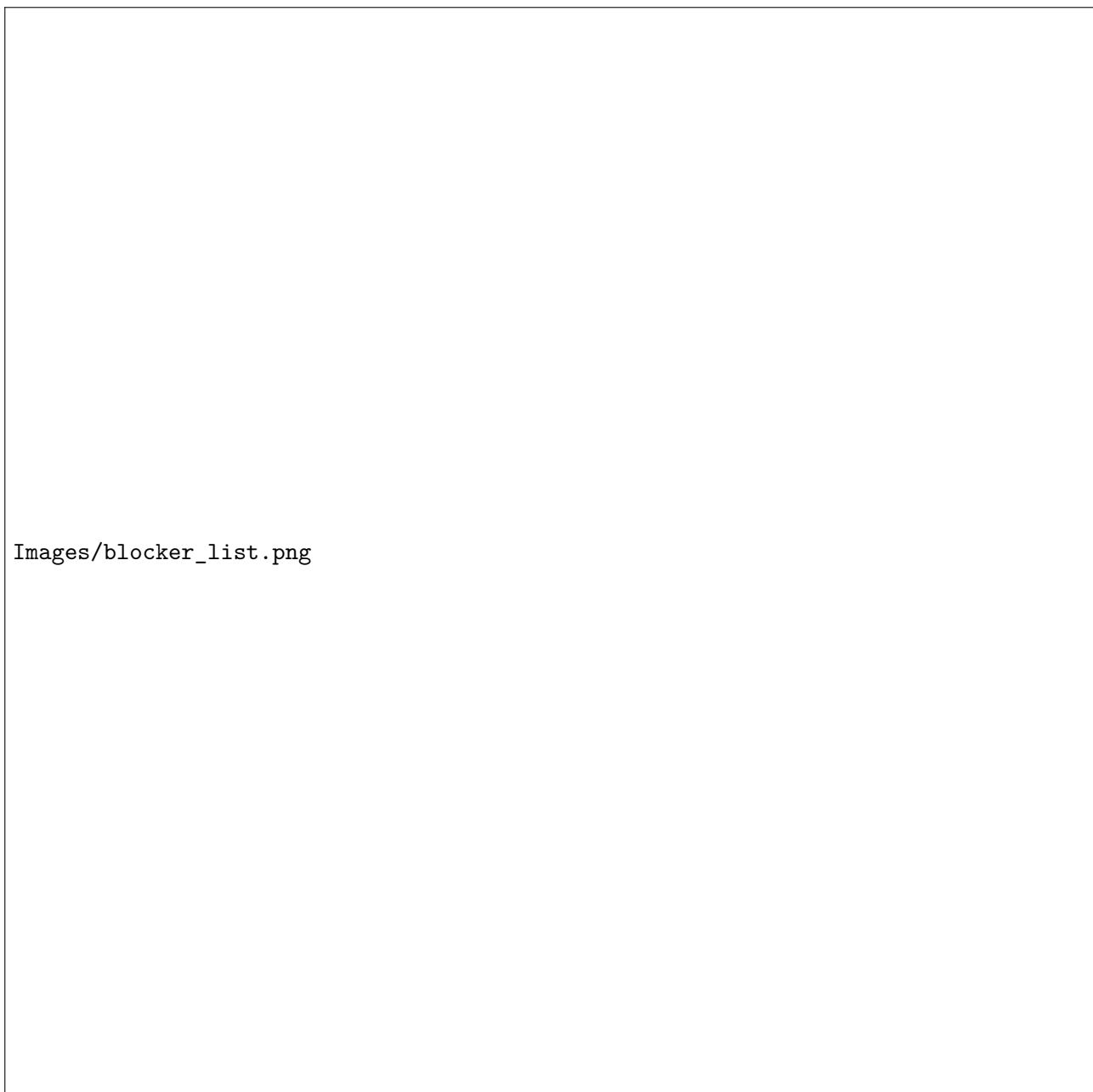
FIGURE F.6.9 – Tableau Kanban du sprint



## F.6.6 Blocages (Impediments)



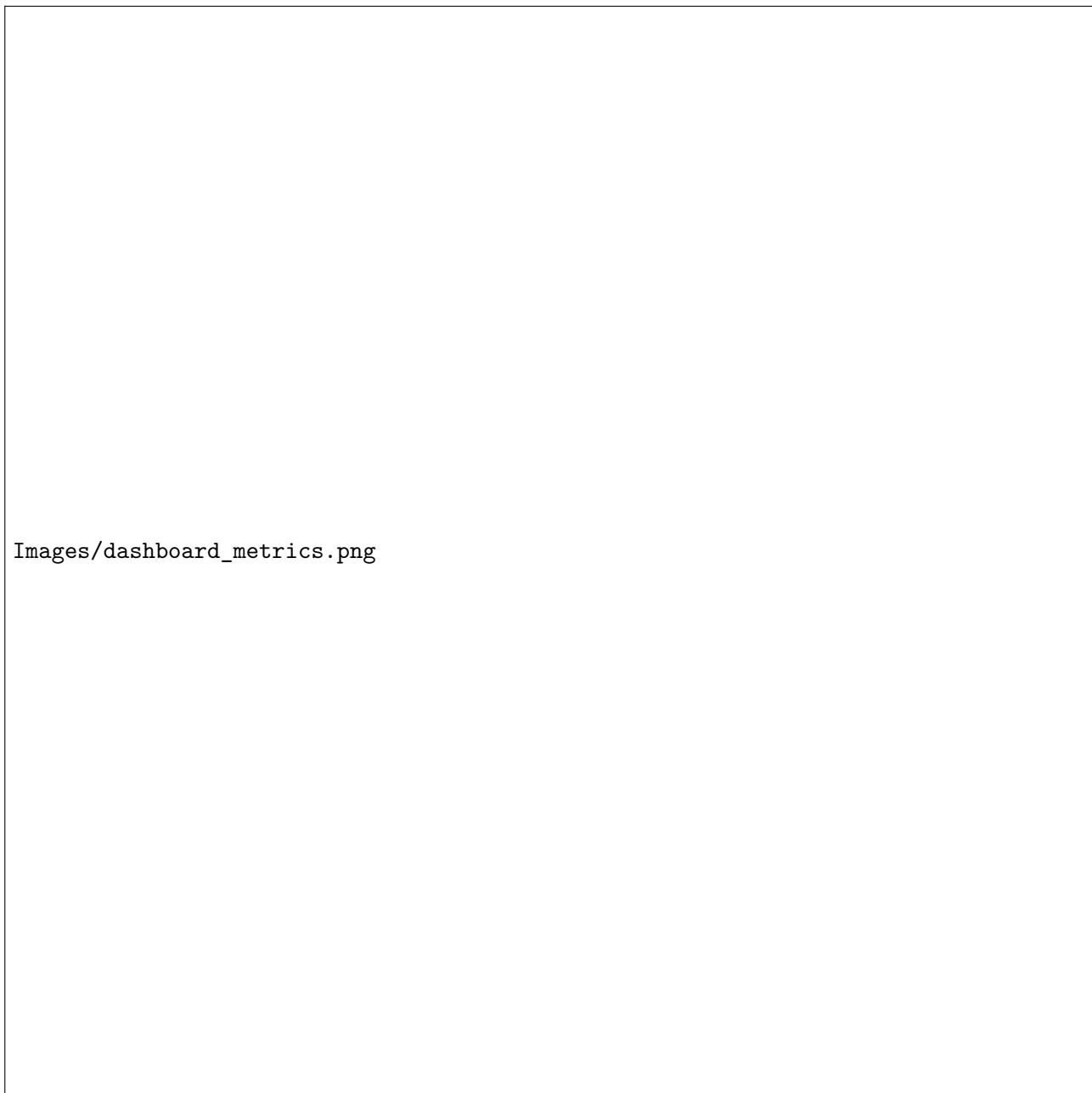
FIGURE F.6.10 – Signalement d'un blocage



Images/blocker\_list.png

FIGURE F.6.11 – Vue des blocages (Scrum Master)

### F.6.7 Dashboard (PO & SM)



Images/dashboard\_metrics.png

FIGURE F.6.12 – Métriques du projet : burndown, vélocité, blocages



#### F.6.8 Module Administration

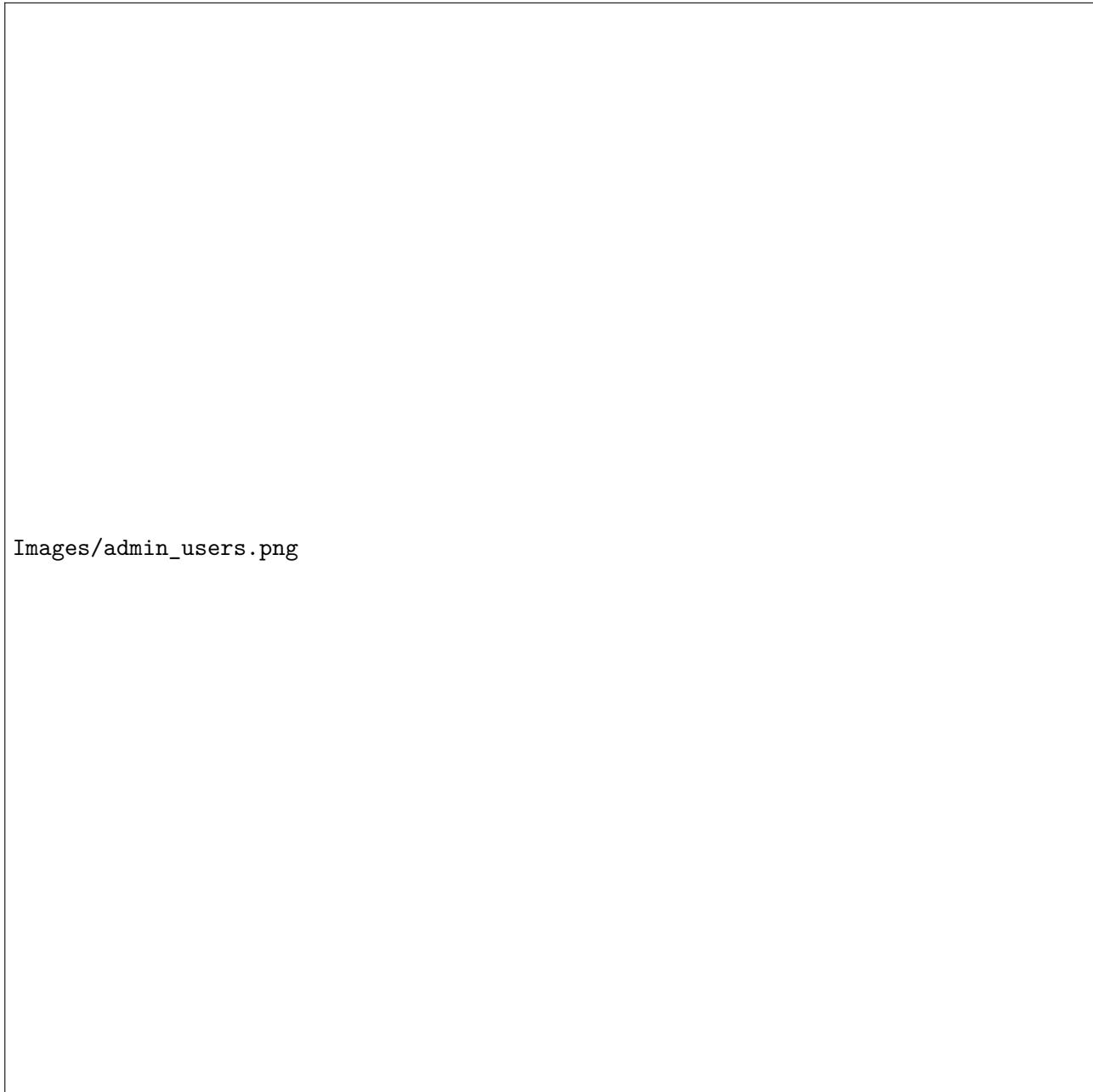


FIGURE F.6.13 – Gestion des utilisateurs

### F.7 Conclusion

Le frontend Angular constitue la partie visible de la plateforme de gestion SCRUM. Grâce à une architecture modulaire, une séparation claire des responsabilités et une intégration fluide avec le backend Spring Boot, il offre une interface professionnelle adaptable aux besoins des différents acteurs SCRUM.

Ce frontend est prêt à accueillir des extensions futures, notamment l'intégration du moteur BPMN Camunda pour automatiser une partie des processus SCRUM.



## CHAPITRE G

# Déploiement et Intégration

## G.1 Introduction

Cette section présente le déploiement de la plateforme SCRUM ainsi que l'intégration des différents composants techniques : frontend Angular, backend Spring Boot, base de données PostgreSQL et moteur BPMN Camunda.

Le choix de Docker garantit une installation reproductible, un environnement standardisé et une isolation complète des services. L'objectif est d'obtenir une architecture stable, facilement déployable et adaptée à l'évolution future du projet.

## G.2 Architecture technique déployée

La plateforme repose sur une architecture en conteneurs Docker, composée des services suivants :

- **Frontend Angular** déployé dans un conteneur Nginx ;
- **Backend Spring Boot** exposé en API REST ;
- **PostgreSQL 15** pour la persistance des données ;
- **Camunda 7** pour l'exécution des processus BPMN ;
- **Reverse Proxy Nginx** (optionnel) en vue d'un déploiement web.

