UTBM - Département Informatique

Projet – GL52

Semestre P2020

Sujet :

Implémentation d'un logiciel de gestion de projet suivant la méthodologie SCRUM

Anaïs JARNO
Yi Wen MIAO
Honghao XU
Shijia YU

utbm
université de technologie
Belfort-Montbéliard

TABLE DES MATIÈRES

1 INTRODUCTION

1.1 BUTS ET DESTINATAIRES DU DOCUMENT

Nous sommes des étudiants en deuxième année d'informatique à l'UTBM. Dans le cadre de l'UV de GL52 – Génie logiciel, nous avons à réaliser un projet, en suivant la méthodologie vue en cours, pour mettre en application les connaissances acquises durant les cours et les séances de travaux dirigés.

Le sujet de l'E-SCRUM nous a semblé intéressant, nous avons alors choisi de réaliser notre projet à ce propos.

Il s'agit de développer, réaliser une application permettant de supporter le processus de développement SCRUM, c'est-à-dire, avec les fonctionnalités de gestion des rôles, gestion du backlog, gestion des sprints et des tâches associées.

1.2 DÉFINITIONS ET ABRÉVIATIONS

Méthodologie SCRUM : approche qui reprend le cycle de développement itératif et incrémental en le structurant autour de ce que l'on appelle les sprints.

SPRINT : cycle de développement qui dure en général 2 à 4 semaines et termine par un produit apportant une valeur ajoutée au client.

BACKLOG DU PRODUIT : liste incomplète de tâches et de fonctionnalités à implémenter (USERS STORIES) répondant aux besoins du client et définies par le PRODUCT OWNER

BACKLOG DU SPRINT : sélection d'éléments, parmi les tâches du BACKLOG PRODUIT, traités durant le sprint. Cette liste définie par le SCRUM MASTER conjointement avec l'équipe de développement (la SCRUM TEAM) lors de la réunion de planification du sprint. Leur réalisation est effectuée par la SCRUM TEAM.

Diverses RÉUNIONS rythment le projet, comme la mêlée quotidienne, la planification du sprint (précédemment évoquée) ou encore la revue de sprint. Elles sont organisées par le SCRUM MASTER.

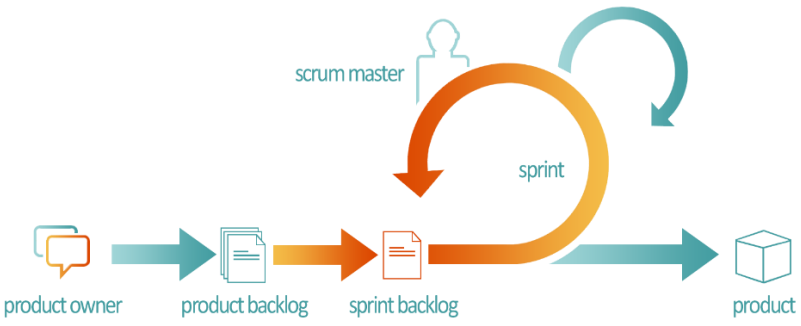


Figure 1 - Schéma résumé du fonctionnement de la méthode SCRUM

1.3 PRÉSENTATION GÉNÉRALE DU DOCUMENT

Ce rapport est organisé de la façon suivante :

La première partie est celle de l'introduction, où nous avons détaillé la présentation de ce document.

La seconde traite plus particulièrement de la description générale du sujet, avec son environnement et les caractéristiques du système.

La troisième décrit les besoins fonctionnels, les cas d'utilisations du système.

La quatrième partie spécifie les structures de données qui seront utilisées dans l'application.

La cinquième partie s'occupe de spécifier l'interfaçage externe, tant au niveau de la configuration requise au niveau matériel, que les besoins logiciels et la modélisation de l'IHM.

La sixième partie détaille les besoins en performance pour assurer le bon fonctionnement du système.

La septième partie regroupe les contraintes de développement, la sécurité et la fiabilité du système.

Enfin, on retrouve en huitième partie, les références, puis l'index et les annexes dans une dixième et dernière partie.

2 DESCRIPTION GÉNÉRALE

2.1 ENVIRONNEMENT, OU CONTEXTE DU SYSTÈME

Les interactions des divers utilisateurs avec le système ont été représentées dans le digramme de contexte suivant. On retrouve les acteurs ainsi que leur utilisation du système.

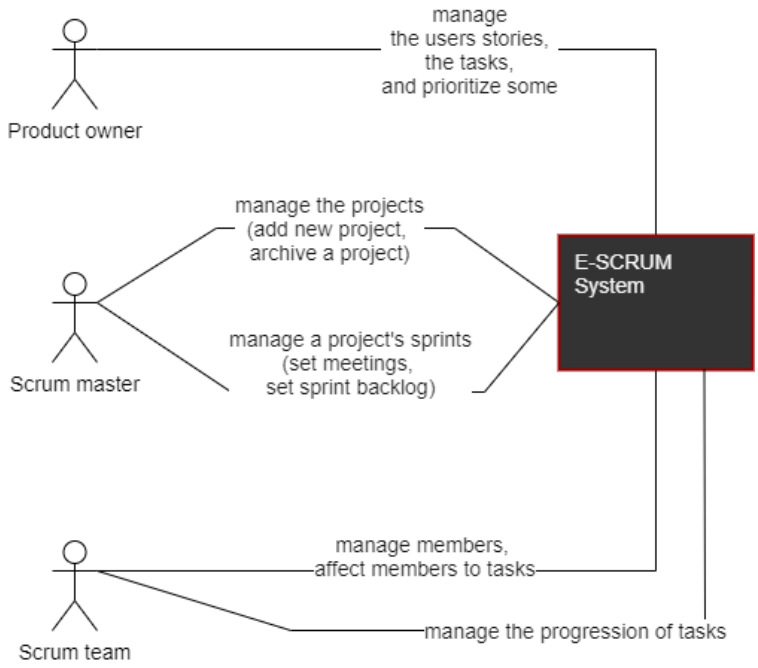


Figure 2 - Diagramme de contexte

Un **utilisateur** est défini, pour chaque projet, selon un de ces trois rôles : **PRODUCT OWNER, SCRUM MASTER, SCRUM TEAM.**

Le rôle de l'utilisateur définit la façon dont il va interagir avec le système.

Le **PRODUCT OWNER** se charge de créer, découper et de prioriser les tâches du BACKLOG PRODUIT. L'utilisateur ayant ce rôle utilise le système pour gérer le projet au niveau de ses tâches à faire et ne nécessite l'utilisation du système que pour cette gestion.

Le **SCRUM MASTER** s'occupe de la gestion des projets. Il crée de nouveaux projets lorsqu'on lui en commandite un, archive un projet lorsqu'il est fini et que le client (donc le PRODUCT OWNER considère que le projet est achevé) et gère les projets existants à travers une gestion des sprints. Il s'occupe de la création d'un SPRINT BACKLOG, et de l'organisation des différentes réunions qui rythment le sprint (comme la réunion de mêlée quotidienne, ou la réunion de planification de sprint...).

La **SCRUM TEAM**, ou plutôt les membres de la SCRUM TEAM, gèrent le développement et l'avancée du projet. Ainsi, ils s'occupent de gérer les membres participant au développement, l'affectation aux tâches et leur progression.

2.2 CARACTÉRISTIQUES DES UTILISATEURS

Chacun de ces utilisateurs a les mêmes accès sur l'application. La différence se jouant sur leur utilisation de l'application.

Le PRODUCT OWNER se contente d'utiliser les fonctionnalités de gestion des tâches à faire du projet.

Il n'est donc par un utilisateur régulier du système, mais plutôt occasionnel. Il l'utilisera pour chaque fin/début de sprint ou en cas de demande spécifique de la part de l'équipe. Il peut nécessiter une formation pour savoir utiliser correctement l'application.

Le SCRUM MASTER utilise le système de manière plus ou moins fréquente, il sert d'intermédiaire entre le PRODUCT OWNER et la SCRUM TEAM si besoin est, et il guide l'équipe de développement (la SCRUM TEAM) tout au long de la réalisation du projet. Il doit donc connaître l'application de manière plutôt détaillée.

Enfin, tout membre de la SCRUM TEAM est un utilisateur fréquent du système car représente l'avancement de son travail en temps (quasi-) réel. Il est coutumier de ce genre de logiciel étant donnée son poste.

2.3 CONTRAINTES PRINCIPALES DE DÉVELOPPEMENT

Utilisation d'un langage orienté objet

Développement d'une plateforme web avec un frontend, un backend et une base de données

- Technologie : libre
- Temps : jusqu'à mi-juin (2 mois)
- Budget : 0€, pas de budget, utilisation d'outils gratuits

3 BESOINS FONCTIONNELS

Le diagramme de cas d'utilisation suivant fournit un aperçu global du comportement du système. Il illustre le comportement du système qui est visible à l'utilisateur final, et qui est externe au système. Dans l'image, nous voyons plusieurs cas d'utilisation liés à l'opération, le contrôle et la supervision du système.

On retrouve les actions disponibles à chaque type d'utilisateur vis-à-vis du système, comme un utilisateur de type SCRUM TEAM peut éditer une tâche en lui assignant un membre de l'équipe, changer son statut et confirmer les invitations aux réunions.

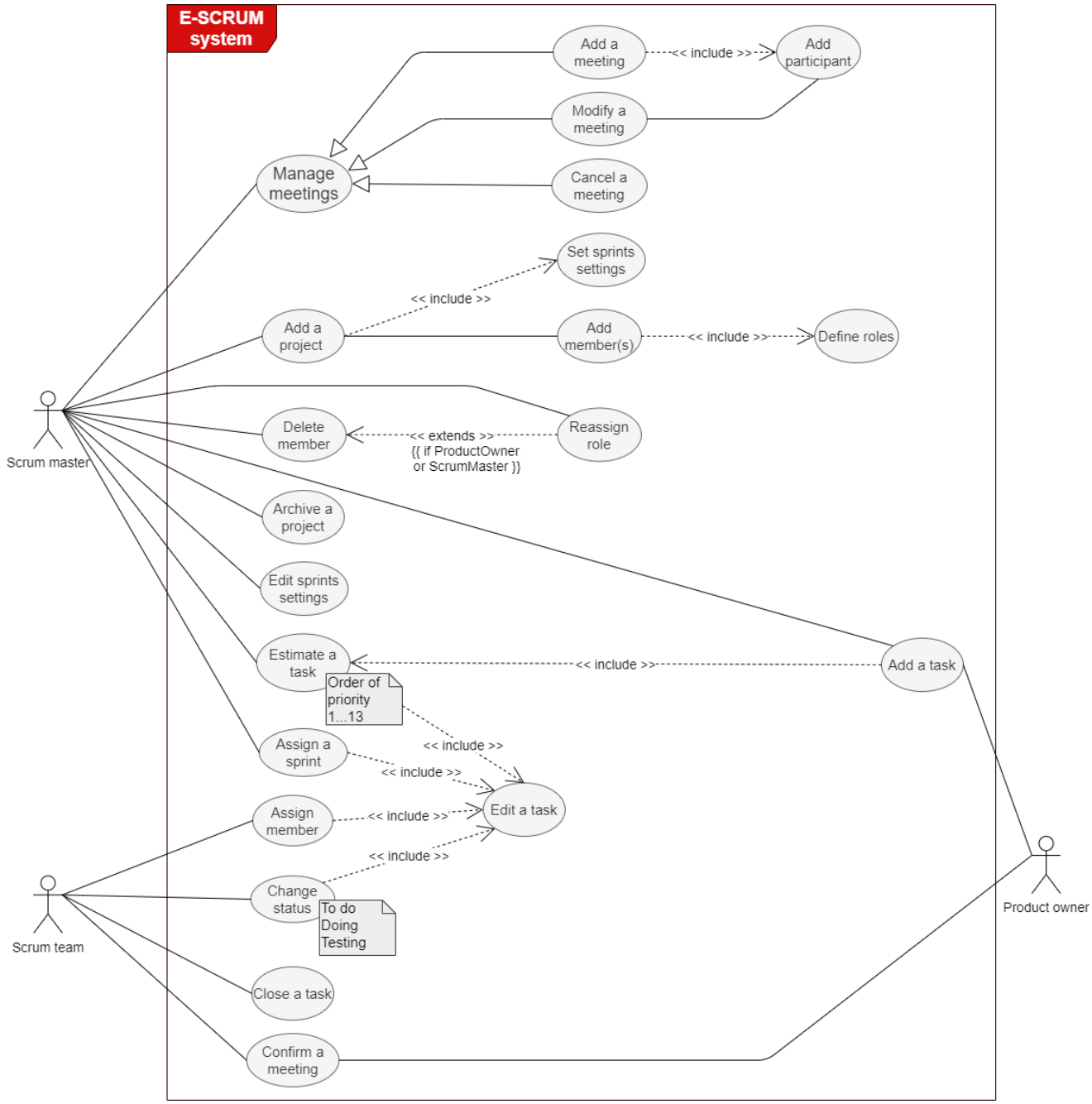


Figure 3 - Diagramme de cas d'utilisation

Anaïs JARNO - Yi Wen MIAO - Honghao XU - Shijia YU

On retrouvera en annexe les scénarios, les descriptions textuelles, ainsi que les diagrammes de séquence pour les deux cas d'utilisation suivants : **ajouter un projet** (*add a project*) et **ajouter une tâche** (*add a task*).

Dans les scénarios, on retrouve les séquences d'étapes décrivant les interactions entre l'utilisateur et le système pour réaliser leur objectif. Ceux-ci seront accompagnés de descriptions textuelles précisant le contexte, c'est-à-dire les données nécessaires, les sorties système etc.

Enfin, les diagrammes de séquences offrent une représentation graphique de la chronologie des échanges de messages entre les acteurs et le système. Ils spécifient ainsi les interactions entre les objets selon un point de vue temporel.

4 SPÉCIFICATION DES STRUCTURES DE DONNÉES

Les données que nous allons manipuler vont être stockées dans une base de données et utilisées dans toutes notre application. Plusieurs tables vont donc être créées. Ces tables seront reliées entre elles sous forme d'associations, d'héritage. Pour présenter les structures de données que l'on souhaite modéliser, nous avons créé un diagramme de classes où l'on peut lire les différents types de données pour chaque attribut.

Ainsi, le diagramme ci-dessous montre la structure du système sous la forme d'un ensemble de classes et de relations entre ces classes. Il spécifie la structure et les liens entre les objets qui composent notre système.

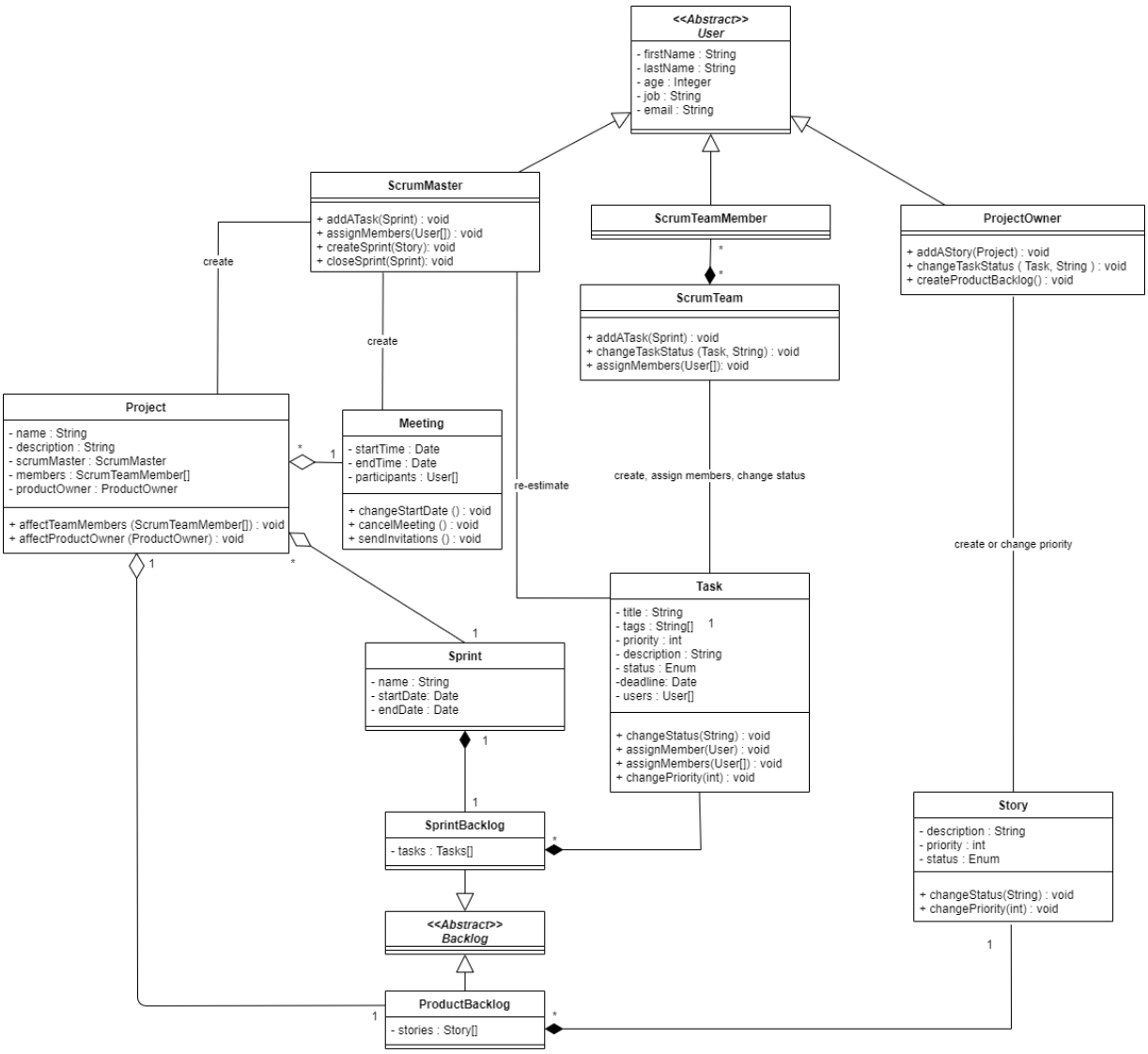


Figure 4 - Diagramme de classes

Anaïs JARNO - Yi Wen MIAO -
Honghao XU - Shijia YU

5 SPÉCIFICATIONS DES INTERFACES EXTERNES

5.1 INTERFACE MATÉRIEL/LOGICIEL

5.1.1 Configuration minimale

Ordinateur serveur

PC 1Gz et 512 Mo de mémoire vive. Une très faible taille de disque dur est nécessaire étant donné le peu de données qui vont être traitées par l'application. Une partition de 1Go paraît amplement suffisante.

Ordinateurs client

PC 450MHz 128 Mo de mémoire vive. Pas d'espace disque requis. Accès à internet, carte réseau.

5.1.2 Périphériques

Sur le serveur et les clients, seuls un clavier et une souris sont requis. Les clients peuvent même utiliser un simple téléphone ayant un accès à internet.

5.1.3 Protocole d'échange

L'application étant conçue sous forme de site web, le protocole utilisé sera TCP/IP.