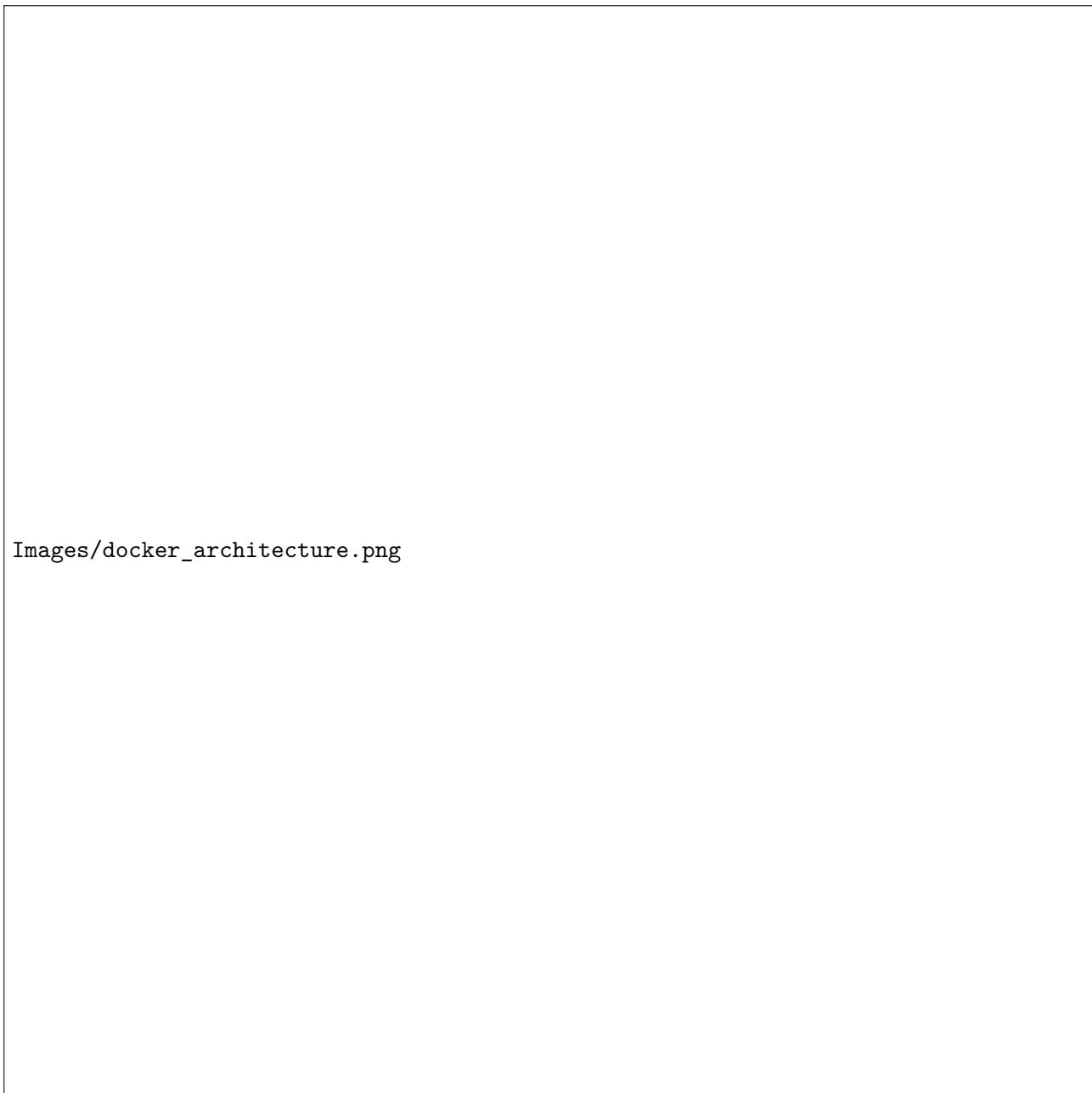


FIGURE G.2.1 – Architecture technique globale de la plateforme (Docker)

Chaque service est isolé dans son propre conteneur, mais l'ensemble communique via un réseau Docker interne.

Cette architecture offre plusieurs avantages :

- portabilité (le projet fonctionne sur tout OS disposant de Docker) ;
- reproductibilité (même configuration pour tous les environnements) ;
- sécurité (isolation des services) ;
- scalabilité (possibilité future de migration vers Kubernetes).

## G.3 Configuration Docker

Le déploiement complet repose sur un fichier `docker-compose.yml` réunissant backend, frontend et base de données.

---

```

1  version: '3.8'
2
3  services:
4
5    backend:
6      build: ./backend
7      container_name: scrum-backend
8      ports:
9        - "8080:8080"
10     depends_on:
11       - postgres
12
13   frontend:
14     build: ./frontend
15     container_name: scrum-frontend
16     ports:
17       - "4200:80"
18
19   postgres:
20     image: postgres:15
21     container_name: scrum-postgres
22     environment:
23       POSTGRES_DB: scrumdb
24       POSTGRES_USER: admin
25       POSTGRES_PASSWORD: admin
26     ports:
27       - "5432:5432"

```

---

Listing G.3.1 – Configuration du déploiement via Docker Compose

Les images Angular et Spring Boot sont générées automatiquement à partir des Dockerfile présents dans chaque module.

## G.4 Intégration du backend Spring Boot

Le backend expose une API REST sur le port 8080. Chaque conteneur est configuré pour communiquer avec la base PostgreSQL via l'URL interne Docker :

`jdbc:postgresql://postgres:5432/scrumdb`

Cette approche garantit que :

- aucune configuration locale n'est nécessaire ;
- l'environnement est identique en développement et en production ;
- les microservices peuvent être ajoutés ultérieurement sans modification majeure.

## G.5 Déploiement du frontend Angular

Le frontend Angular est compilé en production puis servi par Nginx. Un Dockerfile minimal permet d'obtenir un conteneur léger et performant :

---

```
1 FROM nginx:alpine
2 COPY dist/frontend /usr/share/nginx/html
```

---

Listing G.5.1 – Dockerfile du frontend Angular

Le frontend communique ensuite avec le backend via l'API REST sécurisée par JWT.

## G.6 Intégration de Camunda 7 (pré-intégration)

Même si l'intégration BPMN complète n'est pas finalisée, l'architecture du backend prévoit déjà l'utilisation du moteur Camunda 7.

Les points suivants sont prêts dans la structure logicielle :

- ajout des dépendances Camunda dans `pom.xml` ;
- configuration du moteur BPMN embarqué ;
- endpoints prévus pour déployer et exécuter des processus BPMN ;
- liaison future avec le module « Meeting » pour automatiser les workflows SCRUM.

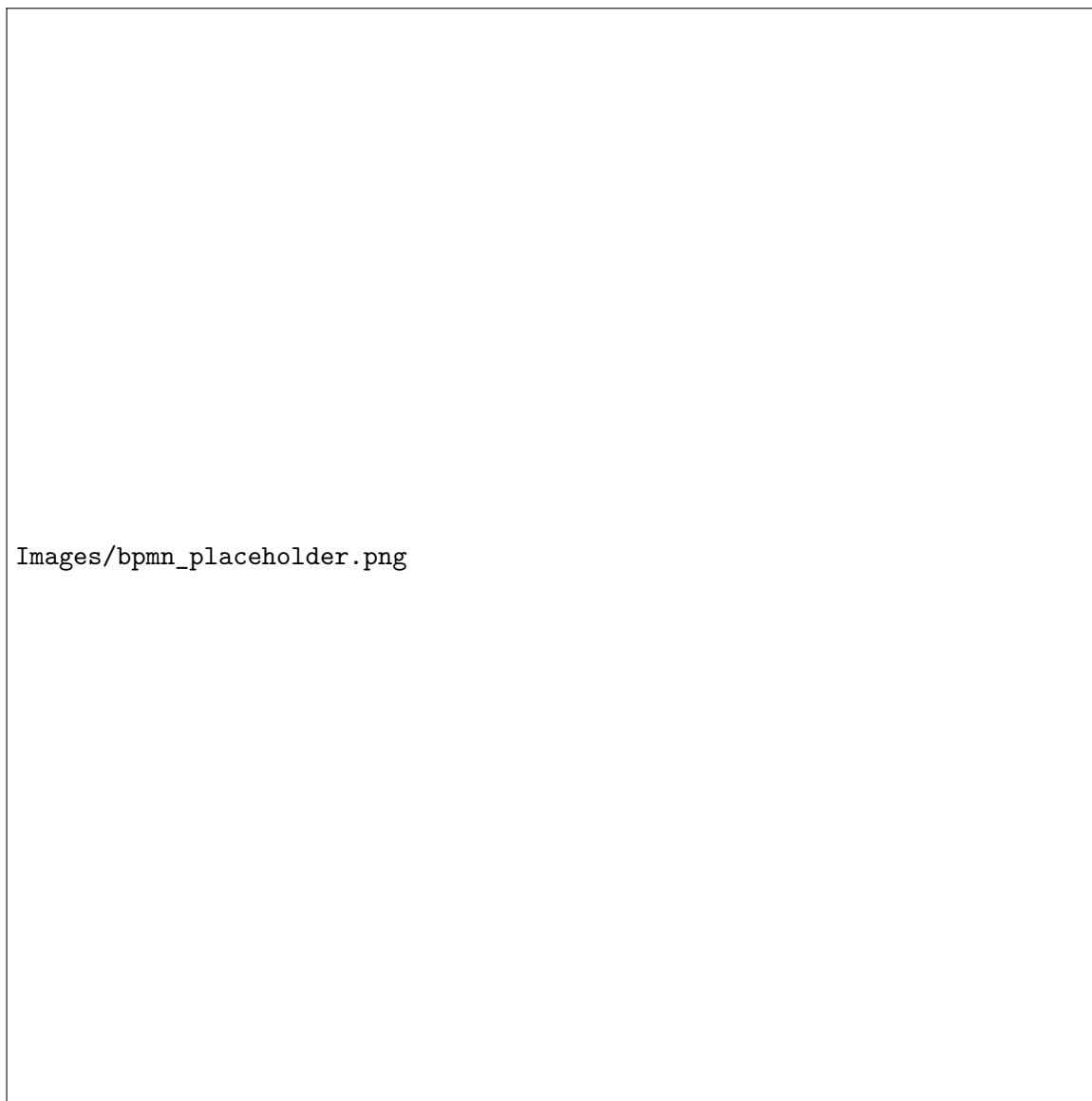


FIGURE G.6.1 – Exemple de workflow BPMN prévu pour intégration

Cette intégration permettra d'automatiser :

- la création des réunions SCRUM ;
- la gestion et le suivi des blocages ;
- les validations DoD ;
- les cycles Sprint Planning → Daily → Review → Rétrospective.

## G.7 Tests de déploiement

Des tests ont été effectués pour vérifier :

- la communication entre les conteneurs Docker ;
- l'exécution correcte des endpoints REST ;
- la persistance réelle dans PostgreSQL ;
- l'affichage de l'interface Angular en mode production ;
- la stabilité du backend sous charge modérée.

Ces tests ont confirmé la stabilité du système et son fonctionnement global.

## G.8 Conclusion

Le déploiement via Docker offre une installation rapide, un environnement standardisé et simplifie la maintenance de la plateforme SCRUM.

Il constitue également une base solide pour une future migration vers des infrastructures cloud ou Kubernetes, tout en garantissant une cohérence entre les différents modules du système.

Ainsi, cette architecture assure une exploitation fiable, évolutive et professionnelle de la solution développée.



## CHAPITRE H

# Résultats et Tests

## H.1 Introduction

Cette section présente l'ensemble des tests réalisés pour valider la plateforme SCRUM développée. Les tests effectués couvrent :

- les fonctionnalités principales (projets, backlog, sprints, tâches, blocages) ;
- les API REST du backend ;
- la persistance PostgreSQL ;
- les interfaces Angular ;
- la sécurité (JWT, rôles, permissions) ;
- le déploiement Docker.

L'objectif est de démontrer que la solution fonctionne de manière fiable, cohérente et conforme aux exigences fonctionnelles définies au début du projet.

## H.2 Tests fonctionnels

Les tests fonctionnels vérifient que chaque fonctionnalité répond correctement au comportement attendu.

### H.2.1 Test 1 : Crédation d'un projet

**Objectif :** vérifier qu'un Product Owner peut créer un projet et lui attribuer une équipe.

**Résultat attendu :**

- le projet apparaît dans la liste ;
- le Scrum Master et les développeurs sont correctement liés ;
- les permissions sont appliquées (seul le PO peut modifier le projet).