5.1.4 Type de liaison

Tout accès à internet est envisageable : Ethernet, Wifi…

5.2 INTERFACE LOGICIEL/LOGICIEL

Il faudra pouvoir faire tourner l'application en continu : utilisation de services externes pour héberger et déployer l'application.

Sur les clients, le système devra fonctionner peu importe le système d'exploitation (Windows, Linux, Android, iOs ou MacOs) et nécessitera un navigateur compatible avec les recommandations du W3C (exclure Internet Explorer en version inférieur à 7).

5.3 INTERFACE HOMME/LOGICIEL

Un tableau résumant les accès à chaque page et fonctionnalité de l'application a été réalisé pour permettre une visualisation claire, rapide et simplifiée des différences et des droits pour tous les membres utilisant l'application.

Rôle Fonctionnalités	SCRUM MASTER	MEMBRE DE LA SCRUM TEAM	PRODUCT OWNER
Page de connexion			
trois boutons (scrum master, scrum team, product owner) *ce sera suffisant pour le moment*	OUI	OUI	OUI
Dashboard			
affichage de la liste des projets	OUI	OUI	OUI
affichage des meetings à venir	OUI	OUI	OUI
Onglet de gestion des projets			
ajout de projet	OUI		
modifications des membres d'un projet	OUI		
archivage d'un projet	OUI		
Visualisation d'un projet			
Suivi des sprints d'un projet			
création de sprint	OUI		
clôture d'un sprint	OUI		
créer de nouvelles tâches pour un sprint	OUI	OUI	
modifier l'estimation (la priorité) d'une tâche	OUI	OUI	
visualisation des tâches	OUI	OUI	OUI
affectation d'un membre (ou de soi-même) à une tâche		OUI	
modification de l'état de la tâche		OUI	
Gestion du product backlog			
visualiser le product backlog et son avancement	OUI	OUI	OUI
ajouter des stories			OUI
estimer la priorité d'une story			OUI
Onglet de gestion des meetings			
visualiser les meetings programmés	OUI	OUI	OUI
ajouter un meeting	OUI		
voir les invitations à des meetings et répondre		OUI	OUI

Aussi, diverses maquettes de l'application ont été réalisées afin d'offrir un aperçu de ce qui est attendu de l'application finale. D'autant plus que l'ergonomie d'une application est un point important, à ne pas négliger lors de la réalisation d'une application.

Par exemple, ci-dessous, on retrouve la maquette d'une page de connexion simplifiée permettant à l'utilisateur de se connecter selon son rôle. A côté, se trouve la page réelle de connexion à l'application.

Figure 5 - Maquette de la page de connexion

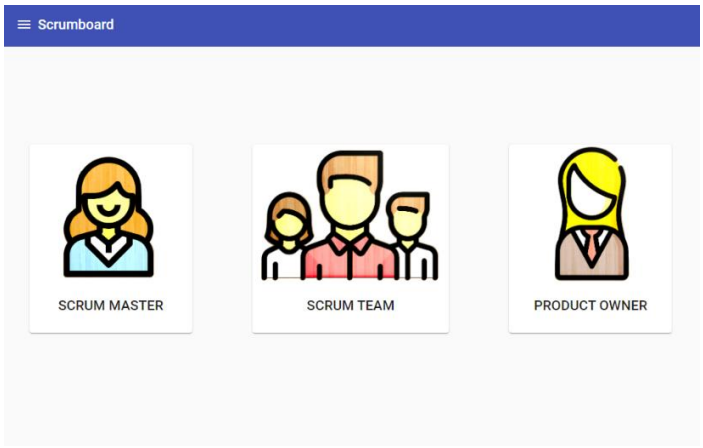


Figure 6 - Page de connexion réelle

Lorsque l'utilisateur se connecte, la structure de l'application change légèrement. L'utilisateur retrouvera sur sa gauche un menu latéral lui permettant de naviguer entre les grandes fonctionnalités de l'application, un bouton de déconnexion en haut à droit de la barre de navigation et enfin, au centre, il retrouvera son espace d'accueil avec les tâches et réunions approchant.

De cette façon, les maquettes des différentes pages de l'application SCRUM sont disponibles en annexe de ce document.

6 BESOINS EN PERFORMANCE

Afin de pouvoir être exploitable, l'application doit répondre à certaines exigences :

- Gérer les connexions de multiples utilisateurs, la connexion simultanée d'un grand nombre d'utilisateurs doit être possible
- Être efficace en ayant un temps de chargement des pages raisonnable, temps de réponse minimal lors de la navigation d'une page à l'autre, imperceptible pour l'utilisateur

Pour répondre à ces besoins, il sera préférable de rechercher à optimiser le schéma de la base de données, et les requêtes pour accéder aux données. Cela permettra au système de fonctionner correctement et rapidement même si la base de données contient beaucoup d'informations.

7 CONTRAINTES DE DÉVELOPPEMENT

7.1 FIABILITÉ ET TOLÉRANCE AUX FAUTES

L'utilisateur devra toujours avoir la possibilité de modifier ce qu'il vient d'entrer de manière à corriger des données éventuellement erronées.

Des contrôles automatiques devront également être mis en place (contrôle de la validité d'une adresse mail, éviter les noms vides…).

7.2 COMPORTEMENT DU SYSTÈME DANS DES SITUATIONS ANORMALES

Le système devra pouvoir réagir de manière lisible aux erreurs de manipulation de l'utilisateur : toute manipulation incorrecte de l'application, ne respectant pas les normes définies (format de date, longueur maximale de chaînes de caractères, etc.) devra générer un message d'erreur explicite permettant à l'utilisateur de comprendre pourquoi l'erreur est survenue et de la corriger aisément.

7.3 SÉCURITÉ

Le système devra gérer les droits des utilisateurs de manière à restreindre l'accès de certaines pages à certaines personnes, selon leur rôle au sein d'un projet.

7.4 STANDARDS, MÉTHODES, OUTILS ET LANGAGES DE DÉVELOPPEMENT

Le projet d'implémentation de la méthodologie SCRUM est une application web, développée pour la partie Frontend en Angular, pour le Backend en Node.js avec une base de données MongoDB. Pour la première itération de l'application, seule la partie Frontend sera fournie.

En effet, en développement Frontend, une tâche courante consiste à simuler un Backend RESTful pour fournir les données à l'application Frontend et s'assurer que tout fonctionne comme prévu. Comme le développement d'un serveur Backend complet prendrait un certain temps, nous avons alors décidé qu'il n'y aurait pas d'implémentation de Backend dans ce projet.

Pour cependant avoir un Frontend parfaitement fonctionnel, le principe récupération des données à partir de services lançant des requêtes HTTP à l'API du Backend a été conservé. Pour ce faire, nous avons mis en place un serveur comportant une base de données factice à l'aide de *JSON Server*[1].

On retrouvera en annexe un schéma d'architecture résumant le fonctionnement provisoire fournit et le modèle vers lequel il devra aller.

[1] Module npm , qui fournit un serveur Express reproduisant le comportement d'une API REST

8 RÉFÉRENCES

Les seules sources utilisées dans l'élaboration de ce dossier de spécifications sont les cours de GL52 – Génie logiciel, dispensés à l'UTBM – Université de Technologie de Belfort-Montbéliard.

Source pour la Figure 1 - Schéma résumé du fonctionnement de la méthode SCRUM
https://www.netresearch.de/blog/die-einfuehrung-von-scrum-bei-netresearch-unser-rueckblick/

ANNEXES

ANNEXE 1 : SCÉNARIOS POUR DEUX CAS D'UTILISATION

Cas d'utilisation : **Ajouter une réunion** ***(Add a meeting)***

Scénario principal :

1. Le SCRUM MASTER s'authentifie dans le système et choisit l'ajout de réunion.
2. Le système lui demande la date de réunion.
3. Le SCRUM MASTER indique la date et l'heure de début/fin de la réunion.
4. La date est réservée et le système donne la liste des projets courants pour la sélection du projet lié à la réunion.
5. Le SCRUM MASTER choisit un projet.
6. Le système enregistre le projet pour la réunion saisie, transmet la liste des personnes participants au projet et attend la sélection d'invités.
7. Le SCRUM MASTER coche les participants à la réunion.
8. Le système enregistre les participants à la réunion et leur envoie une invitation à confirmer.

Cas particulier :

4a. La date est déjà occupée par une autre réunion, le système invite le SCRUM MASTER à sélectionner une autre date. Retour à l'étape 3.

Cas d'utilisation : **Ajouter une tâche** ***(Add a task)***

Scénario principal :

1. L'utilisateur s'authentifie, sélectionne un projet et choisit l'ajout d'une tâche.
2. Le système lui demande de saisir un nom de tâche et un ordre de priorité.
3. L'utilisateur entre le nom de la nouvelle tâche et choisit la priorité.
4. La tâche est créée, le système confirme la création de la tâche à l'utilisateur.

Cas particulier :

4a. Une tâche en cours porte déjà ce nom, le système invite l'utilisateur à saisir un autre nom. Retour à l'étape 3.