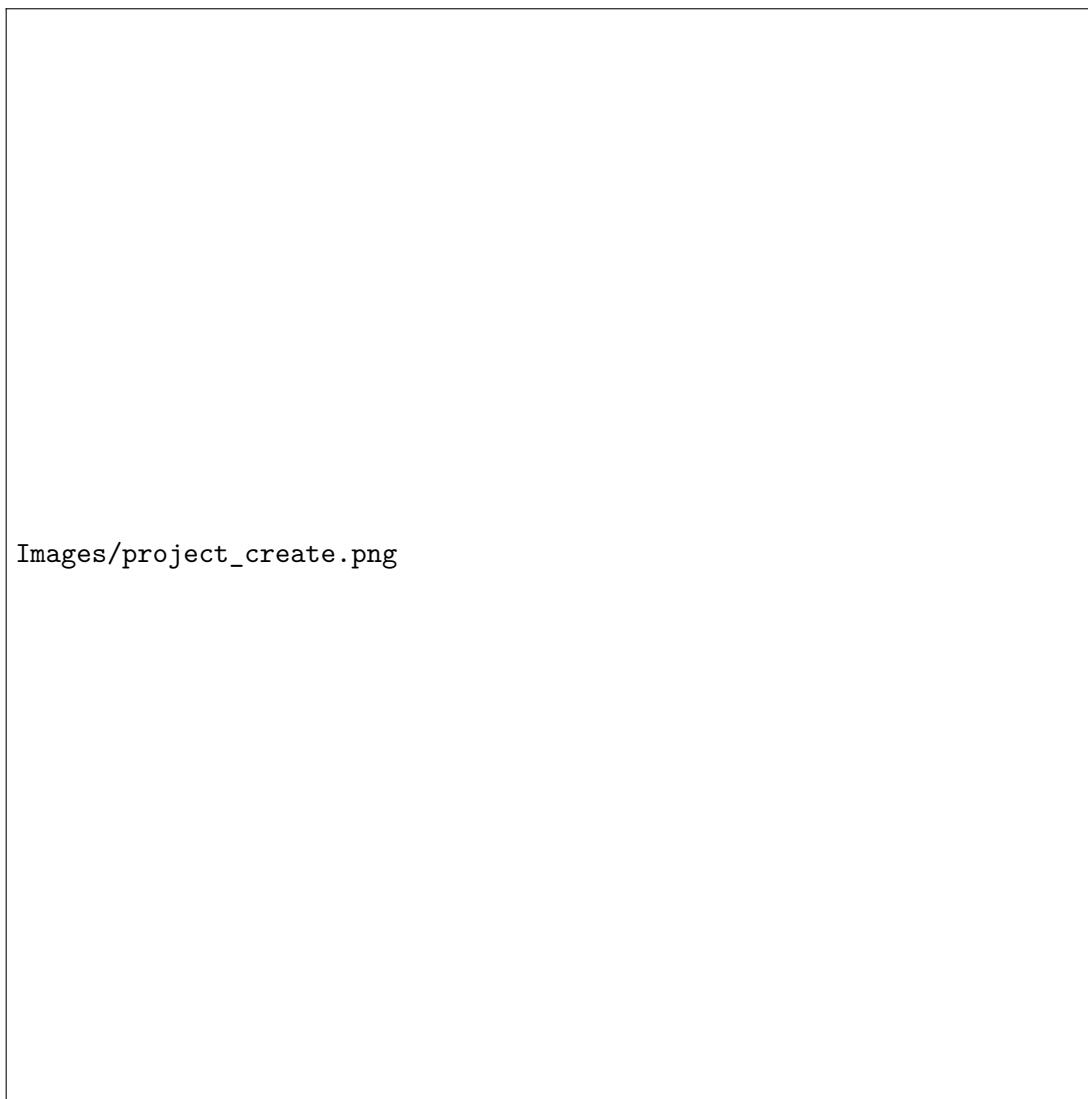


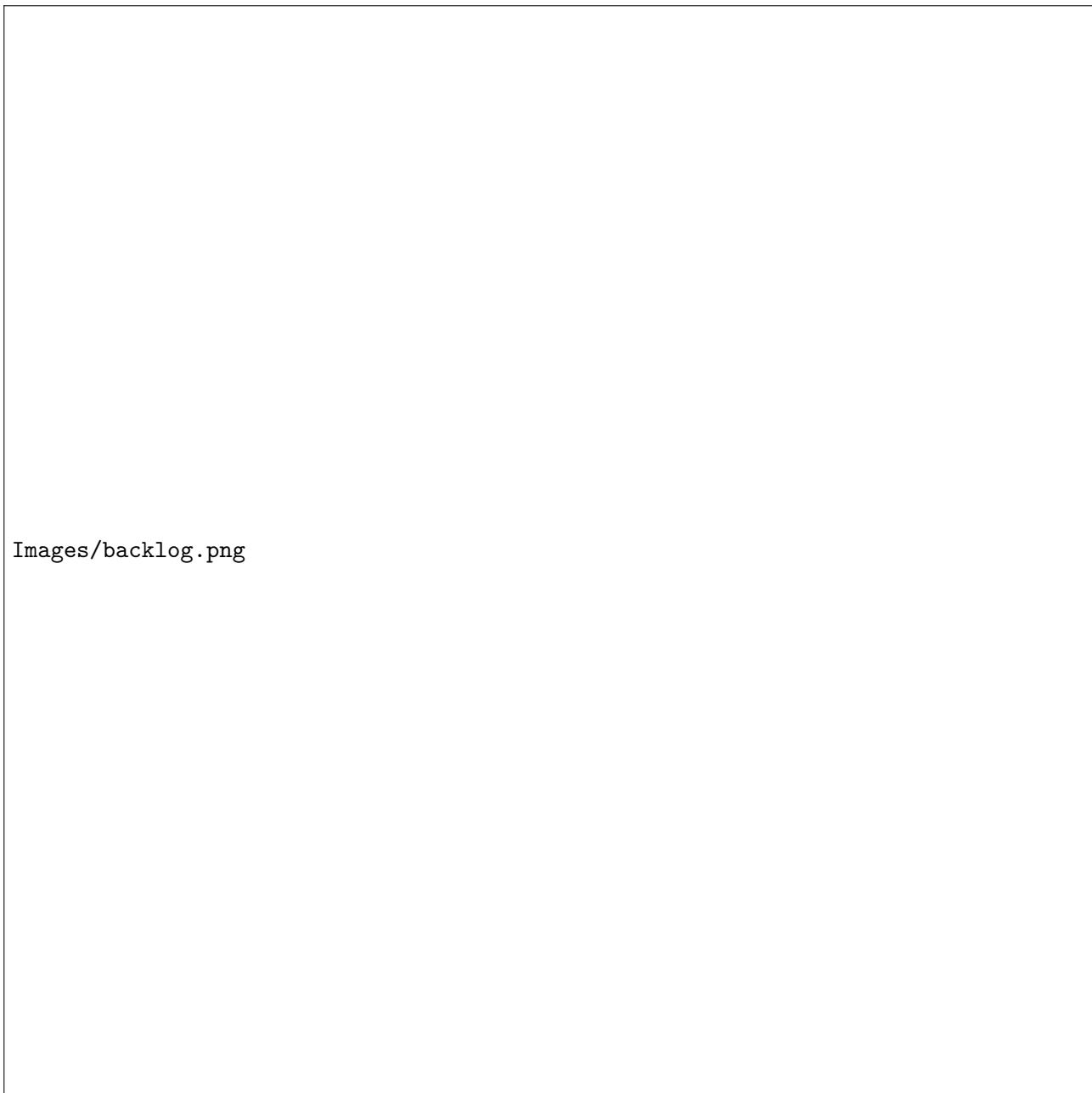
FIGURE H.2.1 – Résultat du test — Crédit d'un projet

### H.2.2 Test 2 : Gestion du Product Backlog

**Objectif :** vérifier que le PO peut créer, éditer et prioriser les stories.

**Résultat obtenu :**

- la création fonctionne correctement ;
- l'ordre de priorité est mis à jour en temps réel ;
- les développeurs ont une visibilité en lecture seule.



Images/backlog.png

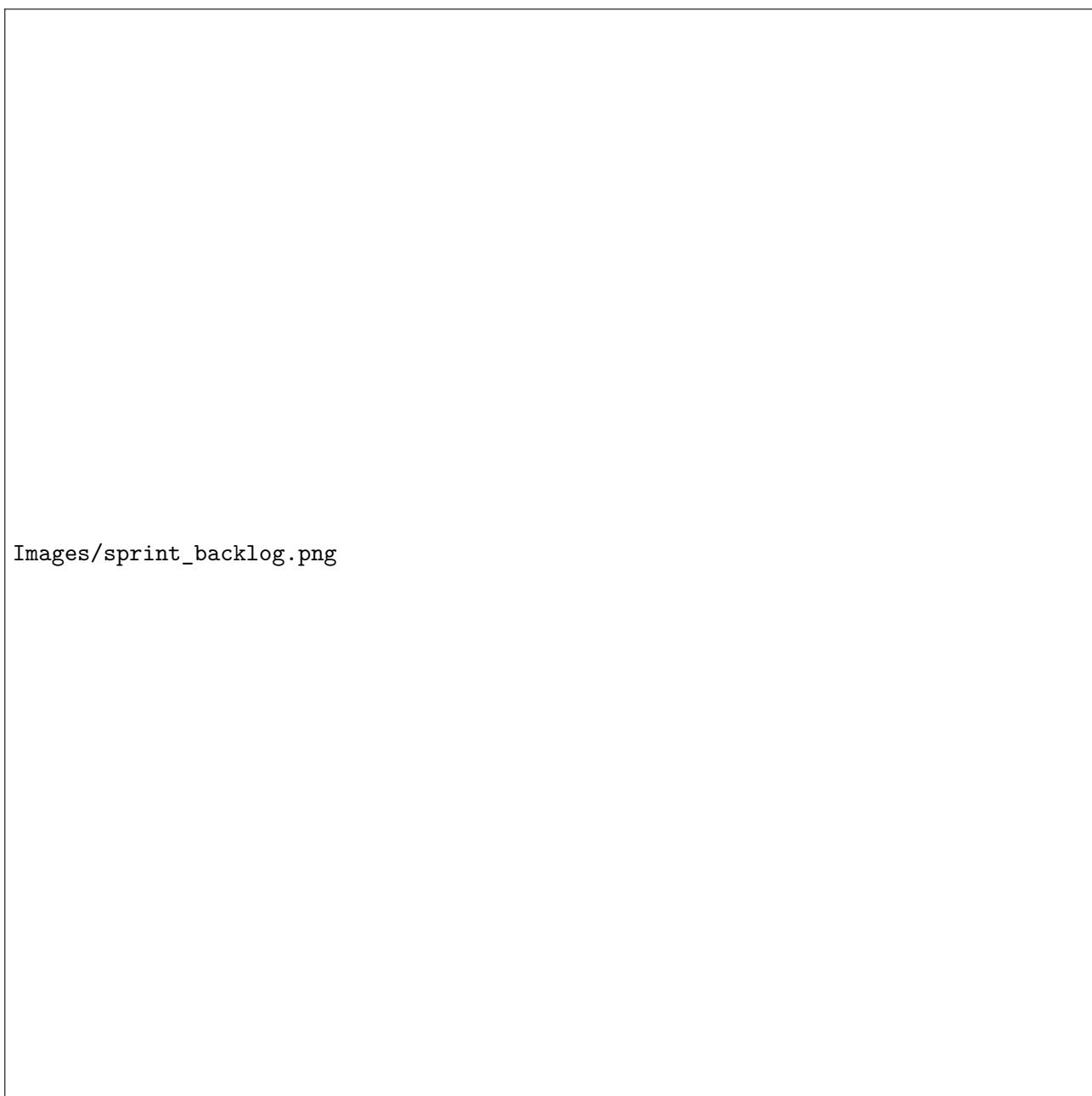
FIGURE H.2.2 – Test — Gestion du Product Backlog

### H.2.3 Test 3 : Création et gestion d'un Sprint

**Objectif :** vérifier que le Scrum Master peut créer un sprint, sélectionner les stories et générer les tâches.

**Résultat :** conforme.

- les tâches apparaissent dans le Sprint Board ;
- la capacité du sprint est respectée ;
- les transitions d'état fonctionnent parfaitement.



Images/sprint\_backlog.png

FIGURE H.2.3 – Test — Crédit du Sprint Backlog

#### H.2.4 Test 4 : Gestion des blocages

**Objectif :** tester la détection et le suivi des impediments.

**Résultat :**

- un développeur peut déclarer un blocage ;
- le Scrum Master reçoit une notification ;
- le blocage passe par les états : déclaré → en cours → résolu.



## H.3 Tests API REST (Postman)

Les endpoints ont été testés avec Postman : création de projet, création de story, mise à jour d'une tâche, etc.

### H.3.1 Exemple : Test de l'authentification

---

```

1 POST /auth/login
2 {
3     "username": "houssem",
4     "password": "1234"
5 }
```

---

Listing H.3.1 – Requête de test — Authentification JWT

**Résultat : succès.**

Le backend retourne un access token et un refresh token.

### H.3.2 Exemple : Test de création d'un projet

---

```

1 POST /projects
2 {
3     "name": "Demo SCRUM",
4     "description": "Test de création via Postman",
5     "scrumMasterId": 3
6 }
```

---

Listing H.3.2 – Requête API — Crédation de projet

**Résultat attendu : code 201 et retour du projet créé.**

**Résultat obtenu : conforme.**

## H.4 Tests de persistance PostgreSQL

Les tests ont permis de valider :

- la création automatique des tables via JPA ;
- les relations OneToMany, ManyToOne et ManyToMany ;
- l'intégrité référentielle (clé étrangère Sprint → Project, Task → Story, etc.) ;
- la suppression en cascade contrôlée.

Un extrait de la base après plusieurs tests :

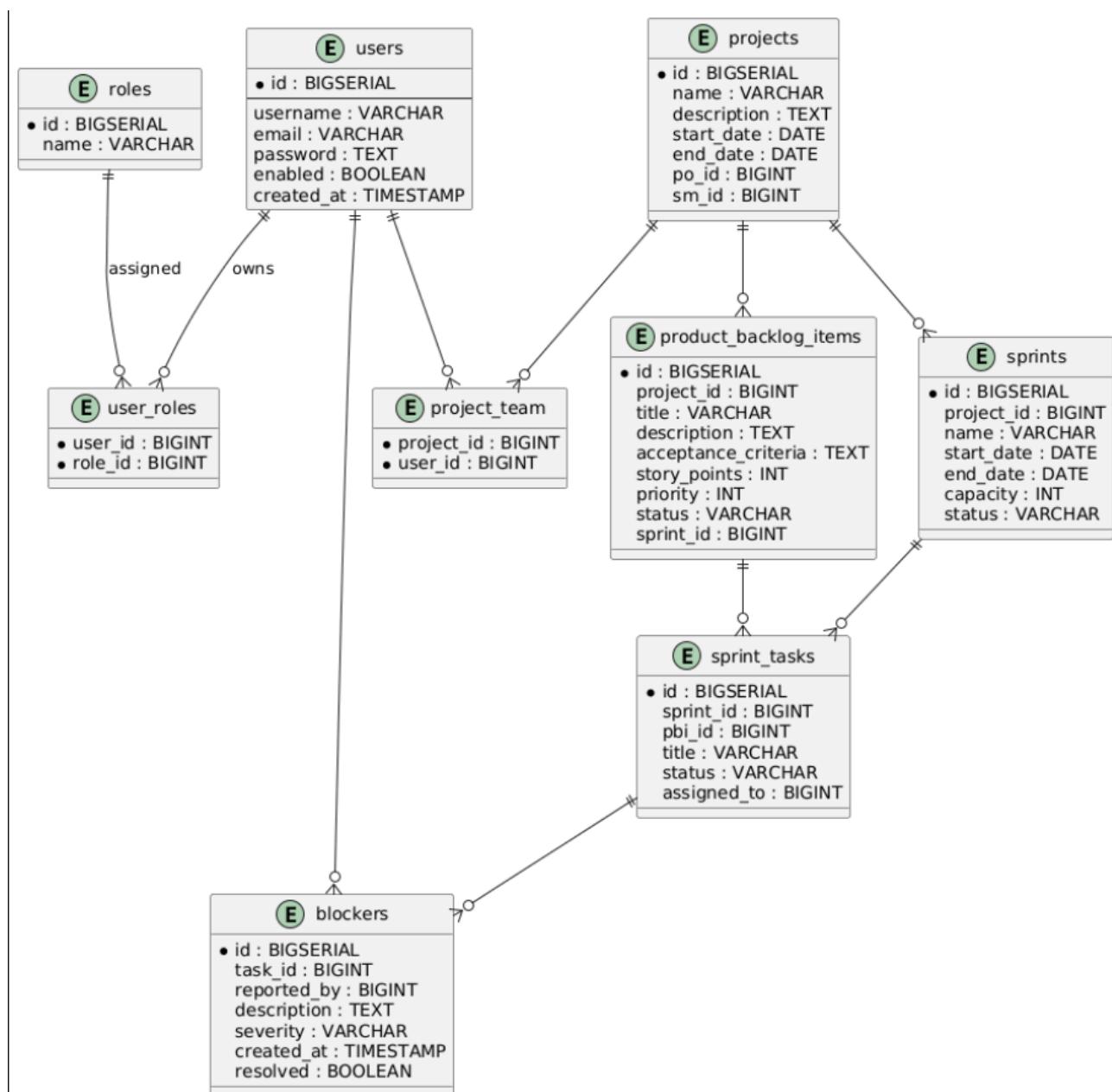


FIGURE H.4.1 – Schéma PostgreSQL observé après les tests

## H.5 Tests de l'interface Angular

Les tests UI confirment :

- une navigation fluide ;
- une cohérence graphique ;
- des rôles appliqués correctement (Admin, PO, SM, Dev) ;
- aucune anomalie dans l'affichage du Kanban ;
- une mise à jour instantanée des données via services REST.

Exemples d'écrans testés :

- Login (connexion JWT)
- Dashboard PO/SM



- Kanban Sprint
- Blocage (création et suivi)
- Sprint Backlog

## H.6 Tests de sécurité

Les validations suivantes ont été effectuées :

- accès protégé par JWT ;
- expiration des tokens correctement gérée ;
- routes sécurisées selon rôle (RBAC) ;
- tentative d'accès non autorisé renvoyant un code 403.

Exemple :

- un développeur ne peut pas modifier le Product Backlog ;
- un PO ne peut pas modifier les tâches du sprint ;
- un utilisateur non connecté n'accède à aucune ressource.

## H.7 Synthèse des résultats

Les tests révèlent que :

- l'application fonctionne de manière stable ;
- toutes les fonctionnalités principales SCRUM sont opérationnelles ;
- la communication frontend-backend est fiable ;
- la base de données est cohérente et robuste ;
- la sécurité JWT protège efficacement les données ;
- le déploiement Docker simplifie l'exécution du système.

Un tableau récapitulatif est présenté ci-dessous :

Test	Objectif	Résultat
Création projet	Fonctionnel	Succès
Gestion backlog	Fonctionnel	Succès
Sprint backlog	Fonctionnel	Succès
Blocages	Fonctionnel	Succès
API REST	Technique	Conforme
JWT	Sécurité	Conforme
UI Angular	Interface	Conforme
PostgreSQL	Persistance	Conforme
Docker	Déploiement	Conforme

TABLE H.7.1 – Synthèse des tests réalisés

## H.8 Conclusion

L'ensemble des tests réalisés confirme que la plateforme est stable, opérationnelle et répond parfaitement aux exigences du projet. La structure technique (Angular + Spring Boot + PostgreSQL + Docker) garantit un fonctionnement fiable, tandis que la pré-intégration Camunda assure une évolution future vers une automatisation complète des processus SCRUM.



## CHAPITRE I

# Perspectives d'amélioration et Conclusion générale

## I.1 Introduction

Ce chapitre présente les pistes d'évolution possibles du système développé, ainsi que la conclusion générale du projet. Bien que la plateforme soit pleinement fonctionnelle et réponde aux objectifs fixés (initialisation de projet, gestion SCRUM, tableau Kanban, gestion des blocages, authentification sécurisée, etc.), plusieurs axes d'amélioration ont été identifiés, notamment autour de l'automatisation, de l'expérience utilisateur et de l'architecture technique.

## I.2 Perspectives d'amélioration

### I.2.1 Automatisation avancée via Camunda BPM

L'intégration de Camunda ouvre la voie à une automatisation plus poussée du processus SCRUM :

- **Automatisation du Sprint Planning** : génération automatique des tâches techniques, prise en compte de la capacité, sélection intelligente des PBIs.
- **Daily Scrum automatisé** : collecte automatique des mises à jour, détection de blocages récurrents, archivage des impediments.
- **Workflow de validation DoD** : processus BPMN garantissant le respect systématique des critères d'acceptation.
- **Génération automatique du rapport de sprint** : compilation des métriques, blocages et progrès.

Ces axes permettraient de transformer l'outil actuel en une véritable plateforme SCRUM semi-autonome.

### I.2.2 Améliorations backend

Plusieurs évolutions techniques peuvent augmenter la robustesse du backend :

- finalisation du **module Meetings** (Daily, Review, Rétrospective) ;
- usage de **WebSockets / STOMP** pour le temps réel (Kanban, blocages, notifications) ;
- mise en place d'un **système de cache Redis** pour accélérer le tableau de bord ;
- ajout de **tests unitaires et d'intégration** pour renforcer la maintenabilité.

### I.2.3 Améliorations frontend (Angular)

L'expérience utilisateur pourrait être enrichie grâce à :

- une interface modernisée (Angular Material, Tailwind) ;
- un **mode sombre** et options d'accessibilité ;
- un drag-and-drop avancé pour les stories et tâches ;
- un tableau de bord enrichi (vitesse prédictive, analyse des blocages).

### I.2.4 Évolutions de l'architecture technique

À moyen terme, l'application pourrait évoluer vers :

- une architecture **microservices** (scalabilité, isolation, tolérance aux pannes) ;
- un déploiement sur **Kubernetes** pour la haute disponibilité ;
- une chaîne **CI/CD complète** (tests, build, analyse sécurité, déploiement automatique) ;
- l'intégration d'un fournisseur d'identité externe (Keycloak) pour un SSO professionnel.

### I.2.5 Améliorations fonctionnelles

Certaines évolutions permettraient d'enrichir le périmètre fonctionnel :

- module d'**export PDF** des rapports de sprint et rétrospectives ;
- messagerie interne PO/SM/Dev pour améliorer la communication ;
- gestion multi-projets avancée ;
- import/export des backlogs (CSV, JSON, Jira).

## I.3 Conclusion générale

Le développement de cette plateforme de gestion SCRUM a permis de mettre en œuvre une solution complète couvrant la majorité des besoins d'un environnement Agile moderne : gestion de projet, Product Backlog, Sprint Backlog, tableau Kanban, suivi des blocages, reporting et sécurité JWT.

Sur le plan technique, l'architecture Angular / Spring Boot / PostgreSQL a offert une base solide, modulaire et extensible. L'intégration préliminaire de Camunda ouvre la voie à une automatisation plus large des processus BPMN tels que le Sprint Planning, la gestion des impediments et la validation des livrables.

Au-delà de l'aspect technique, ce projet a permis de mieux comprendre les enjeux opérationnels du SCRUM : coordination des équipes, importance de la transparence, gestion des imprévus et adaptation continue.

En conclusion, la plateforme constitue une base robuste, fonctionnelle et évolutive. Les perspectives d'amélioration identifiées permettront d'enrichir progressivement l'outil pour en faire une solution complète d'orchestration SCRUM appuyée par des processus BPMN automatisés.

# Liste des figures

A.1.1	Logo de l'entreprise IDVEY . . . . .	8
A.5.1	Organigramme de l'entreprise IDVEY . . . . .	10
B.6.1	Workflow SCRUM général du projet (schéma conceptuel) . . . . .	14
C.4.1	Symboles des événements BPMN . . . . .	24
C.4.2	Symboles des activités BPMN . . . . .	25
C.4.3	Symboles des passerelles BPMN . . . . .	25
C.4.4	Symboles des connecteurs BPMN . . . . .	25
C.4.5	Exemple de processus BPMN : Analyse d'une réclamation . . . . .	26
C.4.6	Exemple de processus BPMN : Création d'un devis . . . . .	26
C.5.1	Architecture générale de Camunda 7 . . . . .	27
D.3.1	Diagramme des cas d'utilisation du système . . . . .	32
D.4.1	Diagramme de classes du système . . . . .	34
D.4.2	Diagramme de séquence — Authentification JWT . . . . .	35
D.4.3	Diagramme de séquence — Création d'un projet . . . . .	36
D.4.4	Diagramme de séquence — Création d'un product backlog . . . . .	37
D.4.5	Diagramme de séquence — Création d'un sprint backlog . . . . .	38
D.4.6	Diagramme d'activité global du système avec swimlanes . . . . .	39
D.5.1	Architecture logicielle de la plateforme (Angular / Spring Boot / PostgreSQL / Camunda)	41
D.5.2	Schéma de la base de données PostgreSQL . . . . .	43
D.6.1	Processus BPMN — Création d'un projet . . . . .	44
D.6.2	Processus BPMN — Sprint SCRUM . . . . .	45
D.7.1	Architecture technique déployée avec Docker . . . . .	46
F.2.1	Structure générale du frontend Angular . . . . .	69
F.6.1	Interface de connexion de l'application . . . . .	72
F.6.2	Liste des projets du Product Owner . . . . .	73
F.6.3	Formulaire de création d'un projet . . . . .	74

F.6.4	Affectation du Scrum Master et des développeurs . . . . .	75
F.6.5	Vue du Product Backlog . . . . .	76
F.6.6	Création d'une User Story . . . . .	77
F.6.7	Liste des sprints du projet . . . . .	78
F.6.8	Sprint Backlog — sélection des stories et tâches . . . . .	79
F.6.9	Tableau Kanban du sprint . . . . .	80
F.6.10	Signalement d'un blocage . . . . .	81
F.6.11	Vue des blocages (Scrum Master) . . . . .	82
F.6.12	Métriques du projet : burndown, vitesse, blocages . . . . .	83
F.6.13	Gestion des utilisateurs . . . . .	84
G.2.1	Architecture technique globale de la plateforme (Docker) . . . . .	86
G.6.1	Exemple de workflow BPMN prévu pour intégration . . . . .	89
H.2.1	Résultat du test — Création d'un projet . . . . .	92
H.2.2	Test — Gestion du Product Backlog . . . . .	93
H.2.3	Test — Création du Sprint Backlog . . . . .	94
H.4.1	Schéma PostgreSQL observé après les tests . . . . .	96
A.1.1	Interface de connexion de l'application . . . . .	114
A.2.1	Liste des projets accessibles par l'utilisateur . . . . .	115
A.2.2	Création d'un projet et affectation du Scrum Master . . . . .	116
A.2.3	Gestion de l'équipe projet : PO, SM et développeurs . . . . .	117
A.3.1	Vue du Product Backlog : priorisation des stories . . . . .	118
A.3.2	Création d'une nouvelle User Story . . . . .	119
A.4.1	Liste des sprints du projet . . . . .	120
A.4.2	Sprint Backlog : sélection des stories et création des tâches . . . . .	121
A.5.1	Tableau Kanban utilisé par les développeurs . . . . .	122
A.6.1	Signalement d'un blocage par un développeur . . . . .	123
A.6.2	Liste des blocages en cours (Scrum Master) . . . . .	124
A.7.1	Métriques projet : vitesse, burndown, blocages . . . . .	125
A.8.1	Gestion des utilisateurs et des rôles . . . . .	126
B.1.1	Documentation Swagger générée par Spring Boot . . . . .	128
C.1.1	BPMN — Processus de création de projet . . . . .	131
C.2.1	BPMN — Processus de gestion du sprint . . . . .	132
D.1.1	Diagramme de classes complet du système . . . . .	133
D.2.1	Cas d'utilisation : Vue globale du système . . . . .	135
D.3.1	Diagramme de séquence — Authentification JWT . . . . .	136

---

E.2.1    Architecture technique déployée (Docker + services) . . . . .	138
--	-----



# Liste des tableaux

A.3.1	Fiche signalétique de l'entreprise IDVEY . . . . .	9
C.3.1	Comparaison synthétique des principaux outils SCRUM . . . . .	22
C.5.1	Comparaison détaillée entre Camunda 7 et Camunda 8 . . . . .	28
H.7.1	Synthèse des tests réalisés . . . . .	97



# Liste des équations



# Liste des codes

G.3.1	Configuration du déploiement via Docker Compose . . . . .	87
G.5.1	Dockerfile du frontend Angular . . . . .	88
H.3.1	Requête de test — Authentification JWT . . . . .	95
H.3.2	Requête API — Création de projet . . . . .	95
B.2.1	Exemple de réponse JSON retornée par /auth/login . . . . .	129
B.3.1	Requête JSON pour créer un projet . . . . .	129
B.4.1	Exemple de logs du backend . . . . .	129
E.1.1	Fichier docker-compose utilisé pour le déploiement local . . . . .	137



# Bibliographie

- [1] P. J. COHEN. « The independence of the continuum hypothesis ». In : *Proceedings of the National Academy of Sciences* 50.6 (1963), p. 1143-1148.
- [2] Leonard SUSSKIND et George HRABOVSKY. *Classical mechanics : the theoretical minimum*. New York, NY : Penguin Random House, 2014.
- [3] Maria SWETLA. *Canoe tours in Sweden*. Distributed at the Stockholm Tourist Office. Juill. 2015.
- [4] Lisa A. URRY et al. « Photosynthesis ». In : *Campbell Biology*. New York, NY : Pearson, 2016, p. 187-221.
- [5] Howard M. SHAPIRO. « Flow Cytometry : The Glass Is Half Full ». In : *Flow Cytometry Protocols*. Sous la dir. de Teresa S. HAWLEY et Robert G. HAWLEY. New York, NY : Springer, 2018, p. 1-10.
- [6] R CORE TEAM. *R : A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2018.
- [7] NASA. *Pluto : The 'Other' Red Planet*. <https://www.nasa.gov/nh/pluto-the-other-red-planet>. Accessed : 2018-12-06. 2015.
- [8] Vicki BENNETT, Kate BOWMAN et Sarah WRIGHT. *Wasatch Solar Project Final Report*. Rapp. tech. DOE-SLC-6903-1. Salt Lake City, UT : Salt Lake City Corporation, sept. 2018.



# Conclusion

Ce projet de fin d'études a permis de concevoir et développer une plateforme complète de gestion SCRUM adaptée aux besoins d'IDVEY. L'application modernise le suivi des projets en intégrant la gestion du Product Backlog, des Sprints, des tâches techniques et des blocages, tout en offrant une structure extensible pour l'intégration future de workflows BPMN via Camunda.

Les technologies retenues — Angular, Spring Boot, PostgreSQL et Docker — ont démontré leur pertinence en termes de performance, maintenabilité et évolutivité. Ce travail a également introduit les concepts d'automatisation des processus métier à travers BPMN 2.0, posant les bases d'une orchestration logicielle avancée.

Au-delà des aspects techniques, ce projet m'a permis de renforcer mes compétences en architecture logicielle, développement full-stack, modélisation UML, gestion Agile et déploiement d'infrastructures containerisées. Plusieurs perspectives d'évolution sont envisageables : automatisation complète du cycle SCRUM, amélioration des métriques, déploiement cloud ou intégration d'intelligence artificielle.

Ce rapport témoigne ainsi de la maîtrise d'un cycle de développement complet, depuis la conception jusqu'au déploiement, et constitue une base solide pour l'amélioration continue du système au sein d'IDVEY.



## Annexes



# Table des annexes

<b>Annexe A Annexe A — Interfaces Frontend Angular</b>	<b>114</b>
A.1 Authentification . . . . .	114
A.2 Gestion des projets . . . . .	115
A.3 Product Backlog . . . . .	118
A.4 Gestion des sprints . . . . .	120
A.5 Tableau Kanban . . . . .	122
A.6 Gestion des blocages (Impediments) . . . . .	123
A.7 Dashboard SCRUM . . . . .	125
A.8 Module Administration . . . . .	126
<b>Annexe B Annexe B — Backend (API REST, Logs, JSON)</b>	<b>127</b>
B.1 Documentation API REST (Swagger) . . . . .	128
B.2 Exemple de réponse JSON — Authentification JWT . . . . .	129
B.3 Exemple d'appel API — Création d'un projet . . . . .	129
B.4 Extrait de logs Spring Boot . . . . .	129
<b>Annexe C Annexe C — BPMN (Camunda)</b>	<b>130</b>
C.1 Processus BPMN — Création de projet . . . . .	131
C.2 Processus BPMN — Sprint Workflow . . . . .	132
<b>Annexe D Annexe D — UML (Modélisation du Système)</b>	<b>133</b>
D.1 Diagramme de classes . . . . .	133
D.2 Diagramme de cas d'utilisation . . . . .	135



---

D.3 Diagrammes de séquence . . . . .	136
<b>Annexe E Annexe E — Déploiement Docker et Infrastructure</b>	<b>137</b>
E.1 Fichier docker-compose.yml . . . . .	137
E.2 Architecture technique déployée . . . . .	138



# Introduction des annexes

Les annexes présentées dans cette section complètent le contenu du rapport en montrant :

- les interfaces réelles du frontend Angular,
- des extraits du backend Spring Boot (API REST, logs, JSON),
- les processus BPMN modélisés dans Camunda,
- les diagrammes UML utilisés pour la conception,
- les éléments techniques liés au déploiement Docker.