ANNEXE 2 : DESCRIPTION TEXTUELLE DE DEUX CAS D'UTILISATION

Fonction : **Ajouter une réunion** ***(Add a meeting)***

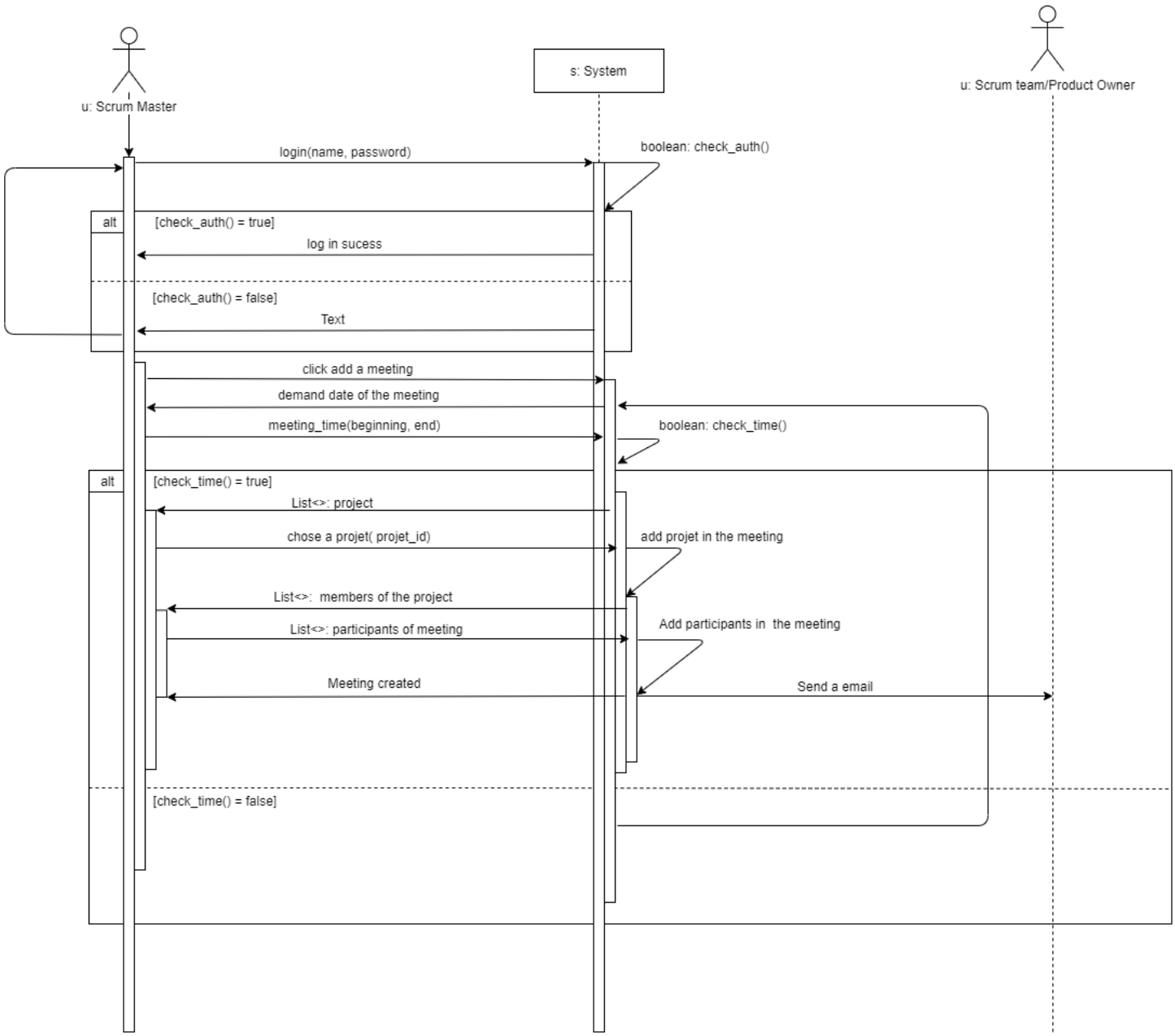
Description Le SCRUM MASTER crée une nouvelle réunion au projet et envoie des invitations aux participants.
Acteurs concernés SCRUM MASTER, SCRUM TEAM, PRODUCT OWNER
Entrées Projet, date et heure de début/fin de la réunion, liste des participants
Sorties Nouvelle réunion, invitations à confirmer aux les participants
Besoins Liste des projets de l'utilisateur et liste de participants du projet sélectionné
Pré-condition Connexion SCRUM MASTER réussie
Post-condition La réunion est enregistrée dans la base de données. Les participants reçoivent un e-mail de confirmation.

Fonction : **Ajouter une tâche** ***(Add a task)***

Description Le PRODUCT OWNER crée une nouvelle tâche et lui assigne un ordre de priorité.
Acteurs concernés PRODUCT OWNER
Entrées Nom de tâche, ordre de priorité
Sorties Nouvelle tâche
Besoins Projet concerné
Pré-condition Connexion PRODUCT OWNER réussie
Post-condition La tâche créée est enregistrée dans la base de données.