Elles constituent une preuve de la mise en œuvre concrète du projet ainsi qu'un support technique pour la reproductibilité et l'évolution du système.



## ANNEXE A

# Annexe A — Interfaces Frontend Angular

## A.1 Authentification

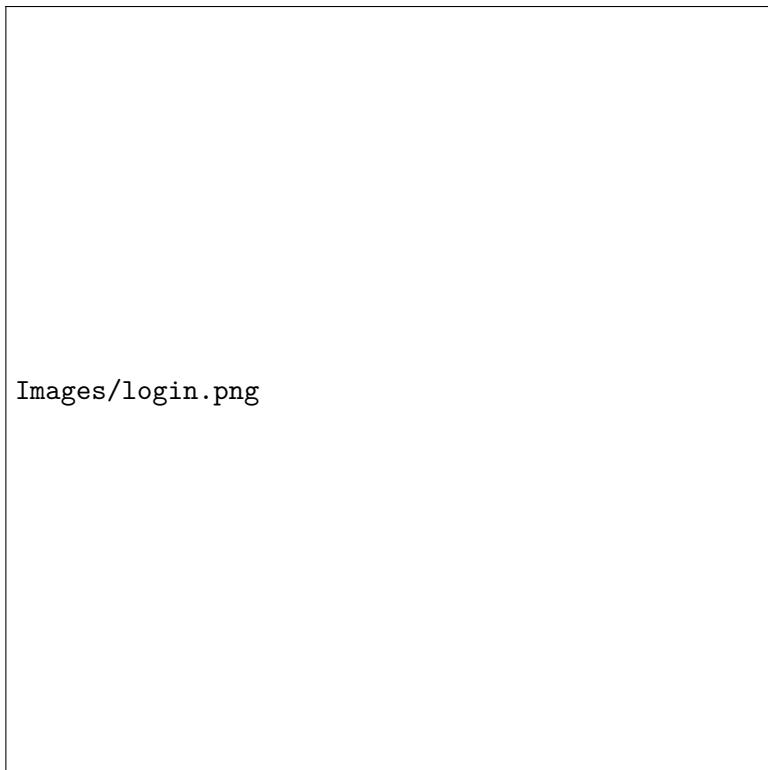
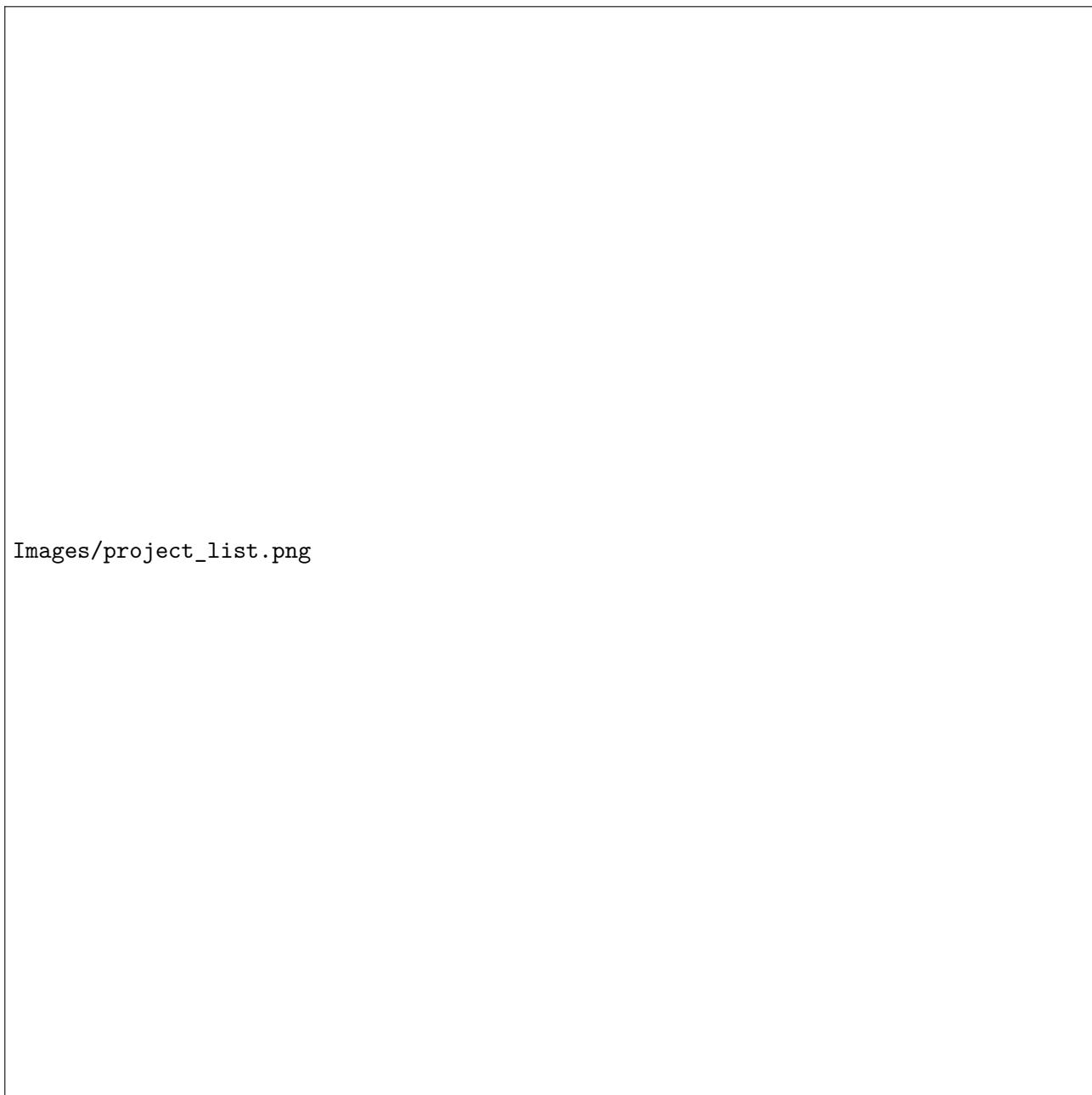


FIGURE A.1.1 – Interface de connexion de l’application

## A.2 Gestion des projets



Images/project\_list.png

FIGURE A.2.1 – Liste des projets accessibles par l'utilisateur

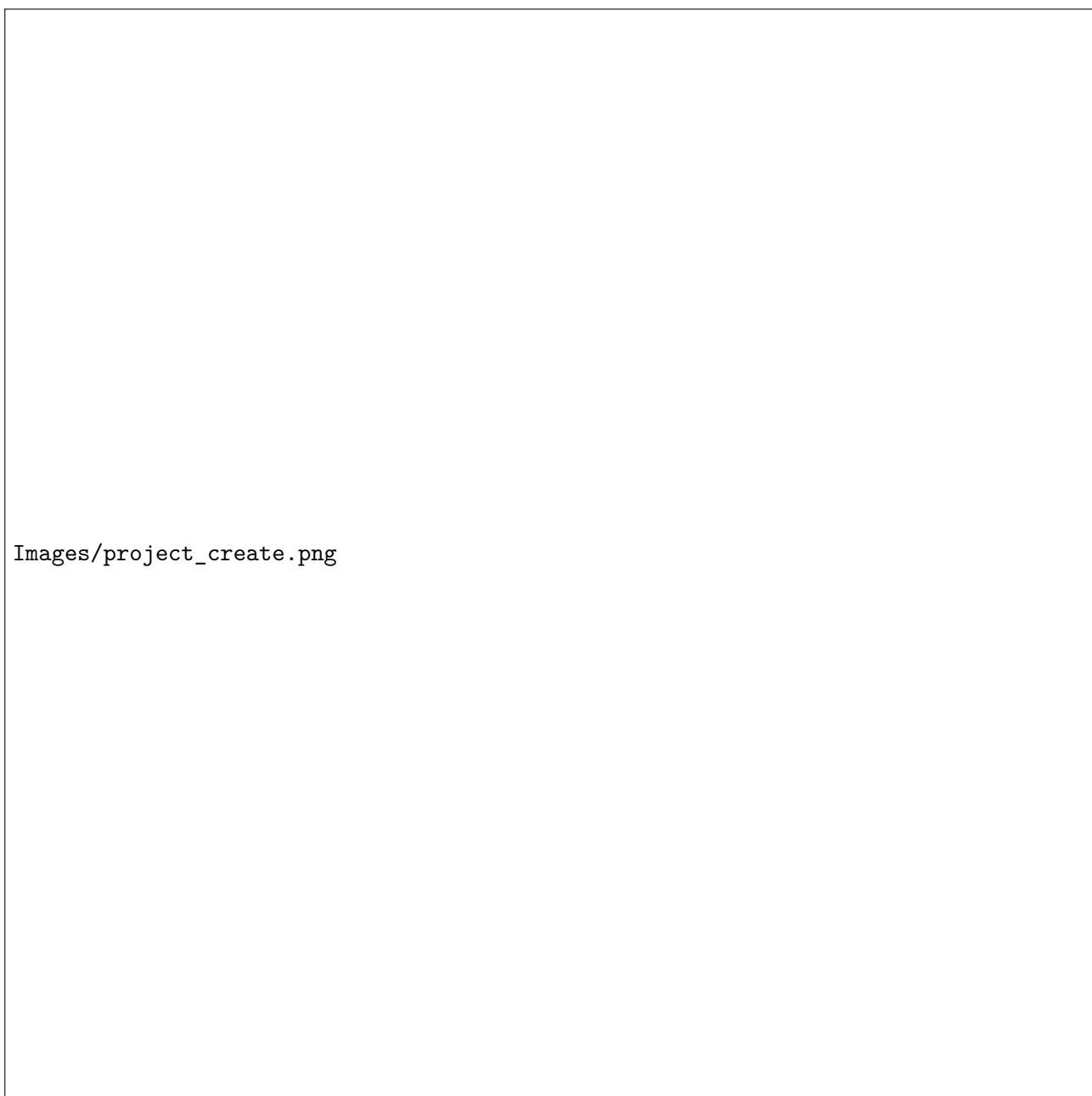
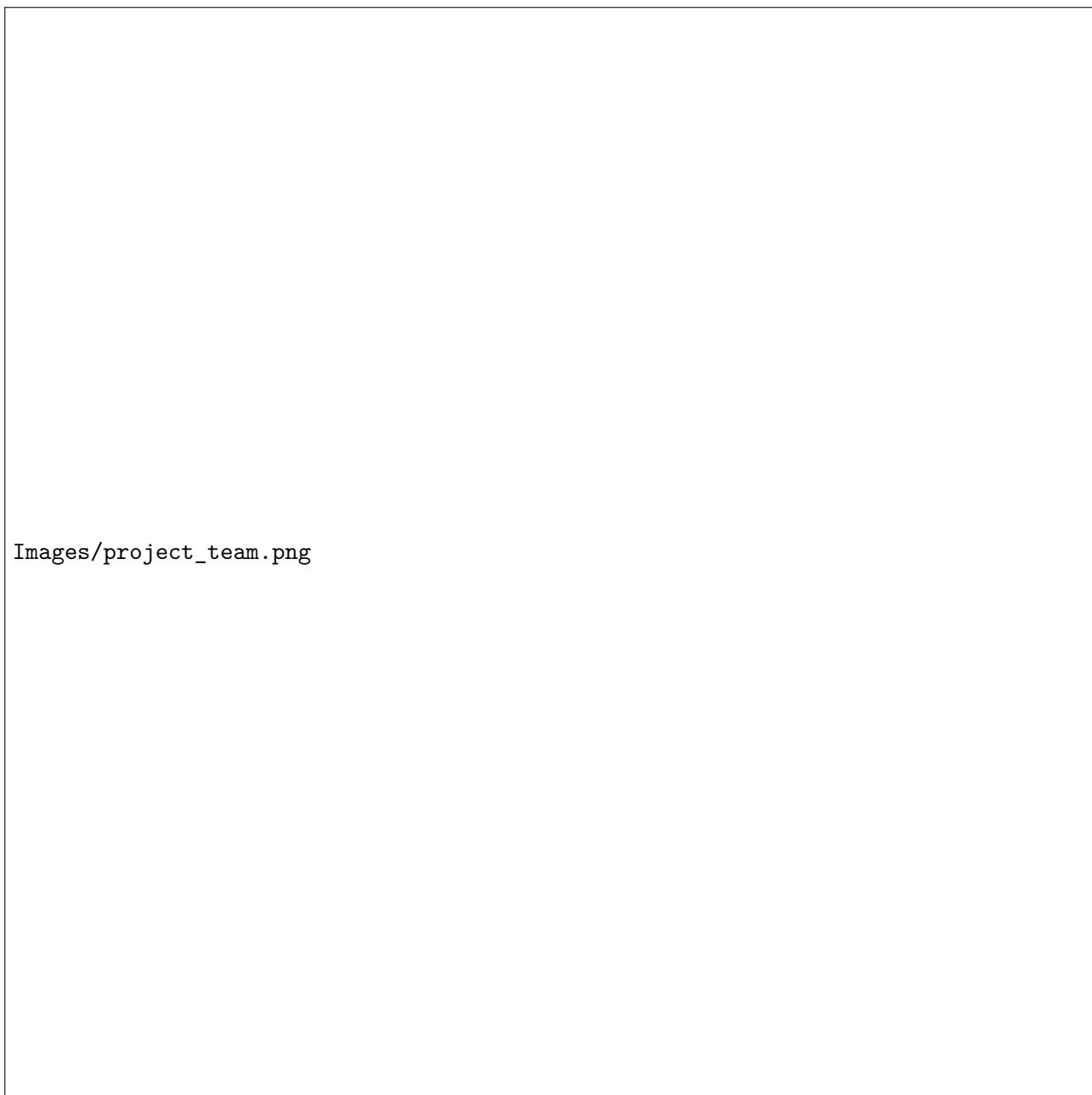


FIGURE A.2.2 – Création d'un projet et affectation du Scrum Master



Images/project\_team.png

FIGURE A.2.3 – Gestion de l'équipe projet : PO, SM et développeurs

### A.3 Product Backlog



Images/backlog.png

FIGURE A.3.1 – Vue du Product Backlog : priorisation des stories

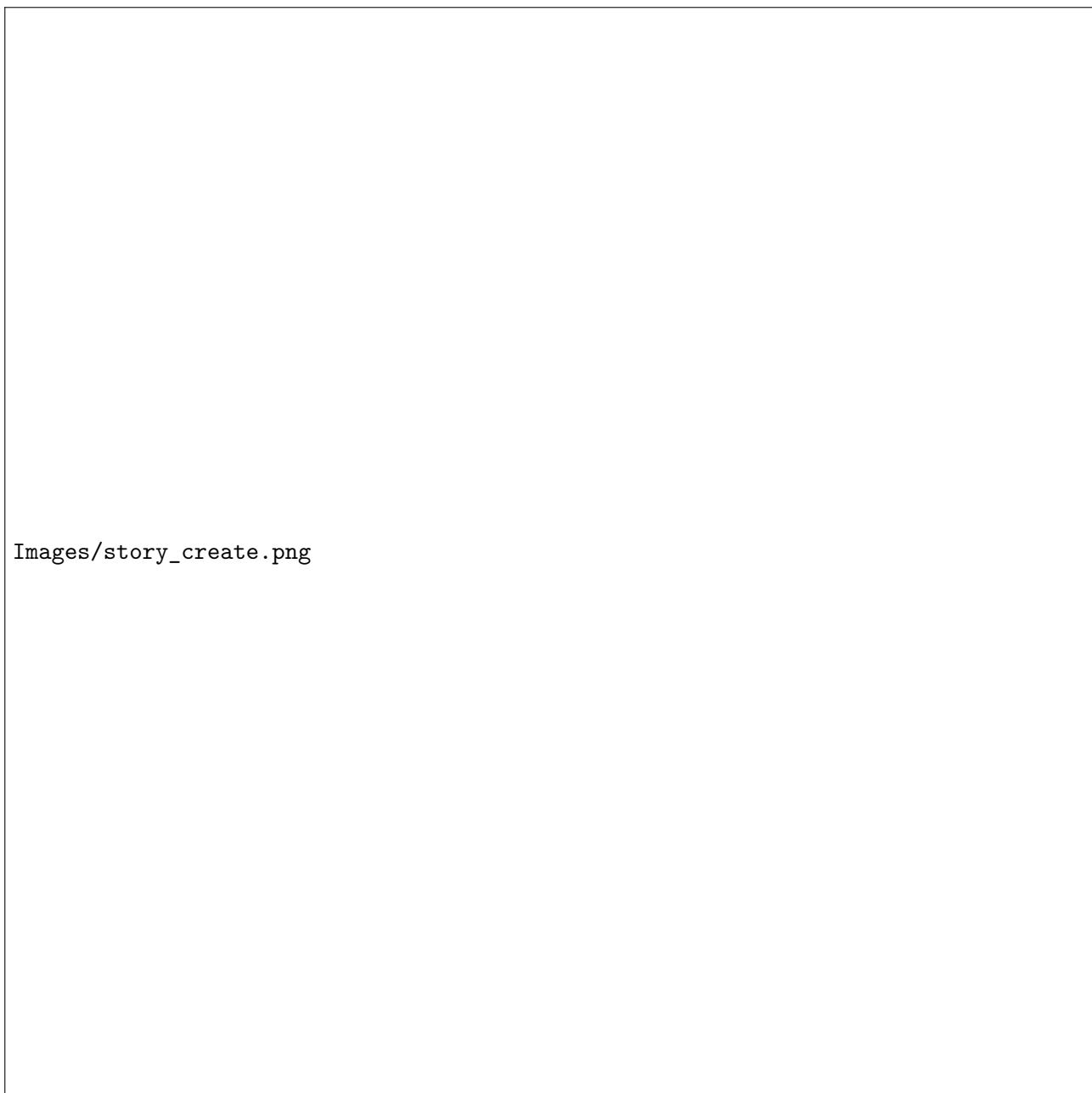


FIGURE A.3.2 – Création d'une nouvelle User Story

## A.4 Gestion des sprints

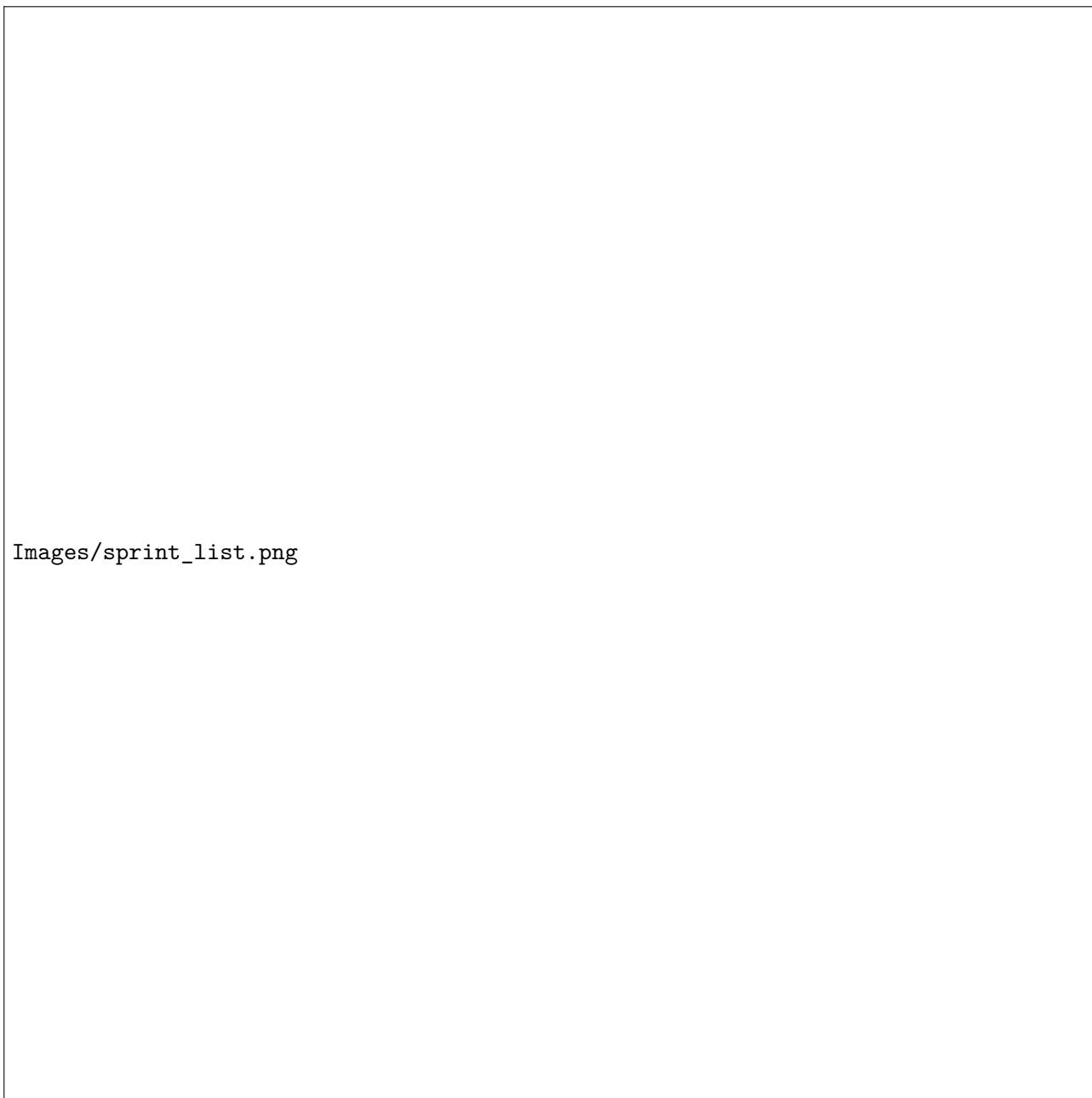
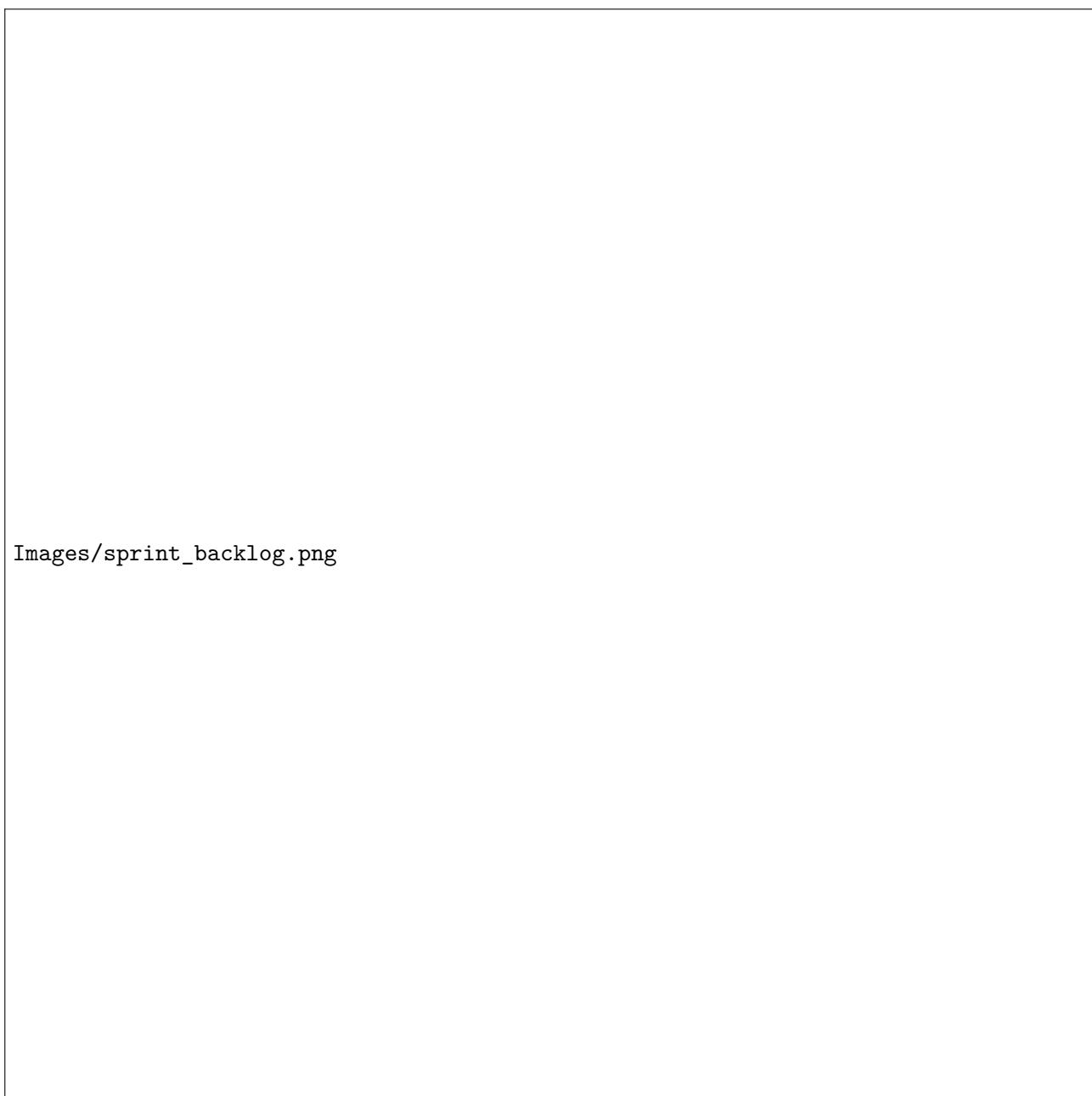


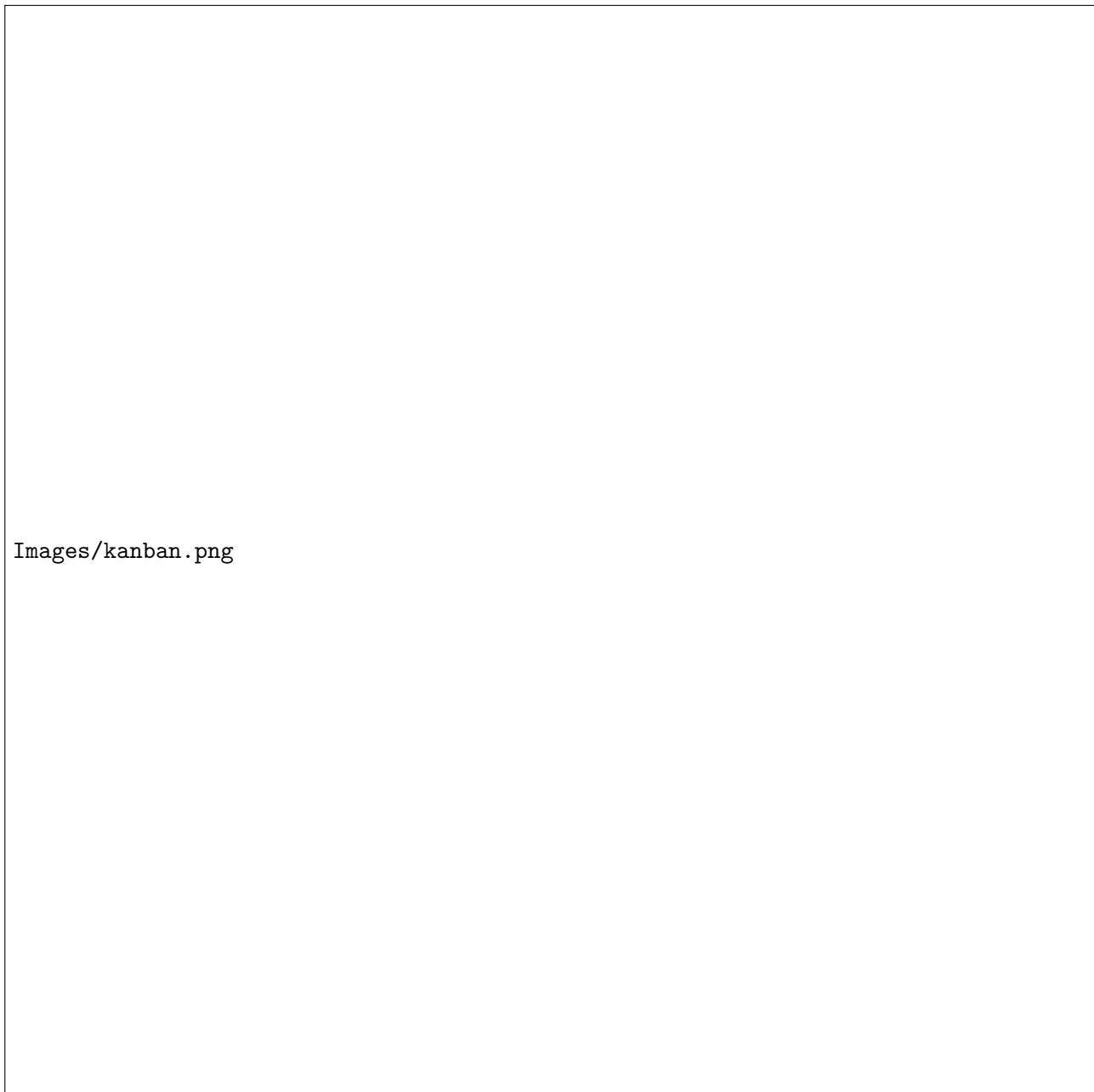
FIGURE A.4.1 – Liste des sprints du projet