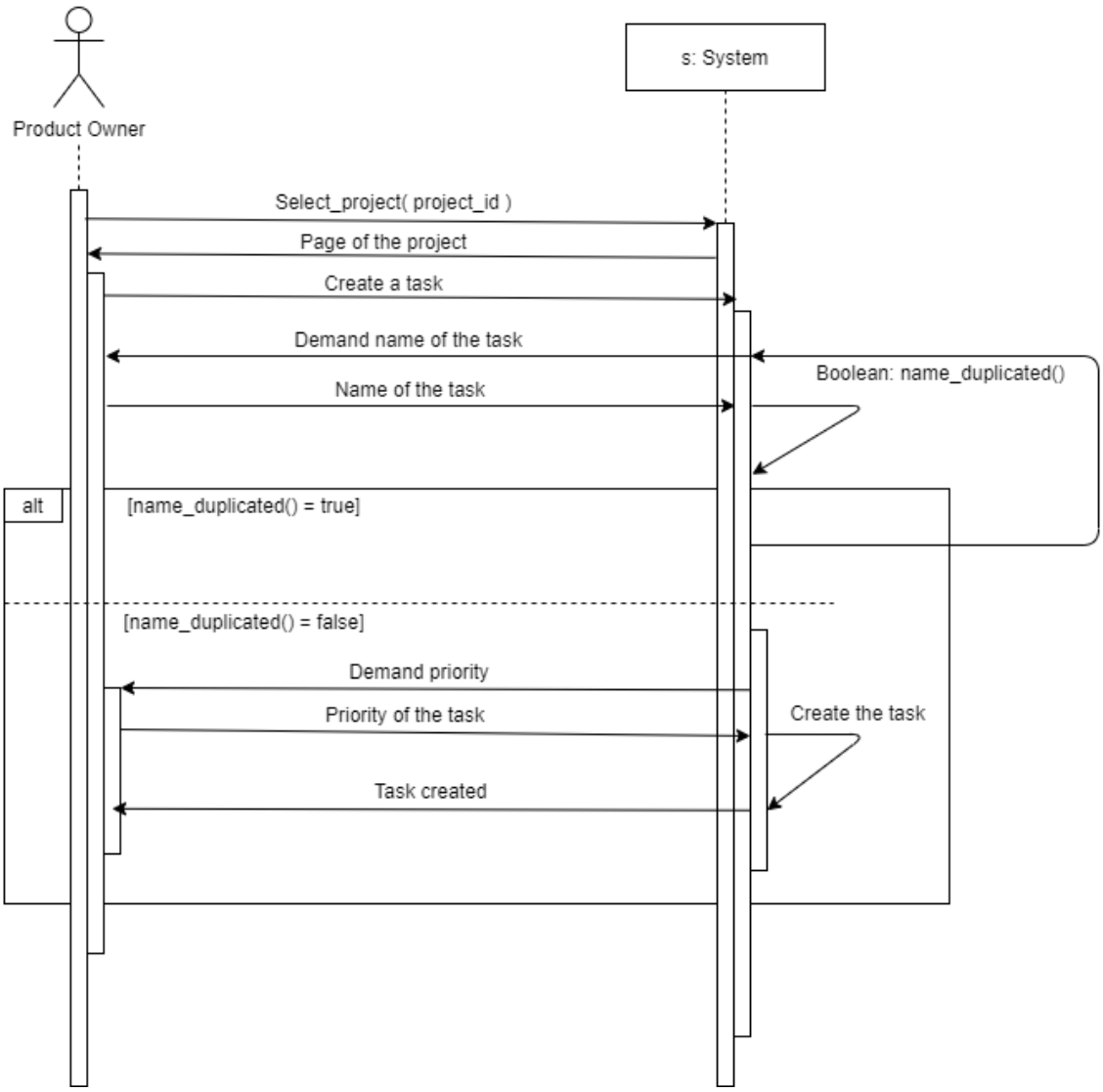
ANNEXE 3 : DIAGRAMME DE SÉQUENCE DE DEUX CAS D'UTILISATION

Sequence diagram *(add a meeting)*



Anaïs JARNO - Yi Wen MIAO -
Honghao XU - Shijia YU

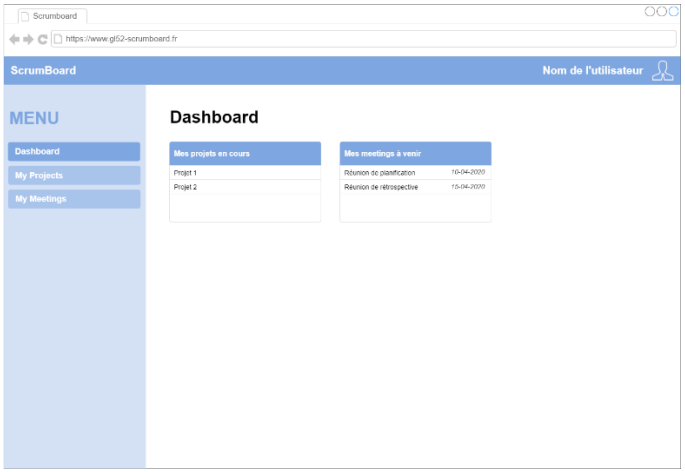
Sequence diagram *(add a task)*



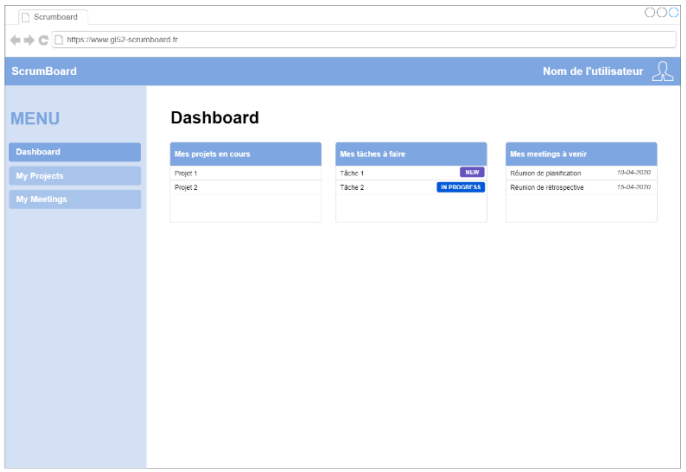
ANNEXE 4 : MAQUETTAGE DE L'APPLICATION

Maquette de la page d'accueil

pour le SCRUM MASTER et le PRODUCT OWNER à gauche



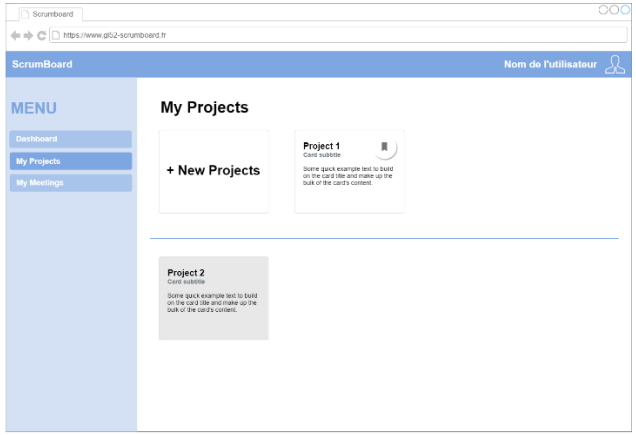
pour la SCRUM TEAM à droite

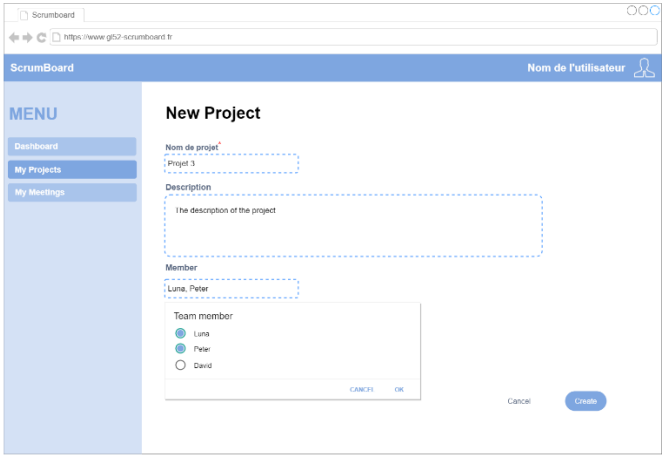


Anaïs JARNO - Yi Wen MIAO -
Honghao XU - Shijia YU

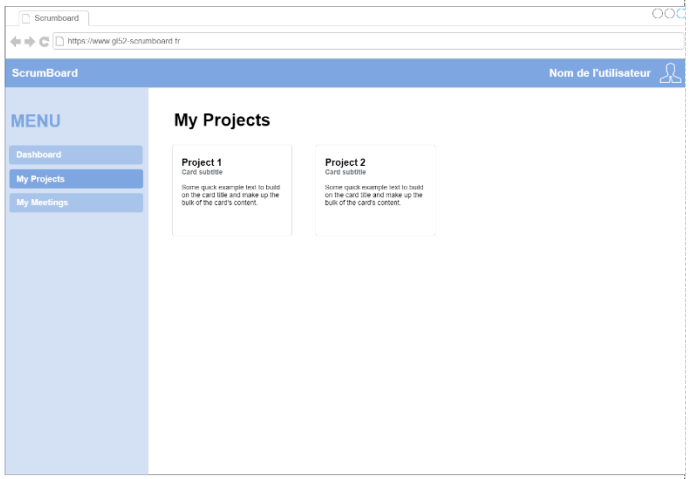
Maquette des pages de gestion de projets

pour le SCRUM MASTER

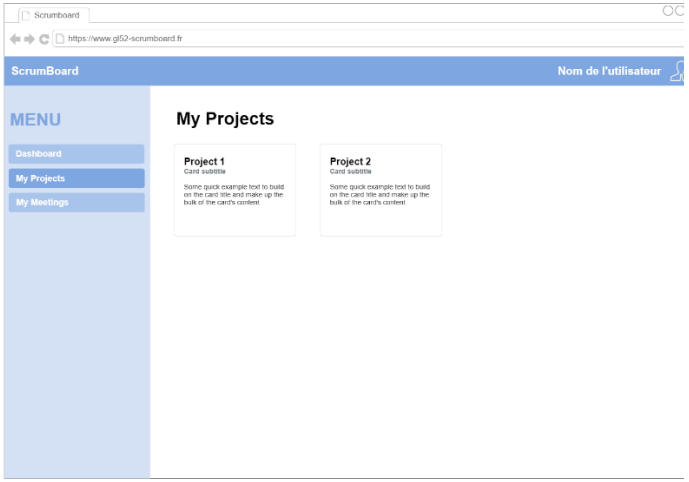




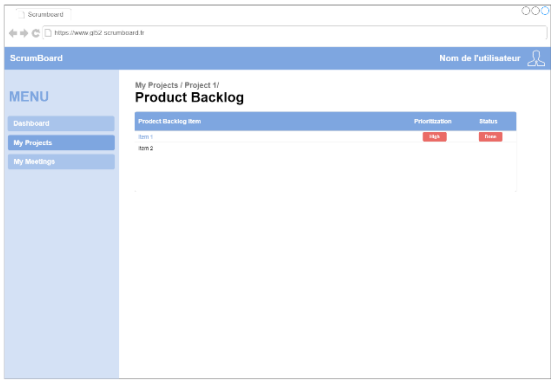
pour la SCRUM TEAM

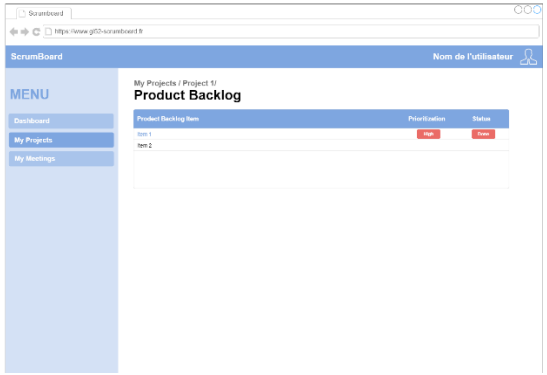


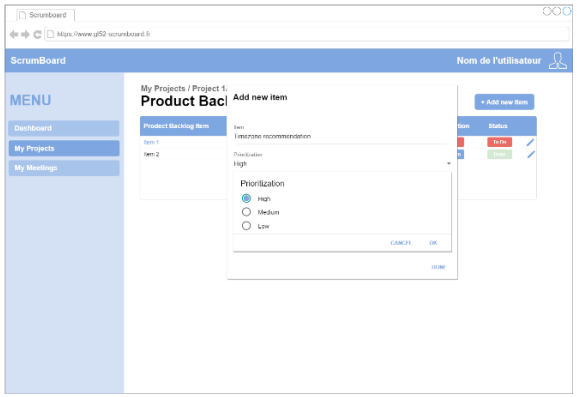
pour le PRODUCT OWNER

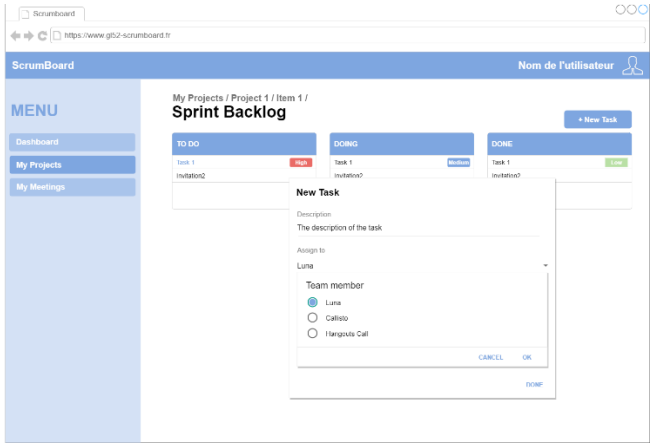


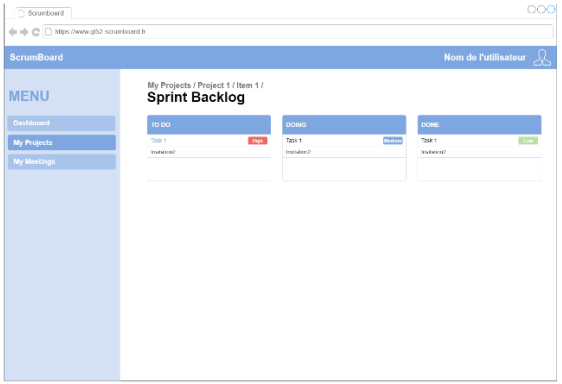
Anaïs JARNO - Yi Wen MIAO -
Honghao XU - Shijia YU