Images/sprint\_backlog.png

FIGURE A.4.2 – Sprint Backlog : sélection des stories et création des tâches

## A.5 Tableau Kanban



Images/kanban.png

FIGURE A.5.1 – Tableau Kanban utilisé par les développeurs

## A.6 Gestion des blocages (Impediments)

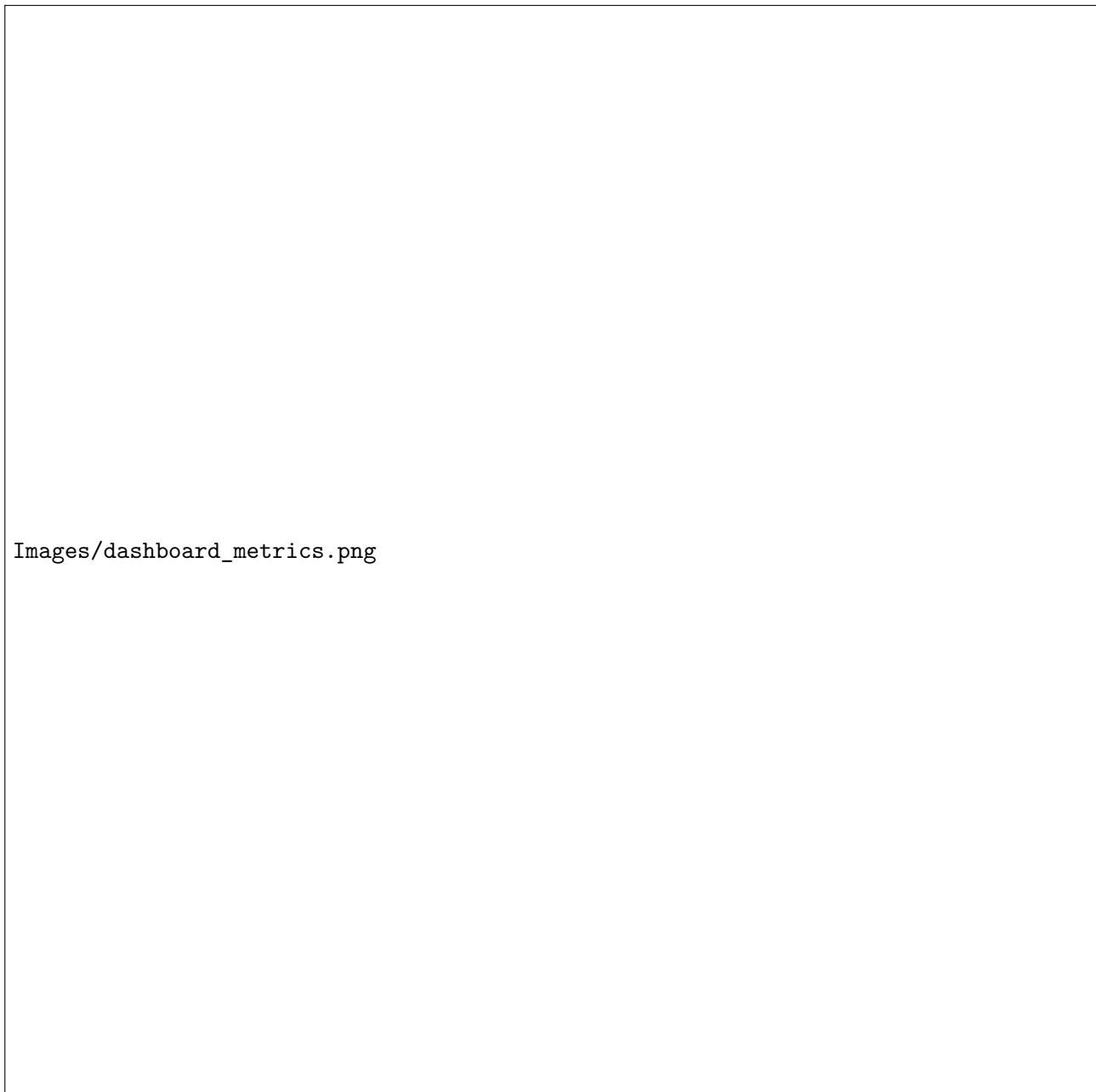


FIGURE A.6.1 – Signalement d'un blocage par un développeur



FIGURE A.6.2 – Liste des blocages en cours (Scrum Master)

## A.7 Dashboard SCRUM



Images/dashboard\_metrics.png

FIGURE A.7.1 – Métriques projet : vélocité, burndown, blocages

## A.8 Module Administration



Images/admin\_users.png

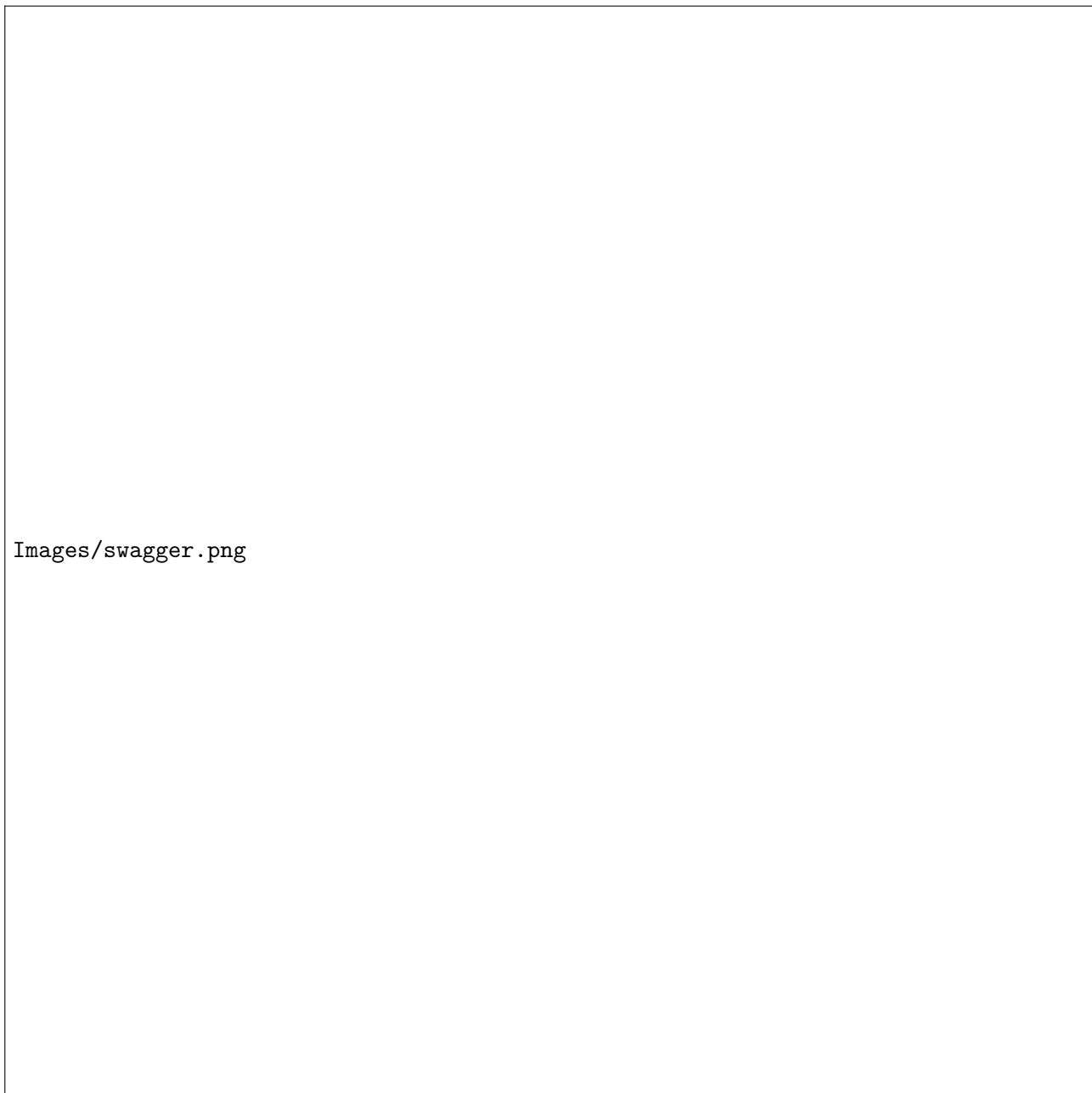
FIGURE A.8.1 – Gestion des utilisateurs et des rôles



## ANNEXE B

# Annexe B — Backend (API REST, Logs, JSON)

## B.1 Documentation API REST (Swagger)



Images/swagger.png

FIGURE B.1.1 – Documentation Swagger générée par Spring Boot



## B.2 Exemple de réponse JSON — Authentification JWT

---

```

1 {
2     "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
3     "refreshToken": "eyJhbGciOiJIUzUxMiIsInR5c..."
4 }
```

---

Listing B.2.1 – Exemple de réponse JSON retournée par /auth/login

## B.3 Exemple d'appel API — Crédation d'un projet

---

```

1 POST /projects
2 {
3     "name": "Plateforme Agile SCRUM",
4     "description": "Digitalisation complète du processus SCRUM",
5     "scrumMasterId": 5
6 }
```

---

Listing B.3.1 – Requête JSON pour créer un projet

## B.4 Extrait de logs Spring Boot

---

```

1 2025-01-23 14:11:37 INFO c.p.s.AuthService - Authentication successful for user 'housssem'
2 2025-01-23 14:11:37 INFO c.p.s.ProjectService - Project 'PFE SCRUM Automation' created
3 2025-01-23 14:11:38 INFO c.p.s.SprintService - Sprint 'Sprint 1' initialized
```

---

Listing B.4.1 – Exemple de logs du backend



ANNEXE C



# Annexe C — BPMN (Camunda)

## C.1 Processus BPMN — Création de projet

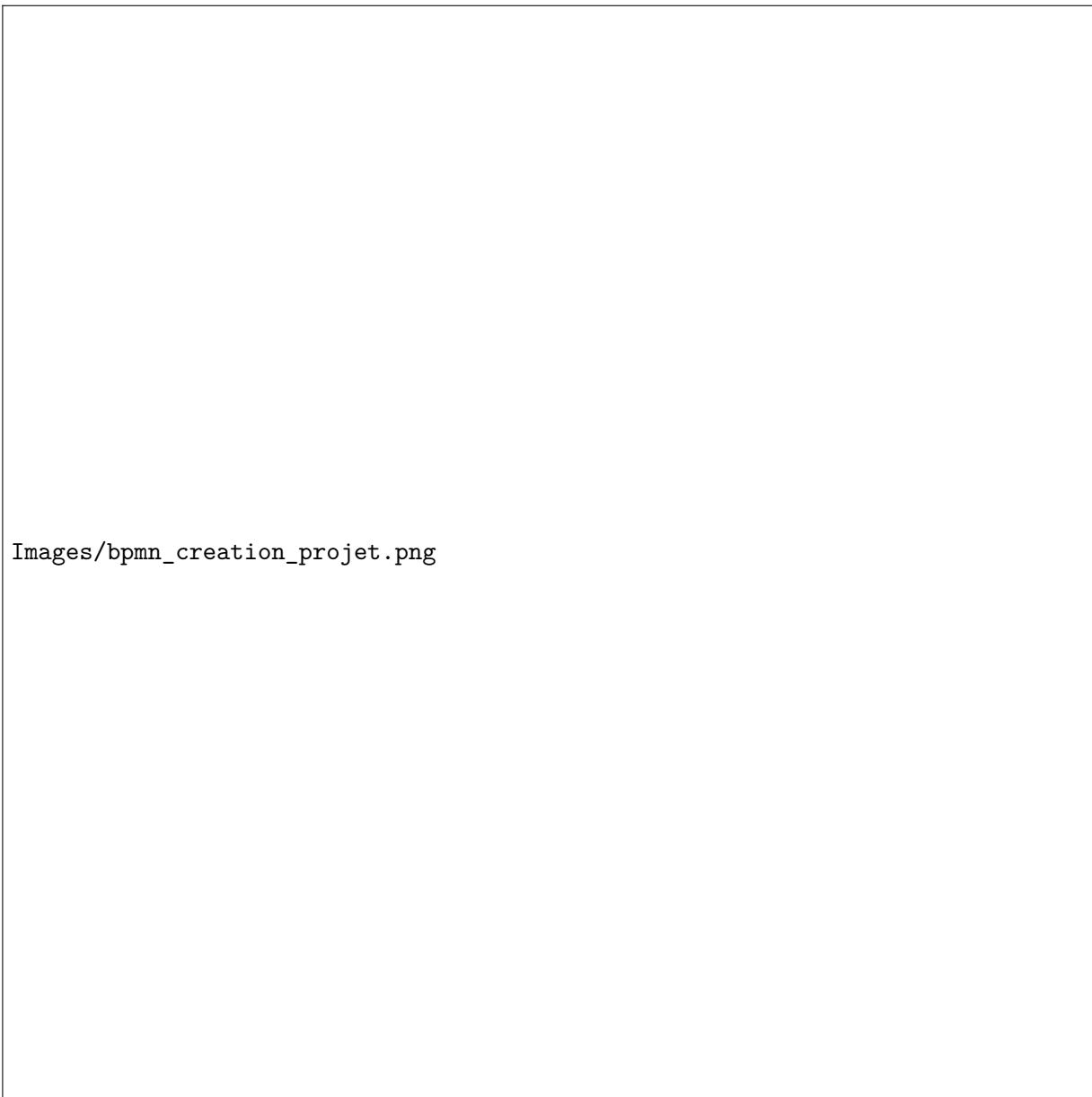
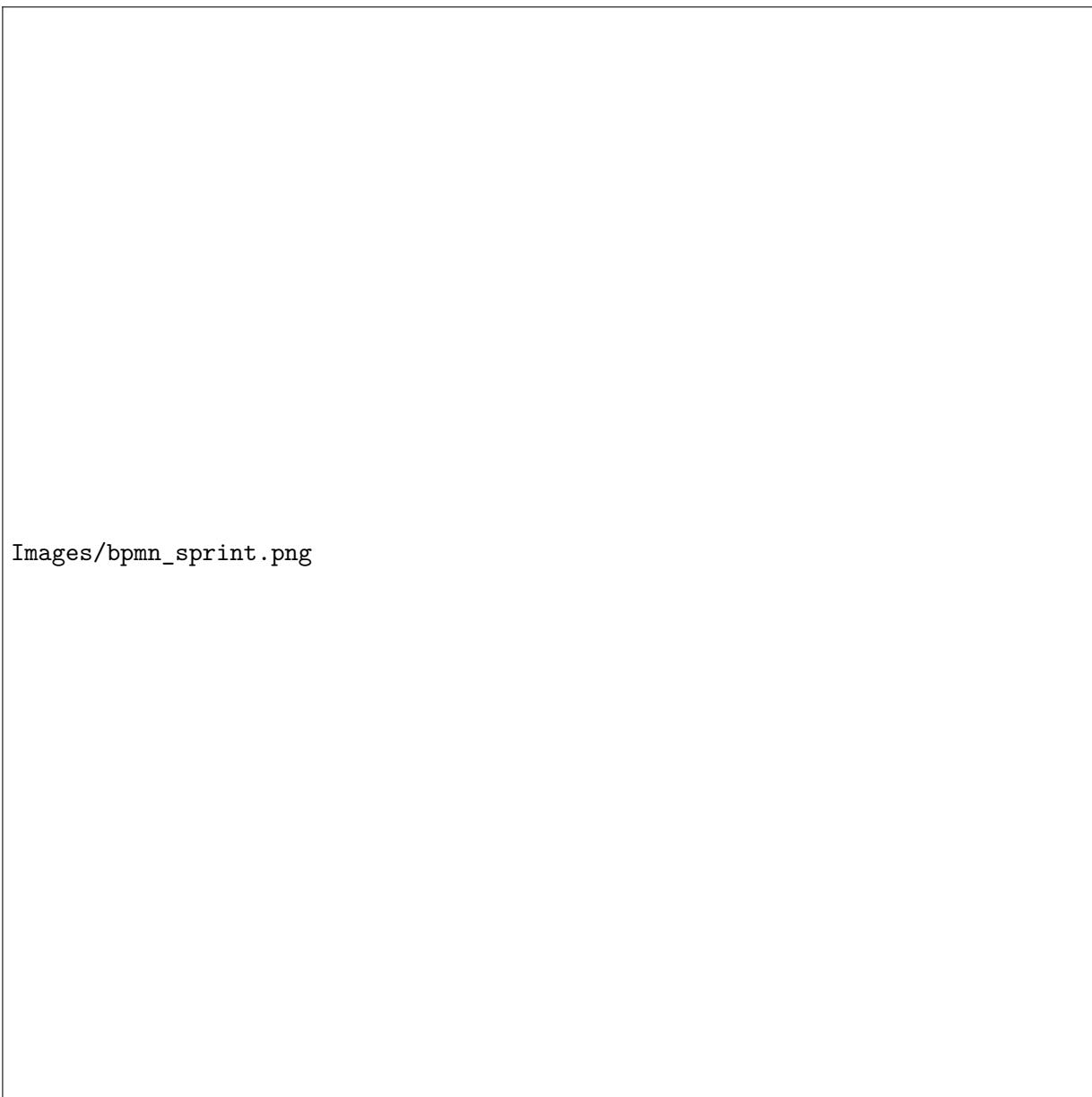