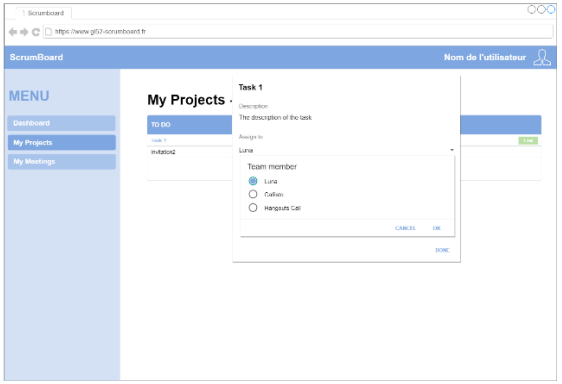








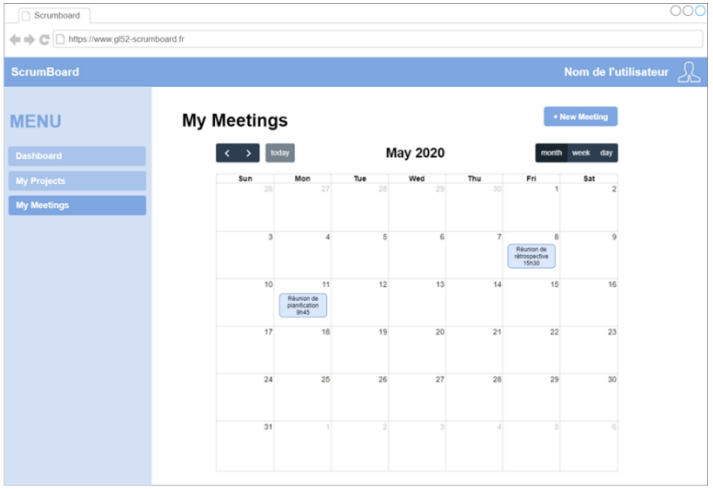


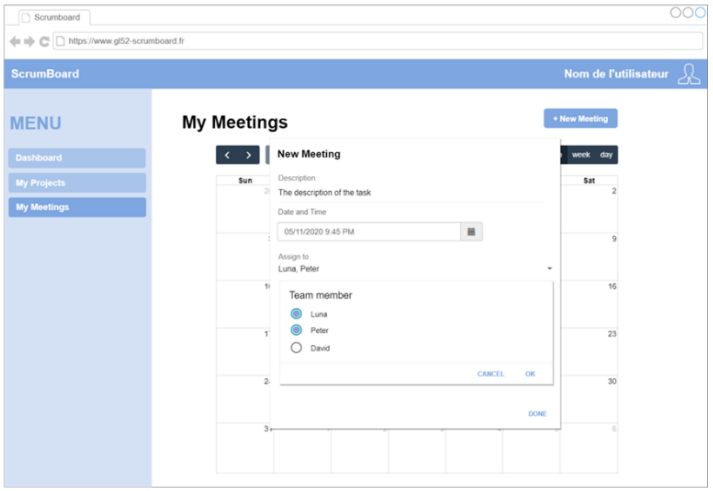


Anaïs JARNO - Yi Wen MIAO -
Honghao XU - Shijia YU

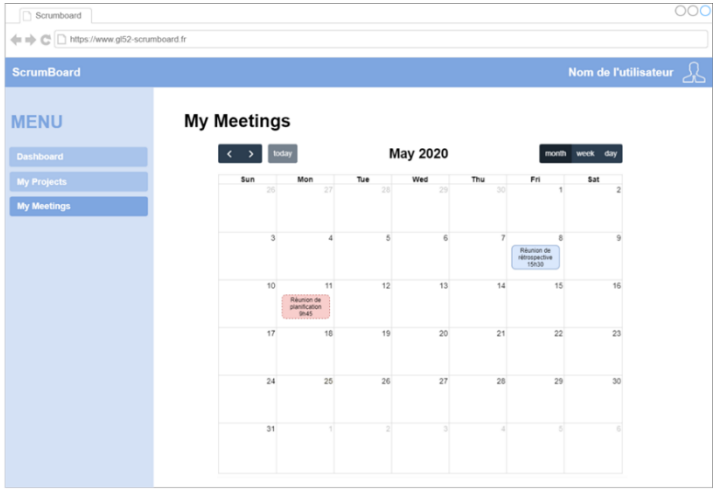
Maquette de la page de réunions

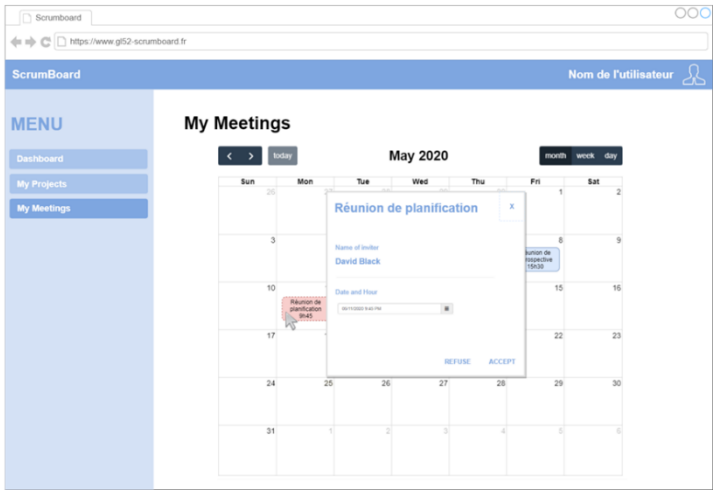
pour le SCRUM MASTER à gauche





pour la SCRUM TEAM et le PRODUCT OWNER à droite

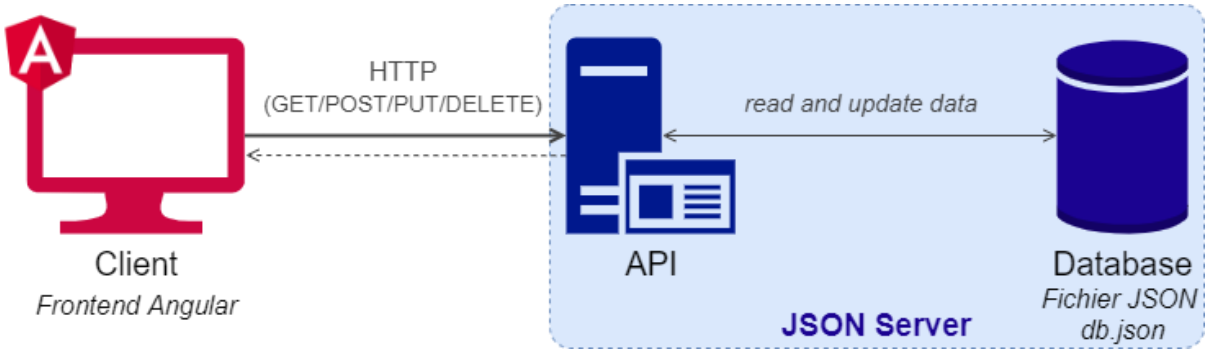




Anaïs JARNO - Yi Wen MIAO -
Honghao XU - Shijia YU

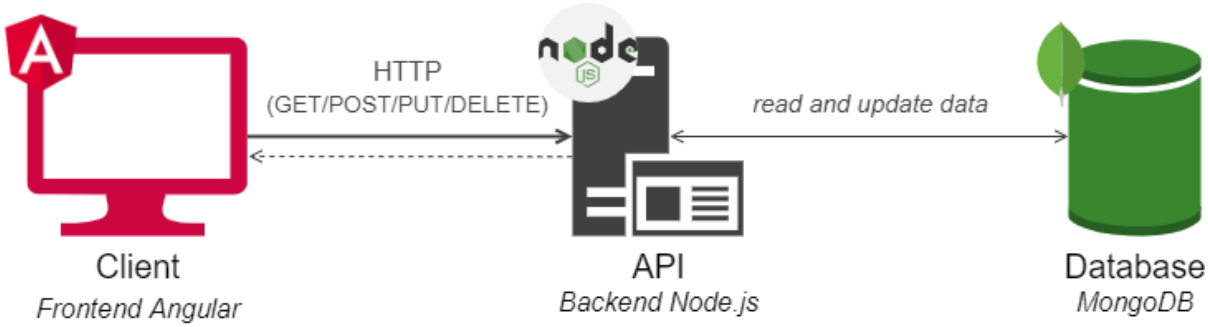
ANNEXE 5 : SCHÉMA D'ARCHITECTURE DE L'APPLICATION

Pour la première itération de l'application, on considérera l'architecture suivante :



Ainsi, la principale fonctionnalité sera le Frontend, et le Backend sera simulé grâce à JSON Server.

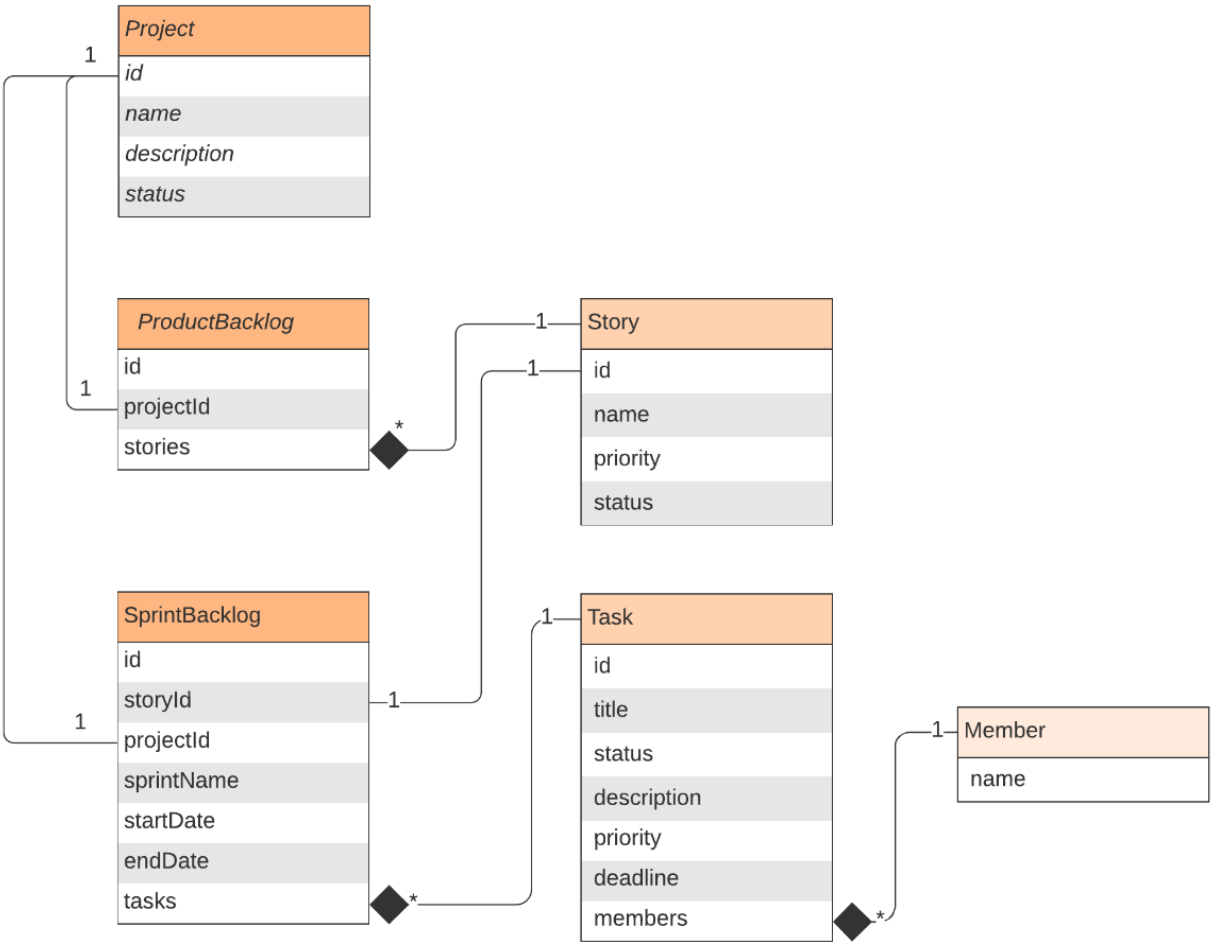
Ensuite, le modèle architectural vers lequel l'application devra tendre est le suivant :



Avec chacune de ces trois grandes parties développées entièrement.

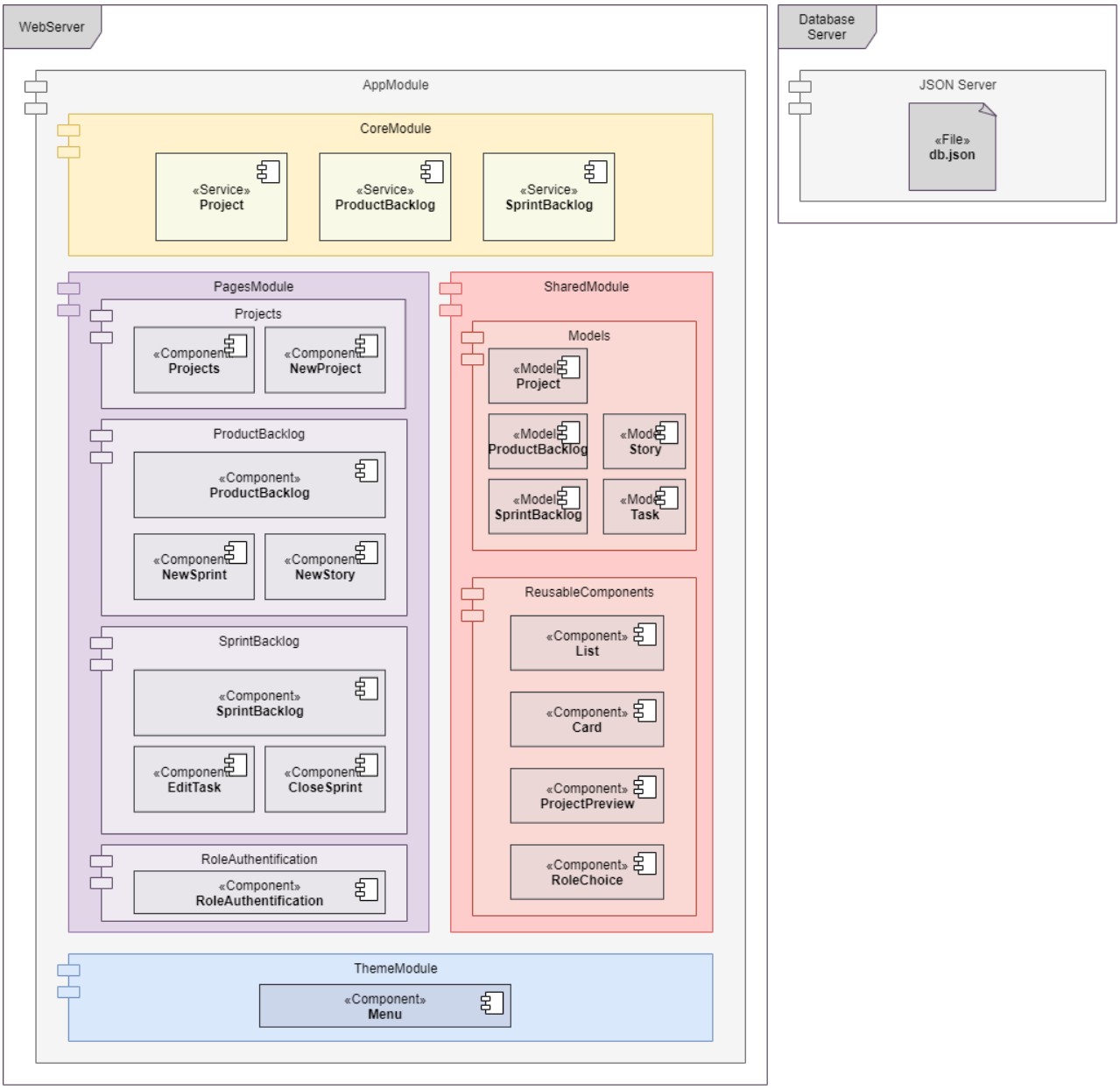
C'est-à-dire, le Backend Node.js fonctionnant en tant qu'API REST complet, et communicant avec notre Base de données MongoDB (il est à noter qu'une base de données de sécurité, ou même une base de données séparant les données d'authentification et les données métiers est à envisager).

ANNEXE 6 : DIAGRAMME D'ARCHITECTURE DE LA BASE DE DONNÉES ACTUELLE



ANNEXE 7 : DIAGRAMMES DE COMPOSANTS - ARCHITECTURE DE L'APPLICATION

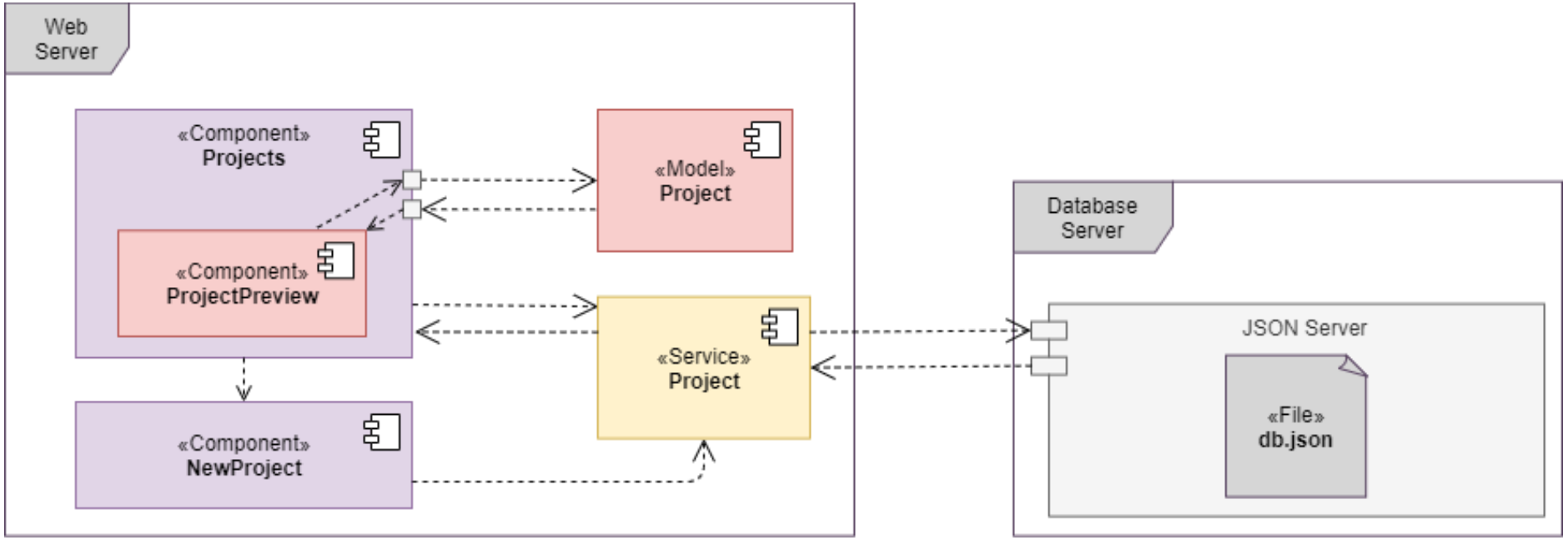
L'application, dans l'état actuel, peut être décomposée selon les composants et modules représentés dans le schéma suivant :



La partie web serveur comporte un AppModule, module de base de l'application, qui lui-même comporte quatre modules principaux : CoreModule, PagesModule, SharedModule et ThemeModule (pour plus d'informations sur leur rôle, rendez-vous dans la partie documentation du code source avec le fichier *Projet-GL52_BonnesPratiquesAngular*).