FIGURE C.1.1 – BPMN — Processus de création de projet



## C.2 Processus BPMN — Sprint Workflow



Images/bpmn\_sprint.png

FIGURE C.2.1 – BPMN — Processus de gestion du sprint

# ANNEXE D

## Annexe D — UML (Modélisation du Système)

### D.1 Diagramme de classes

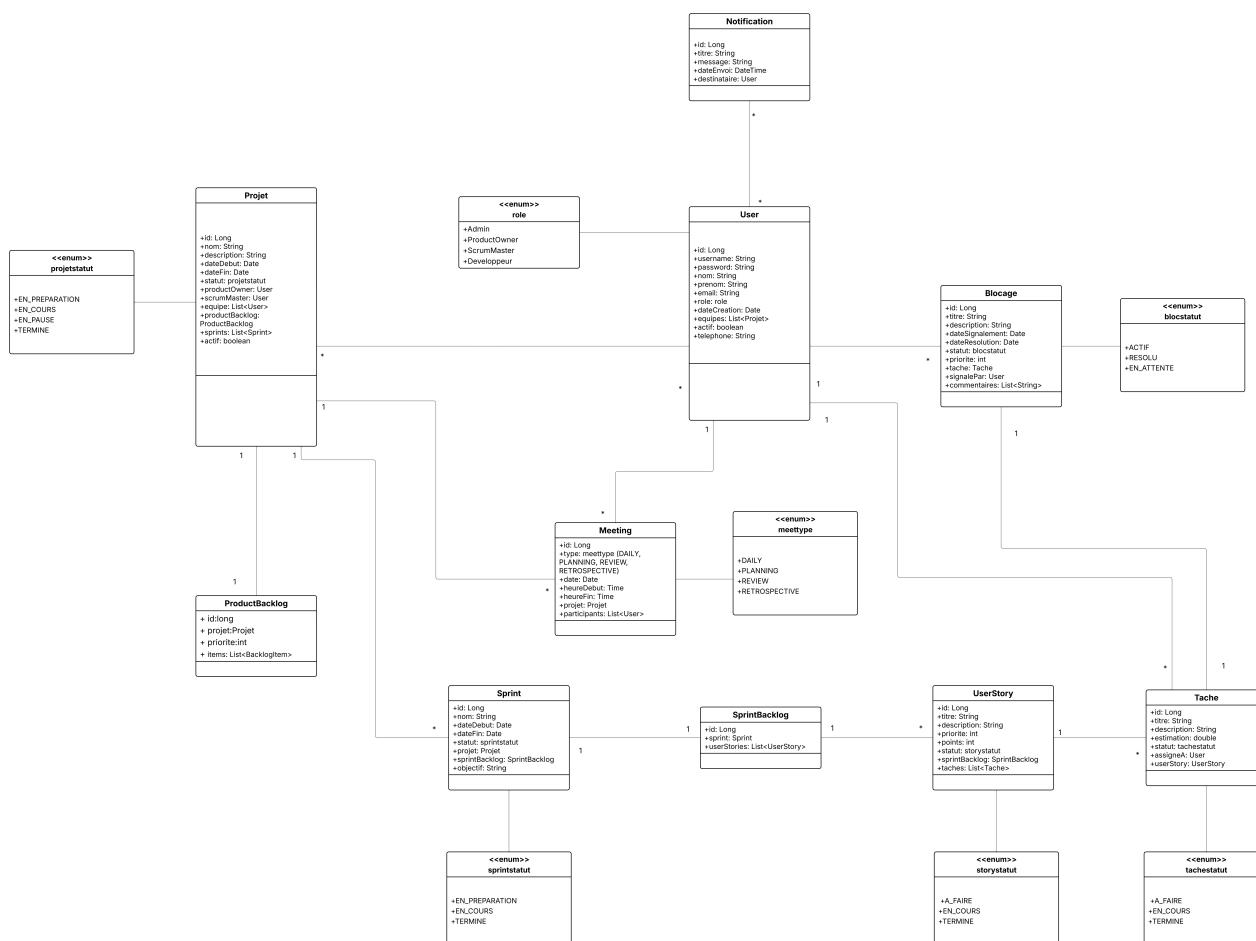
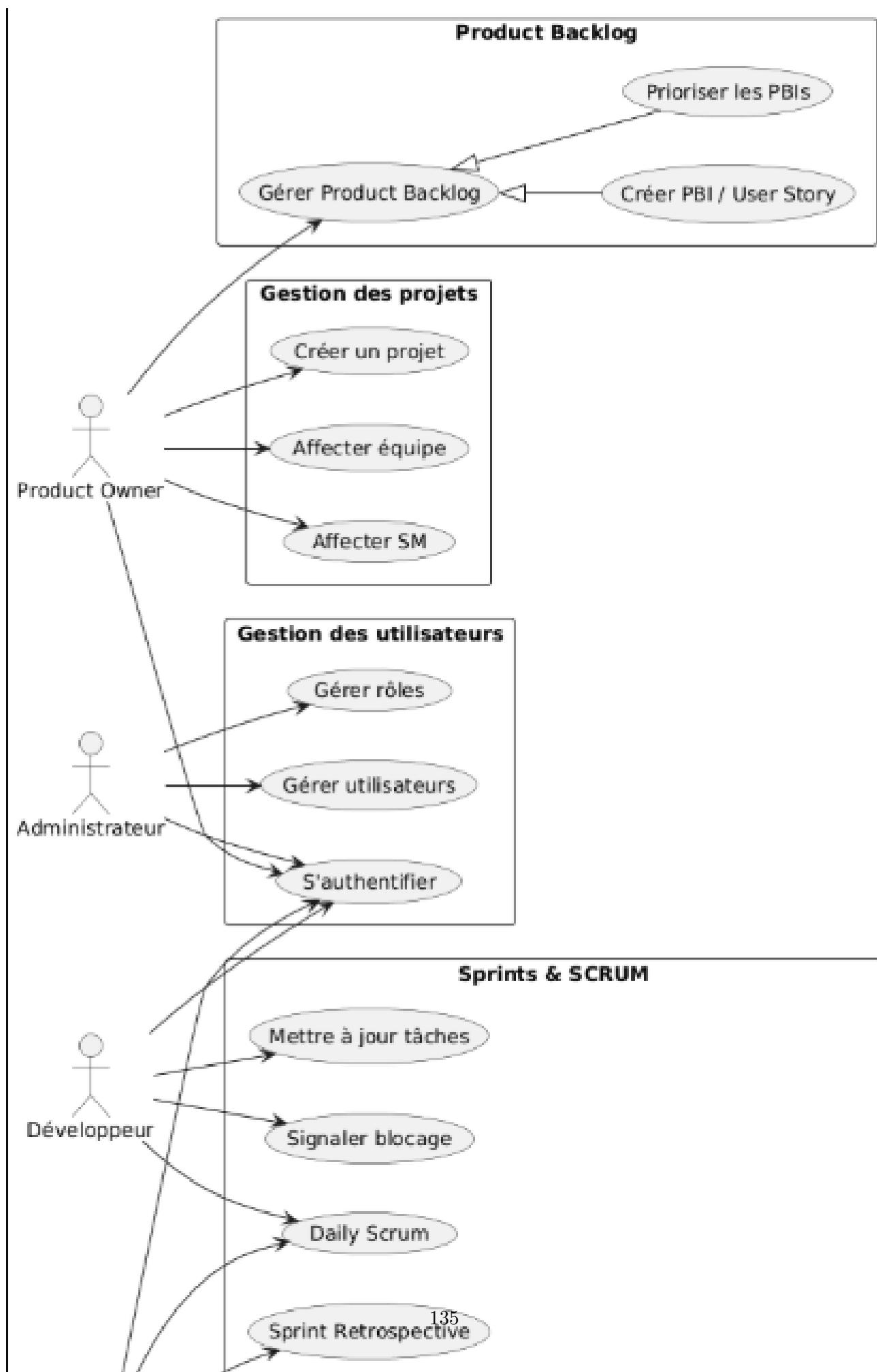


FIGURE D.1.1 – Diagramme de classes complet du système



## D.2 Diagramme de cas d'utilisation



### D.3 Diagrammes de séquence

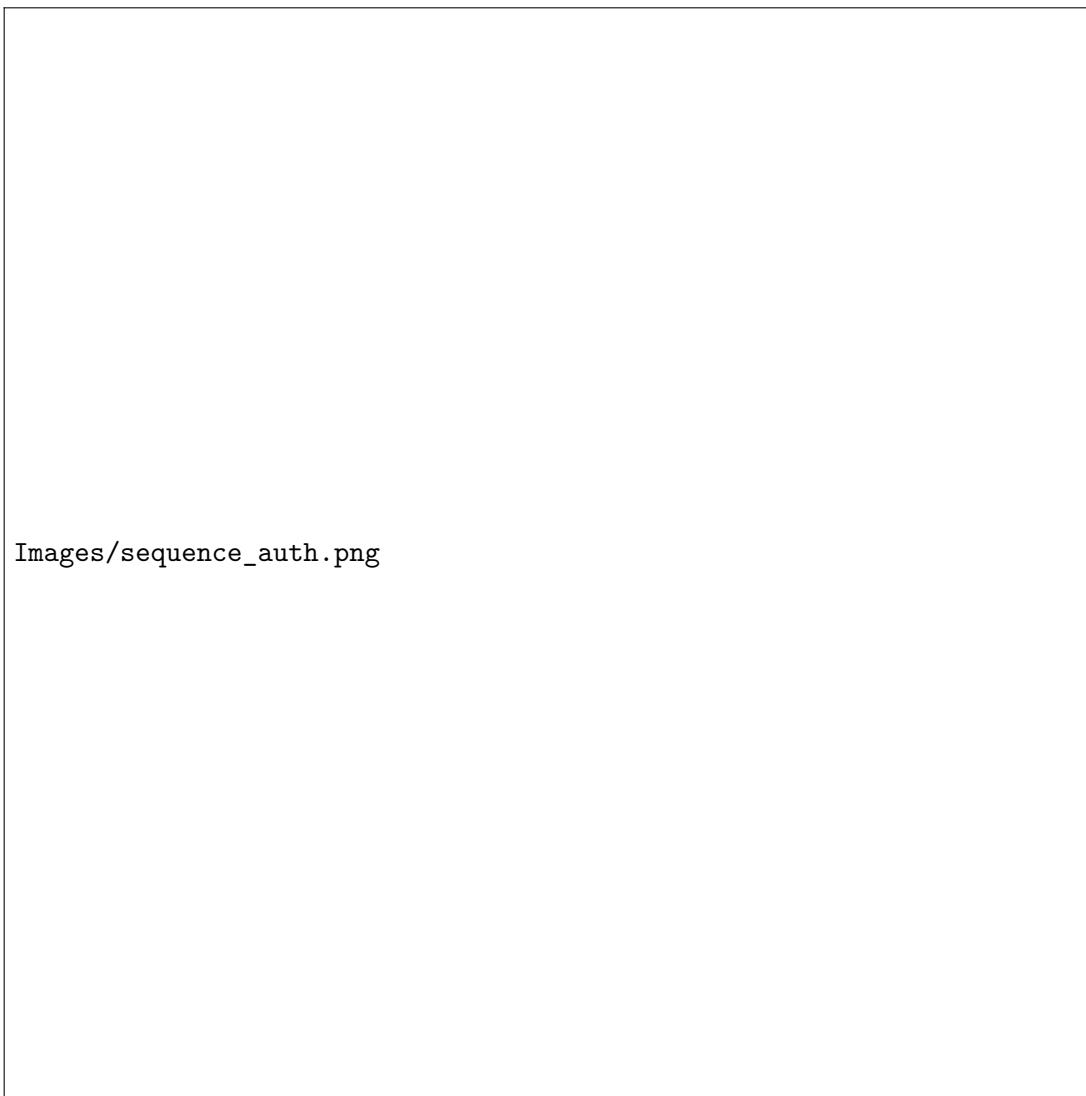


FIGURE D.3.1 – Diagramme de séquence — Authentification JWT



## ANNEXE E

# Annexe E — Déploiement Docker et Infrastructure

### E.1 Fichier docker-compose.yml

```
1  version: '3.8'
2  services:
3    backend:
4      build: ./backend
5      ports:
6        - "8080:8080"
7
8    frontend:
9      build: ./frontend
10     ports:
11       - "4200:80"
12
13   postgres:
14     image: postgres:15
15     environment:
16       POSTGRES_PASSWORD: admin
17     ports:
18       - "5432:5432"
```

Listing E.1.1 – Fichier docker-compose utilisé pour le déploiement local

## E.2 Architecture technique déployée



Images/docker\_architecture.png

FIGURE E.2.1 – Architecture technique déployée (Docker + services)