Ensuite, on retrouve à part, la partie Base de données, avec JSON SERVER et le fichier db.json.

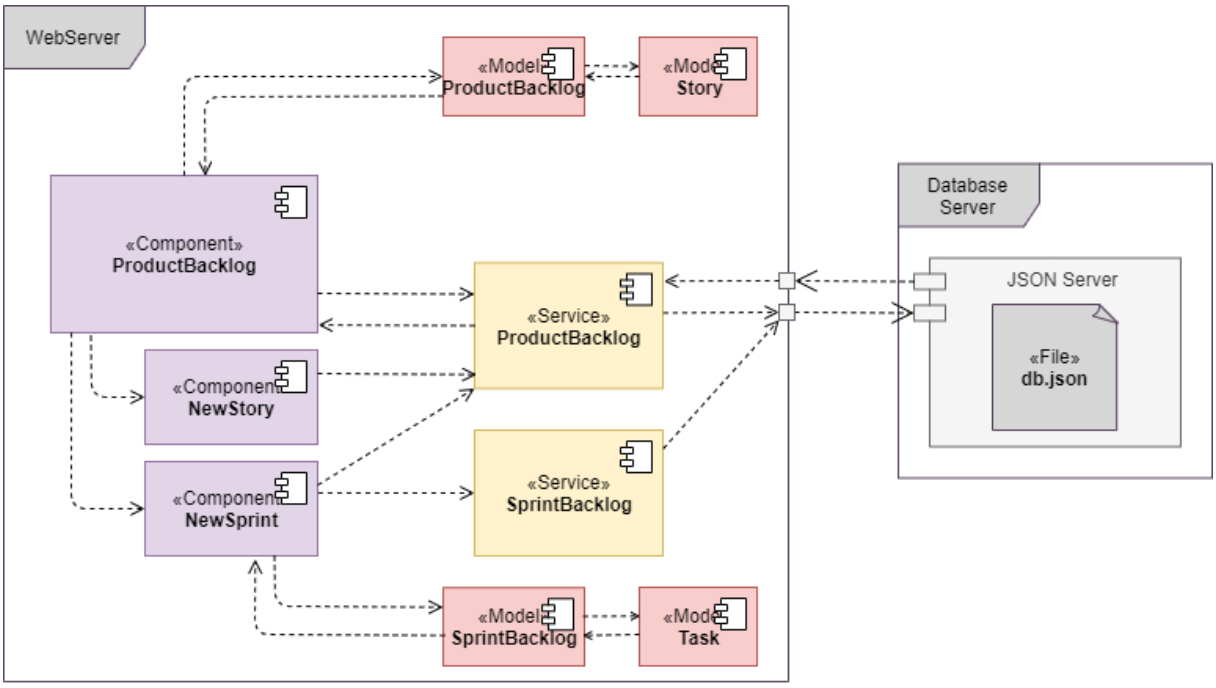
Le diagramme de composants suivant montre les interactions et les composants relatifs à la partie des projets.



Le composant *Projects* gère l'affichage de tous les projets. Il récupère antérieurement les données à afficher en faisant appel au service *Project* qui va récupérer les données de la base. Ces données sont alors formattées sous la forme du modèle *Project*. Il retransmet ensuite les données, projet par projet, au composant *ProjectPreview* qui s'occupe de l'affichage d'un projet.

Le composant *Projects* permet aussi d'accéder à la page d'ajout de projet, représentée par le composant *NewProject*. Celui-ci fait appel au service Projet lors de l'enregistrement d'un nouveau projet à la base de données.

Le diagramme de composants suivant montre les interactions et les composants relatifs à la partie du backlog produit d'un projet. Cette page et toutes ces options sont disponibles à partir du moment où l'utilisateur a cliqué sur l'affichage d'un projet.

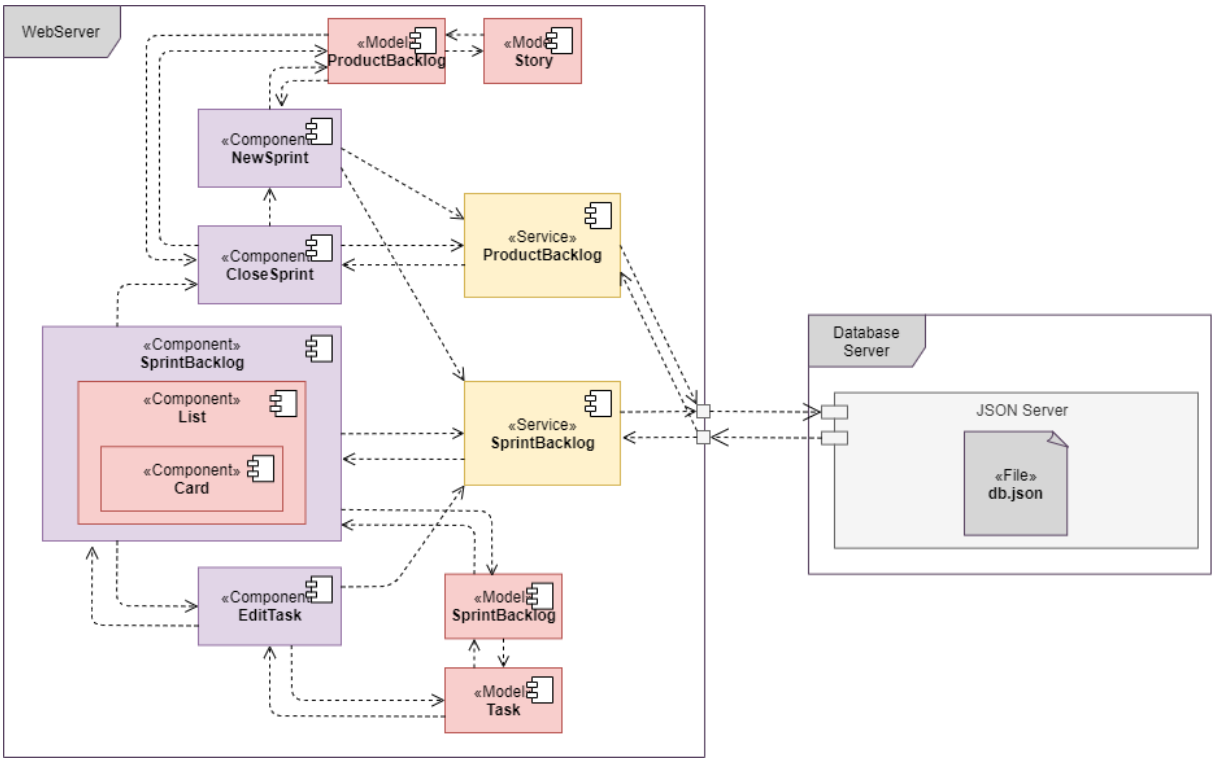


La composante principale est le composant *ProductBacklog* qui gère l'affichage du backlog produit d'un projet. Il fait appel au service *ProductBacklog* pour obtenir les informations (stories…) d'un projet. Il formatte ensuite les données suivant le formatage proposé par le modèle *ProductBacklog*, qui lui-même, met en forme les données concernant les stories suivant le modèle *Story*. Il s'ocuppe ensuite d'afficher ces données.

Si l'utilisateur est connecté en tant que PRODUCTOWNER, sur cette page il aura également accès à un bouton, lui permettant l'accès au composant *NewStory*, gérant l'ajout de nouvelle story au backlog produit. Ce composant fera appel au service *ProductBacklog* pour mettre à jour le backlog produit courant comportant la nouvelle story.

En tant que SCRUMMASTER, l'utilisateur aura accès à un bouton permettant de débuter un nouveau sprint, s'il n'y en a pas déjà en cours, selon une story appartenant préalablement au backlog. Il aura alors accès au composant *NewSprint* qui s'occupe, à partir des données entrées par l'utilisateur, de créer un nouveau backlog de sprint formatté grâce au modèle *SprintBacklog*, et de l'enregistrer dans la base de données grâce au service *SprintBacklog*. Il met également à jour l'état de la story pour laquelle un sprint a démarré et met à jour la backlog produit grâce au service *ProductBacklog*.

Le diagramme de composants suivant montre les interactions et les composants relatifs à la partie du backlog de sprint du sprint courant. Ce composant est disponible à partir du moment où un sprint a été créé pour une story donnée.



Le composant *SprintBacklog* gère l'affichage du backlog de sprint. Il fait appel au service *SprintBacklog* pour obtenir les informations (tasks…) d'un projet. Il formatte ensuite les données suivant le formatage proposé par le modèle *SprintBacklog*, qui lui-même, met en forme les données concernant les tâches suivant le modèle *Task*. Il transmet ensuite ces objets aux composants de *List* qui affichent la liste des tâches via des composants *Card*.

Lors de l'édition d'une tâche par un membre de la *ScrumTeam*, on retrouve l'accès au composant *EditTask*, qui permet à l'utilisateur de modifier les informations de la tâche sélectionnée. Il modifie la tâche grâce aux méthodes du modèle *Task*. Enfin, il met à jour le backlog de sprint en envoyant le sprint courant modifié au service *SprintBacklog*.

Le SCRUMMASTER peut clôturer un sprint, lors de cette action, il a accès au composant *CloseSprint*, qui lui permet soit de directement clôturer le sprint, soit de le faire en transférant les tâches incomplètes à un nouveau sprint. Lors d'un transfert, l'utilisateur a accès à la liste des stories non-complétées et une fois la story du nouveau sprint choisie, il a accès au composant *NewSprint*, décrit précédemment. Enfin, lorsqu'il clôt le sprint, le composant fait appel au service *ProductBacklog* pour mettre à jour le backlog produit du projet avec la story comme complétée.